

Behavior and Confluence Analysis of \mathcal{M} -Adhesive Transformation Systems using \mathcal{M} -Functors

vorgelegt von
Dipl. Inform.
Maria Maximova
ORCID: 0000-0001-9275-806X

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Stephan Kreutzer
Gutachter: Prof. Dr. Uwe Nestmann
Gutachterin: Prof. Dr. Barbara König
Gutachterin: Prof. Dr. Julia Padberg
Gutachterin: Dr. Claudia Ermel

Tag der wissenschaftlichen Aussprache: 26. März 2019

Berlin 2019

ABSTRACT

For modeling dynamic systems, various graphical modeling formalisms exist. In particular, rule-based graph transformation formalisms have proven to be adequate, both to capture system behavior and system adaptations. For some graph transformation-based formalisms there already exist well-established tools, enabling modelers to analyze important semantical properties of considered transformation systems. Yet, there are many further adaptations and variants of such formalisms which, on the one hand, ease their application in certain contexts but, on the other hand, require new analysis techniques and tools. To avoid the implementation of further formalism-specific analysis tools for all these formalism variants, it would be helpful to develop and use a formal mapping of the considered formalisms to a kernel formalism for which analysis techniques and tools do already exist.

Therefore, in this thesis, for a broad class of transformation-based modeling formalisms, we have introduced a technique to relate two formalisms with respect to their semantical properties of interest. The provided connection enables the usage of the analysis methods and tools available for the target formalism also for the source formalism. As a class of considered formalisms we have chosen \mathcal{M} -adhesive transformation systems based on \mathcal{M} -adhesive categories, which share common technical properties and include many relevant notions used for the modeling of system behavior and adaptation. The investigated semantical properties include behavioral equivalence, (local) confluence, termination, functional behavior as well as parallel and sequential independence of transformations.

To establish the described formal relationship between different \mathcal{M} -adhesive transformation systems, we have developed an abstract framework of \mathcal{M} -functors. This framework is introduced first for transformation systems containing only rules *without* nested application conditions and is then extended to rules *with* nested application conditions. This extension is non-trivial concerning the technical aspects and is most important for transformation systems in practice.

The developed abstract framework is instantiated for two relevant modeling formalisms. We related both, hypergraph transformation systems and Petri net transformation systems with individual tokens with typed attributed graph transformation systems. The instantiation is executed by providing concrete \mathcal{M} -functors from the \mathcal{M} -adhesive category of the source transformation system to the \mathcal{M} -adhesive category of the target transformation system and by verifying sufficient technical properties required by the developed theory for the involved categories and the constructed concrete \mathcal{M} -functors. The common target transformation system of typed attributed graphs is a reasonable choice since, for example, the well-established tool AGG, purpose-built for typed attributed graph transformation systems, allows for modeling, simulation, and, in particular, critical pair analysis, which is the first step towards the confluence analysis.

Komplexe dynamische Systeme können unter Einsatz von graphischen Formalismen adäquat modelliert werden. Insbesondere regelbasierte Graphtransformationformalismen werden erfolgreich eingesetzt, um sowohl das Systemverhalten als auch Systemveränderungen zu erfassen. Für einige Graphtransformationformalismen wurden bereits Werkzeuge zur Analyse von semantischen Eigenschaften entwickelt. Weiterführende Entwicklungen von Graphtransformationformalismen vereinfachen einerseits ihre Anwendung in verschiedenen Kontexten, erfordern jedoch andererseits neue Analysetechniken und Analysewerkzeuge. Um die Entwicklung von solchen zusätzlichen formalismusspezifischen Werkzeugen zu vermeiden, ist die Erforschung und Bereitstellung von formalen Abbildungen solcher Formalismen in einen Kernformalismus mit bereits existierender adäquater Werkzeugunterstützung hilfreich.

In dieser Arbeit haben wir für eine große Klasse von transformationsbasierten Modellierungformalismen eine Technik entwickelt, um Instanzen von zwei Formalismen (einem Quell- und einem Zielformalismus) miteinander in Beziehung zu setzen. Diese Beziehung garantiert, dass bestimmte relevante semantische Eigenschaften der Instanz des Zielformalismus (wie Verhaltensäquivalenz, (lokale) Konfluenz, Terminierung, funktionales Verhalten sowie parallele und sequentielle Unabhängigkeit von Transformationen) auch von der Instanz des Quellformalismus erfüllt werden. Die entwickelte Technik ermöglicht somit den Einsatz der Analysewerkzeuge des Zielformalismus auch für Instanzen des Quellformalismus. Mit unserer entwickelten Technik setzen wir Formalismen aus der wohletablierten Klasse von \mathcal{M} -adhäsiven Transformationssystemen in Beziehung, einer Klasse von Formalismen die zahlreiche relevante graphische Modellierungformalismen umfasst. Diese Formalismen basieren auf \mathcal{M} -adhäsiven Kategorien, die durch eine Liste von fundamentalen technischen Eigenschaften charakterisiert sind und relevante Begrifflichkeiten für die Modellierung und Analyse von Systemverhalten und Systemveränderungen bereitstellen.

Aus formaler Sicht stellen wir die Beziehung zwischen zwei \mathcal{M} -adhäsiven Transformationssystemen mit Hilfe von sogenannten \mathcal{M} -Funktoeren her. Die hierfür benötigte Theorie entwickeln wir im ersten Schritt für Transformationssysteme *ohne* verschachtelten Anwendungsbedingungen und verallgemeinern diese nachfolgend für Transformationssysteme *mit* verschachtelten Anwendungsbedingungen. Der Erweiterungsschritt ist formal anspruchsvoll jedoch von großer praktischer Bedeutung, da die Verwendung von verschachtelten Anwendungsbedingungen die Ausdrucksmächtigkeit der Transformationssysteme erhöht und eine breite Verwendung bei der Modellierung findet.

Wir demonstrieren die Anwendbarkeit unseres auf der Ebene von \mathcal{M} -adhäsiven Transformationssystemen entwickelten abstrakten Ansatzes durch seine Instanziierung für konkrete relevante Modellierungformalismen. Wir setzen sowohl Hypergraphtransformationssysteme als auch Petri-Netz-Transformationssysteme mit individuellen Marken jeweils mit den entsprechenden getypten attribuierten Graphtransformationssystemen in Beziehung. Für diese Instanziierung definieren wir zunächst jeweils einen konkreten \mathcal{M} -Funktor zwischen der \mathcal{M} -adhäsiven Kategorie des Quelltransformationssystems und der \mathcal{M} -adhäsiven Kategorie des Zieltransformationssystems und verifizieren im Anschluss, dass der definierte \mathcal{M} -Funktor in Verbindung mit den involvierten \mathcal{M} -adhäsiven Kategorien die durch die Theorie vorgegebenen hinreichenden Bedingungen erfüllt.

ACKNOWLEDGEMENTS

It was a very big challenge for me to finish this thesis. That is why I am very grateful to my supervisors, colleagues, family, and friends for their permanent support and encouragement.

First of all, I want to thank Prof. Dr. Hartmut Ehrig, who, already in my first years as a student, inspired me for the field of theoretical computer science, woke my interest for scientific work, believed in my skills, and gave me an opportunity to learn from his long-time experience as well as from his enthusiasm and precision when doing research. I am very sad, that Hartmut cannot read the final version of my thesis.

I also want to sincerely thank Prof. Dr. Uwe Nestmann for continuing my supervision after retirement of Hartmut as well as for giving me an opportunity to work in his group and to do teaching in the field of theoretical computer science as a student assistant and as a research assistant.

I am especially grateful to Dr. Claudia Ermel for being, before and after retirement of Hartmut, my co-supervisor, for supporting me in different kinds of administrative matters, for motivating me, for being always positive, for giving me feedback, and for many valuable discussions on theoretical and practical aspects of my work.

Apart of that, I am very thankful to my secondary reviewers and co-supervisors, Prof. Dr. Barbara König and Prof. Dr. Julia Padberg, for accepting the task of examining my thesis and their valuable support and positive feedback.

Furthermore, I would also like to thank my former colleagues from the TFS research group for a great time we spent together while doing research and teaching. Many thanks go to Olga Runge for moral support, for fruitful discussions, and for help concerning AGG. Additionally, I thank Dr. Tony Modica, Dr. Karsten Gabriel, Dr. Frank Hermann, and Hanna Schölzel for companionship, for moral support, for sharing the daily working life, and for feedback provided in many talks and discussions. Moreover, I am very thankful to Käte Schlicht for her help in all administrative situations.

I would also like to sincerely thank my current boss and my current colleagues from the Hasso Plattner Institute in Potsdam. I am very grateful to Prof. Dr. Holger Giese for giving me an opportunity to finish my thesis in his group while working in parallel on a DFG project and for supporting me without putting me under pressure. Furthermore, I want to thank Dr. Leen Lambers for her detailed constructive feedback in the final phase of my work and for continuously encouraging me to go on with the completion of my thesis. Last but not least, I would like to thank my colleagues Lucas Sakizloglou, Sona Ghahremani, Jochen Hänsel, Johannes Dyck, Dr. Dominique Blouin, Kerstin Miers, Thomas Brand, Dr. Thomas Vogel, Matthias Barkowsky, He Xu, Christian Zöllner, and Christian Adriano for their great companionship, friendship, and moral support in difficult times.

Finally, I want to express my special thanks to my husband Dr. Sven Schneider for supporting me in every situation, for bearing me in stressful times, for thousands of discussions about my work, for proofreading of several parts of the thesis, and for everything else he has done for me. In addition to this, I am very grateful to my dear parents for believing in me all of the time, for giving me energy to finish this thesis, and for cheering me up every time I needed it.

CONTENTS

i	INTRODUCTION AND FOUNDATIONS	1
1	INTRODUCTION	3
1.1	Setting	3
1.2	Problems and Objectives	7
1.3	Proposed Solution	8
1.4	Structure of the Thesis	10
1.5	Publications	12
2	FOUNDATIONS	15
2.1	Algebraic Graph Transformation Approach	15
2.2	\mathcal{M} -Adhesive Transformation Systems	18
2.2.1	Basic Concepts of \mathcal{M} -Adhesive Transformation Systems	18
2.2.2	Relevant Semantical Properties in the Framework of \mathcal{M} -Adhesive Transformation Systems	21
2.3	Advanced Concepts for Rule-Based Transformations and Their Analysis . .	31
2.3.1	Transformations with Nested Application Conditions	31
2.3.2	Local Confluence Analysis for Transformations with Nested Application Conditions	36
2.4	Concrete Instantiations of \mathcal{M} -Adhesive Transformation Systems	41
2.4.1	Typed Attributed Graph Transformation Systems	41
2.4.2	Hypergraph Transformation Systems	47
2.4.3	PTI Net Transformation Systems	61
ii	BEHAVIOR AND CONFLUENCE ANALYSIS OF \mathcal{M}-ADHESIVE TRANSFORMATION SYSTEMS	79
3	BEHAVIOR ANALYSIS OF \mathcal{M}-ADHESIVE TRANSFORMATION SYSTEMS USING \mathcal{M}-FUNCTORS	81
3.1	Functorial Behavior Transfer between \mathcal{M} -Adhesive Transformation Systems	81
3.1.1	Translation and Creation of Transformations without Nested Application Conditions using \mathcal{M} -Functors	82
3.1.2	Translation and Creation of Transformations with Nested Application Conditions using \mathcal{M} -Functors	89
3.2	Bisimulation-Based Behavior Analysis of \mathcal{M} -Adhesive Transformation Systems	93
3.2.1	\mathcal{F} -Bisimilarity	93
3.2.2	\mathcal{F} -Transfer of Bisimilarity	96
4	CONFLUENCE ANALYSIS OF \mathcal{M}-ADHESIVE TRANSFORMATION SYSTEMS USING \mathcal{M}-FUNCTORS	99
4.1	\mathcal{F} -Transfer of Local Confluence for Transformations without Nested Application Conditions	99
4.2	\mathcal{F} -Transfer of Local Confluence for Transformations with Nested Application Conditions	105

4.3	\mathcal{F} -Transfer of Termination, Confluence, and Functional Behavior	118
iii	APPLICATION TO HYPERGRAPH TRANSFORMATION SYSTEMS	121
5	FUNCTOR BETWEEN CATEGORIES OF HYPERGRAPHS AND TYPED ATTRIBUTED GRAPHS	123
5.1	Typed Attributed Graphs over the Hypergraph Type Graph <i>HGTG</i>	123
5.2	Translation of Hypergraphs into Typed Attributed Graphs	124
6	BEHAVIOR ANALYSIS OF HYPERGRAPH TRANSFORMATION SYSTEMS	129
6.1	Translation of Hypergraph Transformations	129
6.2	Creation of Hypergraph Transformations	137
6.3	\mathcal{F}_{HG} -Transfer of Bisimilarity for Hypergraph Transformation Systems . . .	142
7	CONFLUENCE ANALYSIS OF HYPERGRAPH TRANSFORMATION SYSTEMS	149
7.1	Local Confluence of Hypergraph Transformation Systems without Nested Application Conditions	149
7.2	Local Confluence of Hypergraph Transformation Systems with Nested Application Conditions	152
7.3	ACG-Based Local Confluence Analysis of Hypergraph Transformation Systems	155
7.4	Termination, Confluence, and Functional Behavior of Hypergraph Transformation Systems	168
iv	APPLICATION TO PTI NET TRANSFORMATION SYSTEMS	171
8	RESTRICTED FUNCTOR BETWEEN CATEGORIES OF PTI NETS AND TYPED ATTRIBUTED GRAPHS	173
8.1	Typed Attributed Graphs over the PTI Net Type Graph <i>PNTG</i>	173
8.2	Translation of PTI Nets into Typed Attributed Graphs	174
9	BEHAVIOR ANALYSIS OF PTI NET TRANSFORMATION SYSTEMS	179
9.1	Translation of PTI Net Transformations	179
9.2	Creation of PTI Net Transformations	185
9.3	\mathcal{F}_{PTI} -Transfer of Bisimilarity for PTI Net Transformation Systems	191
10	CONFLUENCE ANALYSIS OF PTI NET TRANSFORMATION SYSTEMS	193
10.1	Local Confluence of PTI Net Transformation Systems without Nested Application Conditions	193
10.2	Local Confluence of PTI Net Transformation Systems with Nested Application Conditions	196
10.3	ACG-Based Local Confluence Analysis of PTI Net Transformation Systems .	199
10.4	Termination, Confluence, and Functional Behavior of PTI Net Transformation Systems	209
v	FUNCTOR DECOMPOSITION STRATEGY FOR VERIFICATION OF REQUIREMENTS AND ITS APPLICATION	211
11	FUNCTOR DECOMPOSITION STRATEGY FOR VERIFICATION OF REQUIREMENTS	213
11.1	Verification of Requirements for the General Theory of \mathcal{M} -Functors	214
11.2	Verification of Requirements for General Theory of Restricted \mathcal{M} -Functors .	216

12 APPLICATION OF FUNCTOR DECOMPOSITION STRATEGY FOR REQUIREMENT VERIFICATION TO HYPERGRAPH AND PTI NET TRANSFORMATION SYSTEMS	221
12.1 Application to Hypergraph Transformation Systems	221
12.2 Application to PTI Net Transformation Systems	222
vi RELATED WORK, CONCLUSION, AND FUTURE WORK	225
13 RELATED WORK	227
13.1 Foundations for Formal Transformations	227
13.2 Formalisms and Tools for Modeling and Analysis of Struct. and Behavior .	228
13.2.1 Graphs	228
13.2.2 Hypergraphs	232
13.2.3 Petri nets	234
13.3 Existing Frameworks for Translation of Different Formalisms	237
13.3.1 Institutions	238
13.3.2 Triple Graph Grammars	239
14 CONCLUSION	241
15 FUTURE WORK	247
15.1 Extension of the Framework	247
15.2 Application of the Framework	248
BIBLIOGRAPHY	251
vii APPENDICES	267
A BASICS OF CATEGORY THEORY	269
B DETAILED PROOFS FOR GENERAL THEORY	275
C DETAILED PROOFS FOR HYPERGRAPH TRANSFORMATION SYSTEMS	311
D DETAILED PROOFS FOR PTI NET TRANSFORMATION SYSTEMS	361
viii LISTS	401
List of Definitions	401
List of Theorems	403
List of Lemmas	405
List of Facts	407
List of Remarks	408
List of Examples	410
List of Figures	411

Part I

INTRODUCTION AND FOUNDATIONS

INTRODUCTION

The modeling and formal analysis of dynamic systems is one of the big challenges in computer science. In this thesis, we focus on transformation systems for different kinds of formalisms and their analysis supported by tools on a formal foundation. Subsequently, we provide a brief overview of the considered setting with the problems and objectives followed by our approach.

1.1 SETTING

The modeling of dynamic systems including their adaptation to a changing environment occurs in various domains such as computer supported cooperative work, multi-agent systems or mobile and distributed networks.

A broad range of, often graphical, formalisms have been developed in the past for the modeling of such systems. Some well-known examples are different kinds of Petri nets [252], automata [125], and statecharts [140]. Also, graph-based formalisms have been developed for other purposes such as e.g. the definition of the abstract syntax of visual models based on UML [236] and business process models using BPMN [237]. Each of the mentioned formalisms defines a set of graphical models and, in most cases, behavioral semantics by means of single step relations resulting in state spaces.

GRAPH TRANSFORMATION APPROACH

The adaptation of graph-based formalisms can be defined by applying transformation rules on the models of these formalisms. Such adaptations are to be understood in this context as local rule-based modifications defining additions and removals of substructures and are central to the graph transformation approach. Graph transformation systems and their semantical properties are intensively studied since the early 1970s [79]. The double pushout (DPO) approach to graph transformation (serving as a basis for our approach) and how it is used to modify graphs and graph-like formalisms is introduced in more detail in Chapter 2.

Graph transformation systems have been extended in various directions to facilitate their modeling and analysis. For example, in a formalism of typed attributed graphs, data has been integrated directly using attributes [88]. Another major step was the development of conditional rule applicability [77] based on application conditions. Intuitively, a rule is applicable if a certain pattern to be modified exists in the host graph that has to satisfy, in addition, all the application conditions associated with this rule. Application conditions for graph transformation rules were first introduced in [77], while negative application conditions (short NACs), representable in a graphical way, were first considered in [139]. In the literature, there are different kinds of application conditions as for instance NACs [139, 143], positive application conditions (short PACs) [87], and nested application conditions [135, 136, 96]. Nested application conditions can contain a combination of PACs and NACs as well as can include an arbitrary level of nesting. In this thesis, we employ nested application conditions, which are equivalent to first-order

logic on graphs and are strictly more expressive than NACs and PACs. It is well-known that the usage of application conditions and explicit data in form of attributes does not increase the expressive power of transformation systems (which are already Turing powerful without these features) since both of these features can be encoded by adding appropriate further structure to the rules and to the graphs. Still their usage is advantageous since it results in simpler rules and smaller intermediate graph structures.

\mathcal{M} -ADHESIVE TRANSFORMATION SYSTEMS

A unified theory for a broad class of transformation systems has been developed with multiple intermediate steps ultimately leading to \mathcal{M} -adhesive transformation systems based on categorical constructions defining transformation at an abstract level (see Chapter 2 for a more in depth coverage of earlier stages of development). An \mathcal{M} -adhesive transformation system consists of an \mathcal{M} -adhesive category (a category with certain general technical properties) along with a set of transformation rules and, for technical purposes discussed in Section 2.2, a class \mathcal{M} of monomorphisms [94]. The handling of the DPO approach at the categorical level, encompassing the general technical properties, has proved to be suitable to obtain various results such as the Local Church-Rosser, Parallelism, Concurrency, Embedding, Extension, and Local Confluence Theorems [88]. The instantiation of \mathcal{M} -adhesive transformation systems for concrete formalisms such as transformation systems over e.g. (typed attributed) graphs, hypergraphs, and Petri nets requires the verification of these general technical properties.

HYPERGRAPHS AND HYPERGRAPH TRANSFORMATION SYSTEMS

Hypergraphs generalize ordinary graphs by allowing each edge to have any finite number of sources and targets (instead of precisely one) additionally assuming an order on these elements. Obviously, hypergraphs and ordinary graphs are equivalently expressive because each ordinary graph is a hypergraph and each hypergraph can be encoded as an ordinary graph (where the ordering is encoded in additional labels). Hypergraphs are used in various settings in mathematics [9, 120, 253], biology [168, 113], and chemistry [27, 178, 113]. In computer science, hypergraphs have been used e.g. for the modeling of online social networks to describe the relations for friendship, cooperation, and business contacts [226, 286, 225] or for the description of inter-networking systems containing components and communication ports between them, which represent the environment of the components [149, 148, 295].

Hypergraph transformation systems have been used in many application domains to model the behavior of distributed or concurrent systems [271, 147, 295, 188], to model machine learning processes [279], and, furthermore, hyperedge replacement systems, as restricted form of hypergraph transformation systems, can be seen as graphical context-free Chomsky grammars [181]. A more detailed discussion of the formal hypergraph rewriting techniques is contained in Subsections 2.4.2 and 13.2.2.

PETRI NETS AND PETRI NET TRANSFORMATION SYSTEMS

The formalism of Petri nets is a model for the concurrent execution of systems allowing for an efficient analysis of various important domain-specific semantical properties such as deadlocks, livelocks, and invariants. The basic formalism of Petri nets handles data at an abstract level since tokens in a common place are indistinguishable. Petri nets have

been extended to model also data handling e.g. in colored Petri nets [161] and, on a much more general level, in algebraic high-level nets (short AHL nets) [82, 153].

The reconfiguration of Petri nets, as for graphs, is of interest in various domains for the controlled model adaptation as discussed before. A first approach to relate Petri nets and graph transformation systems has been proposed by Kreowski in [180] where the firing behavior of Petri nets marked by tokens attached to their places by edges is implemented by certain graph transformation rules. In this unified setting as well as in the setting where the tokens are indistinguishable, it is then possible to apply structure-changing transformation rules to a Petri net enabling its stepwise reconfiguration [92, 260]. The analysis of structural changes of Petri nets executed by means of transformations resulted from these developments [20]. In this thesis, we rely on Petri nets with individual tokens (short PTI nets), as they offer a possibility for token reconfiguration (by means of transition firing) also at the abstract level of the DPO approach allowing for a unified handling on the foundation of \mathcal{M} -adhesive transformation systems. For a more detailed discussion of Petri nets and their transformation see Subsections 2.4.3 and 13.2.3.

SEMANTICAL PROPERTIES

When applying modeling formalisms based on graph transformation, users are often interested in the analysis of general and domain-specific semantical properties. The possibility for an adequate tool support allowing for such analysis with minimal user interaction is limited, however, by the inherent undecidability of many useful semantical properties in Turing powerful formalisms such as transformation systems. Consequently, analysis algorithms are often sound (every statement on the satisfaction is correct) and incomplete (a statement is not always derived) for these formalisms.

In this thesis, we focus on the semantical properties that are widely accepted to be relevant for a broad range of applications of transformation systems such as behavioral equivalence, (local) confluence, termination, functional behavior as well as parallel and sequential independence of transformations.

Rule-based transformation has always been concerned with the property of confluence stating that the order of the rules to be applied is not relevant for an ultimately obtained result. More formally, confluence is satisfied if for every given object of the underlying category and every two sequences of transformations starting in this object, there are two further sequences of transformations continuing the former sequences to a common object of the underlying category. Such confluence analysis is relevant for many formalisms allowing for parallel execution: considering the token game of Petri nets, conflicting firing of transitions is related. In fact, these domain-specific conflicts turn out to be conflicts in the transformation-based setting of PTI nets as well when rules are used to encode the firing behavior. Also, since graph transformation systems are inspired by term rewriting systems, the confluence property also ensures that it is possible for a given term to compute an equivalent normal form in finite time without having to employ a certain derivation strategy [57]. Moreover, confluence guarantees the so-called functional behavior in the sense that unique results are obtained by the computations. However, unresolvable conflicts of rule applications (leading altogether to the non-confluence of a system) indicate the existence of “true non-determinism” [144] in a system. In bigger applications, analysis may reveal that isolated components exhibit functional behavior (such as local computations in a distributed network). In these cases, the confluence of

such components can be used to verify other relevant semantical properties of the entire system.

Independent from the transformation setting, confluence is guaranteed for terminating systems that are locally confluent in the sense that transformation sequences of length one starting in a common object can be merged together [234]. In the context of graph transformation, this result is useful since local confluence can be analyzed by means of computation of so-called critical pairs and their subsequent analysis. Using critical pair analysis, we can statically detect conflicts between transformation steps of similar structure since, according to the Completeness of Critical Pairs Lemma [88], for each pair of conflicting transformation steps there is a critical pair representing this conflict in a minimal context (see Subsubsection 2.2.2.4 and Subsection 2.3.2 for a more in-depth introduction including the analysis of critical pairs).

The semantical property of termination is, as stated above, relevant for the analysis of confluence and functional behavior. Moreover, even in applications where functional behavior is not required, the termination or even the non-termination (e.g. in permanently running systems such as embedded systems) of a software component may be a desirable property to be enforced and, hence, analyzed. The analysis of termination is, of course, not limited to software components as it is also relevant in many other contexts such as e.g. business processes and parsing of visual languages.

The semantical property of parallel and sequential independence has a technical nature and is often considered in the context of local confluence analysis of transformation systems where the independence of any two transformation steps ensures the local confluence of the entire system. Moreover, this property helps to detect the conflicting and causally dependent rule applications during the analysis of a transformation system. We speak of parallel independence if two transformation steps can be applied in any order yielding the same result, while we speak of sequential independence if two transformation steps in a transformation sequence can be exchanged without affecting the overall result of the sequence [141, 88].

Behavioral equivalence has a different nature compared to the semantical properties introduced above since it does not provide by its own a property to be analyzed. It rather requires an additional system (e.g. a transformation system defining a certain reachability graph) that serves as a specification for the system under analysis. Behavioral equivalence then provides a definition of how these two systems are to be compared for equivalent behavior. Furthermore, for many interpretations, more basic equivalence properties can be derived from behavioral equivalence such as e.g. language equivalence. In the context of graph transformation systems, behavioral equivalence expresses for two systems intuitively that they can execute equivalent transformation steps from related graphs of these two systems. For many other formalisms such as e.g. process calculi that define system behavior based on transition systems, behavioral equivalence has been formalized using the notion of bisimulation.

Note that for the concrete formalisms of hypergraphs and PTI nets we do not discuss in this thesis further semantical properties specific to their application domains.

TOOLS

There is a multitude of tools with varying sets of features for the modeling, simulation, and analysis of different kinds of transformation systems.

Since the confluence property is of great importance in the context of our approach, we have chosen the tool AGG [292] for the application of our theory in this thesis. To the best of our knowledge, AGG (a purpose-built tool for typed attributed graph transformation systems) is the only graph transformation tool to this day supporting local confluence analysis via critical pairs computation for typed attributed graph transformation systems with NACs and PACs [292, 121]. The spectrum of further tools with comparable capabilities but different limitations is then discussed in detail in Chapter 13.

1.2 PROBLEMS AND OBJECTIVES

Various formalisms defining a semantical behavior of a system lack formal-based tool support for the analysis of semantical properties relevant for the system. In Figure 1 two systems, System 1 and System 2, are depicted. System 1 in the left column lacks tool-based support for the verification of Property 1, while System 2 in the right column can analyze Property 2 (corresponding to the Property 1 formulated in the language of System 2) using an appropriate tool support (Tool 2). In the following, we focus on the research question of how to use System 2 with Tool 2 on a formal level to analyze properties of System 1.

In this thesis, we rely on \mathcal{M} -adhesive transformation systems, which are a suitable abstraction mechanism for the description of different formalisms in the context of the DPO approach to graph transformation. For the considered formalisms, we allow for the usage of nested application conditions.

Even for formalisms in the domain of \mathcal{M} -adhesive transformation systems, there is often no suitable formal-based tool support for semantical analysis. For example, for hypergraph and Petri net transformation systems there are several general-purpose tools, in which hypergraph and Petri net transformation systems can be encoded. Besides AGG, e.g. the tools GROOVE [121], Henshin [69, 6], and SyGrAV [53, 290] offer the possibility to model hypergraphs and hypergraph transformation systems resp. Petri nets and Petri net transformation systems by converting them in an informal way into the corresponding typed attributed graph representation used by these tools. The drawback of this approach is the additional effort for the verification of the informal conversion procedure. Such a conversion procedure must suitably preserve and reflect syntax and semantics between two different formalisms. This idea was already used on an informal level for the implementation of net transformations in the *RON-tool* (a modeling, simulation, and analysis tool for algebraic higher-order nets) [291, 19] ensuring that the graph structure and the graph match morphisms for the translated rules calculated by AGG correspond to valid net structures and valid net match morphisms. This informal translation was the inspiration for our formal approach.

Besides the mentioned class of formalisms, we focus on semantical properties such as behavioral equivalence, (local) confluence, termination, functional behavior as well as parallel and sequential independence of transformations.

Our main objective is the development of a formal framework enabling tool-based support for the analysis of semantical properties for a transformation system defined according to another given formalism. This framework should (a) be verified on a formal level, (b) provide a clean interface that can be used for a broad range of formalisms, and (c) be sufficiently powerful to be able to transfer a reasonable set of semantical properties of interest. The

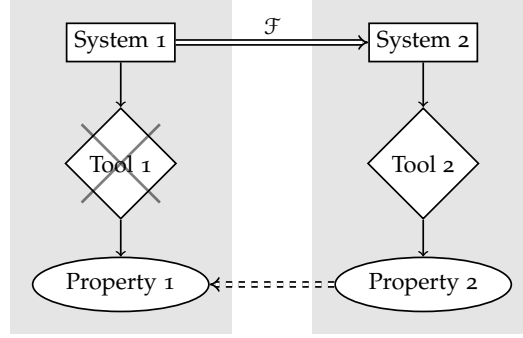


Figure 1: Abstract perspective on the research problem and the proposed solution

applicability of the framework must be evaluated by applying it to different instantiations of \mathcal{M} -adhesive transformation systems. Moreover, since further properties may be important for certain domains in the future, the extendability of the framework regarding the semantical properties to be analyzable must be guaranteed.

In the next section, we discuss in detail our approach to achieve the previously mentioned main objective.

1.3 PROPOSED SOLUTION

As a solution for the stated research problem, we introduce a technique to relate different formalisms with respect to semantical properties of interest. The connection provided allows for the transfer of semantical properties among the involved formalisms and therefore enables the analysis techniques and tools available for the target formalism also for the source formalism.

For an abstract perspective on our proposed solution, consider again [Figure 1](#). In this figure, as already mentioned before, System 1 lacks tool-based support (since no suitable Tool 1 is available) for the analysis of Property 1. Using our approach, we relate System 1 with System 2 and Property 1 with Property 2 using a functor \mathcal{F} . This relationship between the systems and semantical properties then guarantees that the judgment of Tool 2, which decides whether System 2 satisfies Property 2, implies the desired judgment, namely, whether System 1 satisfies Property 1. Therefore, we bypass the missing domain-specific tool-based support of System 1 using Tool 2 from a different domain. In fact, System 1 and System 2 are behavioral equivalent in the sense of bisimilarity when System 2 is constructed from System 1 using a functor \mathcal{F} according to our approach.

In the following, we discuss our proposed solution (in terms of our formal approach), i.e., the chosen systems, semantical properties, and tools as well as the elementary steps of the proposed solution in more detail. For the systems considered, as already mentioned before, we restrict ourselves to the framework of \mathcal{M} -adhesive transformation systems, which (a) cover a broad class of transformation systems and (b) restrict the transformation systems by requiring the satisfaction of certain general technical properties that allow for the derivation of various reasonable results. To enhance the descriptive expressiveness of transformation systems, we allow for nested application conditions, which control the applicability of transformation rules. In our abstract framework, we focus on

semantical properties that are of general interest to many different transformation systems such as behavioral equivalence, (local) confluence, termination, functional behavior as well as parallel and sequential independence of transformations. The analysis of such semantical properties for a concrete formalism is then covered at the instantiation level of our abstract framework.

In Part ii we develop our abstract formal framework as follows. As a first step, for two given \mathcal{M} -adhesive transformation systems $AS_1 = ((\mathbf{C}_1, \mathcal{M}_1), P)$, $AS_2 = ((\mathbf{C}_2, \mathcal{M}_2), \mathcal{F}(P))^1$, we define in our abstract framework \mathcal{M} -functors \mathcal{F} as functors of type $\mathbf{C}_1 \rightarrow \mathbf{C}_2$ with two further specific technical properties [207]. As usual, an \mathcal{M} -functor defines a translation of objects and morphisms from \mathbf{C}_1 to \mathbf{C}_2 . The two additional technical properties of an \mathcal{M} -functor (see Definition 41) then guarantee that transformation steps from AS_1 are properly translated into the corresponding transformation steps in AS_2 . As a second step, we determine additional sufficient properties (see Definition 42) to be satisfied by an \mathcal{M} -functor, guaranteeing the creation of transformation steps in AS_1 from the corresponding transformation steps in AS_2 [207]. This extension is sufficient for the transfer of behavior among the two considered transformation systems. As a third step, we include the handling of nested application conditions [211]. The effect on the transfer of direct transformations with nested application conditions results in further sufficient properties to be satisfied by an \mathcal{M} -functor. Finally, we consider the semantical property of confluence to be analyzed and introduce further sufficient confluence-specific properties that an \mathcal{M} -functor has to fulfill. See also Figure 82 in the conclusion for an overview showing for the semantical properties of interest the sufficient properties (also called *requirements* in the following) they depend on.

Based on this fundamental framework, we consider previously mentioned semantical properties to be analyzed such as (local) confluence, termination, functional behavior as well as parallel and sequential independence of transformations. For each of these semantical properties, we ensure that the framework is appropriate (by containing a suitable set of sufficient properties) for the transfer of analysis results from System 2 to System 1.

Our formal framework satisfies the general requirements stated in our main objective in Section 1.2 since we have (a) verified our results rigorously, (b) provided a clean interface based on the list of sufficient properties to be verified for the functor at hand, and (c) considered a reasonable set of semantical properties. Also, based on our experience with the properties considered, we are confident that further semantical properties can be integrated at acceptable cost and effort.

In Parts iii and iv, we apply our abstract framework to two concrete formalisms enabling the formal-based tool analysis of the discussed semantical properties in two new domains. In both cases, we employ the well-known tool AGG [292] because, to the best of our knowledge, it is the only tool capable to compute critical pairs for typed attributed graph transformation systems with NACs and PACs. Based on the computation of critical pairs, it is then possible to analyze (local) confluence of a considered transformation system. However, for these two new domains, further tool-based analysis is desirable as discussed in detail in Chapter 15.

¹ An \mathcal{M} -adhesive transformation system $AS = ((\mathbf{C}, \mathcal{M}), P)$ consists of an \mathcal{M} -adhesive category $(\mathbf{C}, \mathcal{M})$ where \mathbf{C} is an underlying category and \mathcal{M} is a class of monomorphisms, and a set of transformation rules P . For more details see Subsection 2.2.1.

Firstly, in Part [iii](#), we instantiate the \mathcal{M} -functor \mathcal{F} by providing the concrete functor \mathcal{F}_{HG} . This functor translates hypergraphs and hypergraph morphisms from the category **HyperGraphs** into typed attributed graphs and typed attributed graph morphisms from the category **AGraphs**_{HGTG} over the hypergraph specific type graph $HGTG$. We instantiate all of the theorems on the transfer of analysis results by verifying that the functor \mathcal{F}_{HG} satisfies the required sufficient properties mentioned before.

Secondly, in Part [iv](#), we introduce another instantiation of our theoretical framework allowing to relate PTI nets and typed attributed graphs. We consider the \mathcal{M} -adhesive transformation system of PTI nets where rule applications for transition firing with non-injective match morphisms may produce semantically incorrect transformation steps since tokens may be lost due to their merging. The restriction of the entire transformation system and, hence, of the underlying category to injective morphisms is out of question since the resulting category would not be \mathcal{M} -adhesive. Our solution, also introduced in Part [ii](#), is the definition of restricted \mathcal{M} -functors and the adaptation of our formal framework to this special case. The restricted \mathcal{M} -functor $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$ is then defined on all objects but only on injective morphisms, i.e., the fragment that is used for the intended behavior of PTI net transformation systems. For this variant of the abstract theory, we introduce the concrete restricted \mathcal{M} -functor \mathcal{F}_{PTI} as an instantiation. This functor translates PTI nets and their injective morphisms from the category **PTINet** into the corresponding typed attributed graphs and their morphisms from the category **AGraphs**_{PNTG} over the Petri net specific type graph $PNTG$. As for the \mathcal{M} -functor \mathcal{F}_{HG} , we instantiate the abstract theoretical results by verifying the list of sufficient properties adapted to the setting of restricted \mathcal{M} -functors.

Finally, in Part [v](#), we develop a *functor decomposition strategy* that is an alternative strategy for the verification of sufficient properties required by the general theory of (restricted) \mathcal{M} -functors. This strategy is based on a decomposition of the considered functor and divides the requirement verification process into two essential steps: (a) the definition of a category equivalence (directly based on the functor at hand) between the source category of the functor and the subcategory of the target category containing only the objects and morphisms that are images of the considered functor and (b) the verification of the sufficient properties only for the inclusion functor between the considered subcategory of the target category and the entire target category. For a restricted functor, this strategy works analogously considering the restriction of the involved categories to injective morphisms only. Concerning our application to the concrete functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} , this functor decomposition strategy simplifies the requirement verification task because the inclusion functors can be handled more easily also often allowing for proofs in a categorical setting.

1.4 STRUCTURE OF THE THESIS

This thesis is structured as follows.

- **Part i (Introduction and Foundations):**

In the remainder of this part, we introduce in Chapter [2](#) foundations relevant for the subsequent parts. In particular, we first recall in Section [2.1](#) the theory of algebraic graph transformation. Then, in Section [2.2](#), we introduce \mathcal{M} -adhesive transformation systems as a class of formalisms we are focusing on in our approach and, in

addition, semantical properties that are relevant for analysis in the context of \mathcal{M} -adhesive transformation systems. Afterwards, in Section 2.3, we extend the transformation approach to the handling of nested application conditions and, finally, provide in Section 2.4 the concrete instantiations of \mathcal{M} -adhesive transformation systems to typed attributed graph transformation systems, hypergraph transformation systems, and PTI net transformation systems.

- **Part ii (Behavior and Confl. Analysis of \mathcal{M} -Adhesive Transformation Systems):**

In this part, we introduce our theoretical approach to the transfer of behavior and semantical properties of interest in the following multiple steps. Firstly, in Section 3.1, we introduce our formal framework of (restricted) \mathcal{M} -functors, provide results for the transfer of direct transformations and sequences of direct transformations ensuring behavioral equivalence of the functor-related parts of the involved transformation systems, and, subsequently, extend these results concerning the behavior transfer to transformations with nested application conditions. Secondly, in Section 3.2, we use these foundations to transfer results on behavioral analysis based on bisimulations. Thirdly, in Sections 4.1 and 4.2, we consider the transfer of local confluence (including as a sufficient condition the transfer of parallel and sequential independence of transformations), one of the central semantical properties to be transferred between two transformation systems, for the case without or with nested application conditions, respectively. Finally, in Section 4.3, we discuss the analysis of further semantical properties, namely, termination, functional behavior, and confluence.

- **Part iii (Application to Hypergraph Transformation Systems):**

In this part, we instantiate the theoretical results from the previous part to hypergraph transformation systems without and with nested application conditions by introducing in Chapter 5 a concrete \mathcal{M} -functor \mathcal{F}_{HG} , which translates hypergraphs into typed attributed graphs, and by ensuring in Chapters 6 and 7 that \mathcal{F}_{HG} satisfies the requirements of the respective abstract theorems. This instantiation allows in Chapter 6 for the application of the results on the transfer of (direct) transformations and bisimilarity as well as in Chapter 7 for the application of the results on the transfer of local confluence, parallel and sequential independence, termination, confluence, and functional behavior to concrete hypergraph transformation systems without or with nested application conditions. Furthermore, in Section 7.3, we provide a detailed description for an analysis process of a concrete hypergraph transformation system using our theoretical framework instantiated for the \mathcal{M} -functor \mathcal{F}_{HG} and the tool AGG applied to the graph transformation system that is the functorial translation of the given hypergraph transformation system.

- **Part iv (Application to PTI Net Transformation Systems):**

Similarly to the previous part, we provide an instantiation of the theoretical results from Part ii to PTI net transformation systems by defining in Chapter 8 a restricted \mathcal{M} -functor \mathcal{F}_{PTI} , which translates PTI nets into typed attributed graphs. In Chapters 9 and 10, we verify that \mathcal{F}_{PTI} satisfies the adapted requirements of the abstract theorems from Part ii and enables, as for the \mathcal{M} -functor \mathcal{F}_{HG} , the application of the results on the transfer of (direct) transformations and bisimilarity (see Chapter 9) as well as on the transfer of local confluence, parallel and sequential independence, termination, confluence, and functional behavior (see Chapter 10) to concrete PTI

net transformation systems without or with nested application conditions. Using a different example, we also provide in Section 10.3 a description of an analysis process for a concrete PTI net transformation system using again our theoretical framework instantiated to the restricted \mathcal{M} -functor \mathcal{F}_{PTI} and the tool AGG applied to the graph transformation system that is the functorial translation of the given PTI net transformation system.

- **Part v (Functor Decomp. Strategy for Verification of Req. and Its Application):**
In this part, we provide a functor decomposition strategy (which is based on the decomposition of a given functor into a category equivalence and an inclusion functor) for the verification of sufficient properties required by the theoretical results from Part ii. This strategy is introduced for \mathcal{M} -functors (see Section 11.1) as well as for restricted \mathcal{M} -functors (see Section 11.2) and, subsequently, its application is discussed for the concrete functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} in Sections 12.1 and 12.2, respectively.
- **Part vi (Related Work, Conclusion, and Future Work):**
The discussion of research related to our approach is given in Chapter 13. Afterwards, in Chapter 14, we conclude this thesis and propose in Chapter 15 possible future research directions.
- **Part vii (Appendices):**
In the final part of this thesis, we include four appendices containing the following additional material. Firstly, in Appendix A, we provide additional categorical foundations consisting of definitions and proofs for technical characteristics used throughout the thesis. Secondly, in Appendix B, we include the proofs for the lemmas required for the main theorems that constitute our general approach presented in Part ii. Thirdly, in Appendix C, we present the proofs, on the one hand, for technical characteristics concerning hypergraph transformation systems and, on the other hand, for the instantiation of our theoretical results presented in Part ii to hypergraph transformation systems according to Part iii. Finally, in Appendix D, we proceed as for the \mathcal{M} -functor \mathcal{F}_{HG} by considering the proofs for PTI net specific technical details and the proofs ensuring a sound instantiation of our theoretical results introduced in Part ii to PTI net transformation systems as presented in Part iv.

1.5 PUBLICATIONS

Several contributions presented in this thesis have been accepted at three peer-reviewed workshops and the *Science of Computer Programming* journal. In all of these papers, I am the first author. In the following, we give a short overview of the submissions and also provide references to technical reports produced for these submissions.

- In our first PNGT paper [207, 208], we introduced our research problem along with initial results from my diploma thesis [206]. We established the notion of (restricted) \mathcal{M} -functors between \mathcal{M} -adhesive categories. We provided sufficient properties guaranteeing that such a (restricted) \mathcal{M} -functor, when it is used to translate an \mathcal{M} -adhesive transformation systems into another, translates and creates (direct) transformations, rule applicability as well as parallel and sequential independence of transformations. We applied these general results by defining a concrete re-

stricted \mathcal{M} -functor \mathcal{F}_{PTI} , which translates PTI nets into typed attributed graphs, and verified the outlined sufficient properties.

- In our second PNGT paper [209, 210], we extended our previous work by considering further semantical properties. In particular, we considered (local) confluence, termination, and functional behavior and verified that our notion of (restricted) \mathcal{M} -functors is able to translate and create these properties in the desired way. To achieve these foundational results, we adapted the list of sufficient properties introduced in the previous paper. As before, we applied our theoretical results by verifying that our restricted \mathcal{M} -functor \mathcal{F}_{PTI} satisfies the list of adapted sufficient properties.
- In our GTVMT paper [211, 212], we extended our work to the case of hypergraph transformation systems. For this purpose, we provided the definition of the \mathcal{M} -functor \mathcal{F}_{HG} (which is not restricted to the translation of \mathcal{M} -morphisms only as it was for the case of the restricted \mathcal{M} -functor \mathcal{F}_{PTI}) translating hypergraphs into typed attributed graphs. Moreover, we verified that the \mathcal{M} -functor \mathcal{F}_{HG} satisfies the list of adapted sufficient properties enabling the application of theoretical results on the transfer of local confluence to concrete hypergraph transformation systems.
- In our journal contribution [213], we extended our previous results on the analysis of local confluence to the case of \mathcal{M} -adhesive transformation systems containing rules with nested application conditions. To apply these new results and to enable analysis using tools such as AGG, we verified that the again extended list of sufficient properties is satisfied by the previously defined \mathcal{M} -functor \mathcal{F}_{HG} .

Major contributions that have not been published earlier are the consideration of bisimulation-based behavior analysis of \mathcal{M} -adhesive transformation systems in Sections 3.2, 6.3, and 9.3, the generalization of the termination, confluence, and functional behavior analysis to \mathcal{M} -adhesive transformation systems with nested application conditions in Section 4.3, the analysis of termination, confluence, and functional behavior of hypergraph transformation systems without or with nested application conditions in Section 7.4, the verification that the restricted \mathcal{M} -functor \mathcal{F}_{PTI} satisfies the corresponding list of sufficient properties enabling the local confluence analysis of PTI net transformation systems with nested application conditions in Section 10.2, the analysis of termination, confluence, and functional behavior of PTI net transformation systems with nested application conditions in Section 10.4, and the functor decomposition strategy for requirement verification together with its application to the concrete functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} in Chapter 11.

In Section 2.1, we recall the approach of algebraic graph transformation with its fundamental ideas and concepts. Afterwards, in Section 2.2, we review the framework of \mathcal{M} -adhesive transformation systems as a generalization of the graph transformation approach and introduce several semantical properties relevant in this framework. Subsequently, in Section 2.3, we consider an extension of \mathcal{M} -adhesive transformation systems allowing to contain rules with nested application conditions and recall the adapted local confluence analysis technique for such extended \mathcal{M} -adhesive transformation systems. Finally, in Section 2.4, we formally introduce three concrete instantiations of \mathcal{M} -adhesive transformation systems, namely typed attributed graph transformation systems, hypergraph transformation systems, and PTI net transformation systems.

2.1 ALGEBRAIC GRAPH TRANSFORMATION APPROACH

Various kinds of graphs can be used to adequately represent the states of many different systems and also the abstract syntax of visual models [122]. Using graphs is often advantageous since they can visualize complex situations on an intuitive level. To describe the local evolution of a system, it is beneficial to be able to define suitable transformation rules specifying the admissible local state modifications. Graph transformation systems are given by such a set of transformation rules that are making use of a common type graph. When a graph transformation system is additionally equipped with a start graph, from which the system is initiated, we obtain a so-called graph grammar, which is quite similar to a Chomsky grammar for formal languages [271, 72, 73, 85]. Graph grammars and the computation of steps by graph transformation are intensively studied in [271, 72, 73]. Different applications of graph transformation and alternative graph based transformation approaches are discussed in detail in Chapter 13.

In this thesis, we are concerned with the algebraic approach [79, 76], which is based on pushout constructions on the involved graphs modeling their gluing. The graphs here can be seen as a special case of an algebra consisting of two basic sets of vertexes and edges together with two operations *source* and *target* defined on these basic sets. Graph morphisms can be considered in the context of the algebraic approach as special algebra homomorphisms and a pushout construction on graphs corresponds to an algebraic quotient algebra construction. Moreover, the usage of pushouts in the algebraic approach allows for the definition of basic theoretical concepts and constructions as well as for the handling of many semantical properties of interest in the context of category theory. In this thesis, we are following the well-established double pushout (DPO) approach explained in detail e.g. in [88] rather than the single pushout (SPO) approach introduced in [265, 200, 199]. For the comparison of both approaches consult e.g. [83] in the Volume I of the Handbook of Graph Grammars [271].

As already pointed out before, the notion of graph transformation rules is one of the central notions in the context of the graph transformation approach. A graph transfor-

mation rule (also called a production) ρ is a span of injective morphisms l and r of the form $\rho = L \xleftarrow{l} K \xrightarrow{r} R$, where L is called a left-hand side (LHS) of the rule, R is called a right-hand side (RHS) of the rule and K is the intersection of the left-hand and the right-hand sides of the rule. For the general case of graphs, L and R represent the graph patterns that should occur in the considered graph before resp. after the rule application. Technically, if we apply a rule to some graph G , first the graph $L \setminus l(K)$ has to be deleted from G and afterwards a copy of the graph $R \setminus r(K)$ has to be added to the result graph, while the graph K (also called a gluing graph) has to be preserved when applying a rule. This described transformation procedure defining a graph transformation step is formally introduced by the DPO approach, where a rule application is defined in terms of category theory by a diagram consisting of two pushouts with total morphisms. Graph transformation sequences are then computed by the iterative application of rules to the graphs obtained from the previous transformation steps.

A graph transformation rule is applicable w.r.t. a given match morphism if the so-called gluing condition is satisfied. This condition intuitively expresses that all dangling edges¹ must be removed and all non-injectively matched nodes and edges must be removed or preserved by a rule application. To increase the descriptive expressiveness of transformation rules, application conditions [77] are often beneficial. They are frequently used for the restriction of rule applicability in graph transformation systems in a wide variety of application areas.

Graphs, which are given by a set of edges, a set of vertexes, and the *source* and *target* functions for the edges as already mentioned before, are a basic and sufficient formalism for many applications. However, additional graph language constructs are desirable leading to high-level structures such as graphs with labels and/or attributes as well as certain structures such as Petri nets, hypergraphs, UML class diagrams [236] etc. Transformation of such high-level structures was introduced by Ehrig et al. in [80, 81] as the concept of High-Level Replacement (HLR) systems in order to establish a common framework for various kinds of graph and Petri net transformation systems. A general instance of the HLR framework is the formalism of typed attributed graph transformation systems introduced in [85] and described in-depth in [88], where attributes and different structural conditions can be enforced on all occurring graphs.

There are many different tools supporting graph transformation. In the 1990s, typed attributed graph transformation was successfully implemented using the Attributed Graph Grammar language in the correspondent tool AGG (Attributed Graph Grammar System) [292, 104, 287]. The development of the AGG-tool has started at the Technische Universität Berlin and continues to this day. AGG supports the specification of algebraic graph transformation systems based on typed attributed graphs with node type inheritance as well as on simple graph rules with nested application conditions and graph constraints [272]. To the best of our knowledge, AGG is the only tool consequently implementing the existing theoretical results for algebraic graph transformation and supporting different analysis techniques for graph grammars containing rules with NACs and PACs, like, e.g., critical pair, dependency and termination analysis, graph parsing as well as consistency and rule applicability checks on graphs [272]. According to [292, 141], AGG provides several graphical editors to create and manipulate graph transformation systems, an interpreter for system simulation as well as a debugger for system verification. The critical pair resp. dependency analysis is offered through a graphical user interface, allowing to browse through the computed critical pairs resp. dependencies. Furthermore, AGG

¹ A dangling edge is an edge whose source or target vertex has been removed.

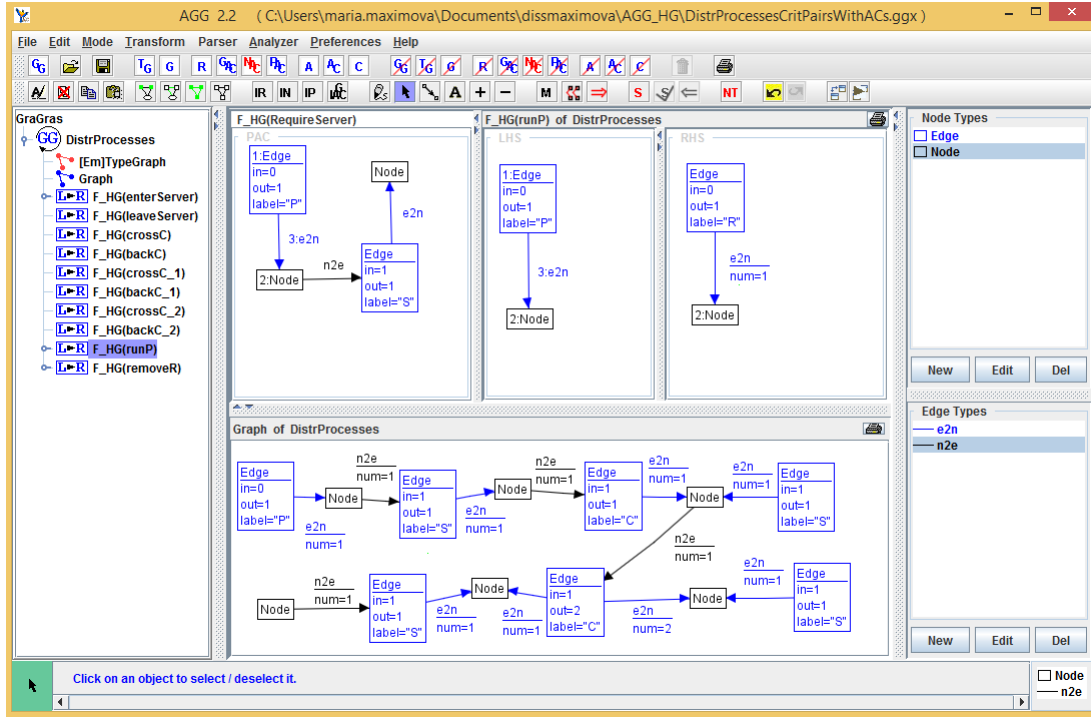


Figure 2: Example view of AGG

supports termination analysis for typed attributed graph transformation systems with injective rules, injective match morphisms and injective NACs using layers for rules and graph types implemented by checking the so-called layering conditions, which are sufficient criteria for termination. Finally, AGG provides *Java* code for the execution of graph transformations that can be used by any graphical or non-graphical *Java* application. Figure 2 above demonstrates an example view of the AGG graphical editor, where the column to the left represents a tree view of the considered graph transformation system including a type graph, a start graph and associated graph transformation rules. The upper part of the picture shows a rule with a PAC and the lower view depicts the start graph of the considered transformation system. Finally, the column to the right shows the allowed node and edge types of the current transformation system. Since AGG is an appropriate tool for local confluence analysis of typed attributed graph transformation systems, we will use it in the following for the verification of our concrete application examples.

Other examples for tools implementing different graph transformation approaches for modeling and analysis reasons are discussed in detail in Chapter 13.

2.2 \mathcal{M} -ADHESIVE TRANSFORMATION SYSTEMS

In this section, we review basic concepts of \mathcal{M} -adhesive transformation systems that are important for our following considerations and give an intuitive as well as a formal introduction to relevant semantical properties of \mathcal{M} -adhesive transformation systems commenting their analysis possibilities.

Since we use the framework of \mathcal{M} -adhesive transformation systems as a fundamental basis for our approach, we assume the reader to be familiar with the concepts used in [88] such as the basics of category theory and double pushout approach, graph transformation systems and High-Level Replacement systems with their semantical properties etc., to be able to follow the formal parts introduced in this thesis.

2.2.1 Basic Concepts of \mathcal{M} -Adhesive Transformation Systems

\mathcal{M} -adhesive transformation systems are one of the categorical frameworks used for graph transformation and High-Level Replacement (HLR) systems in the context of the double pushout (DPO) approach. In this subsection, we recall the basic definitions from the framework of \mathcal{M} -adhesive transformation systems (see [94, 88]), which are based on \mathcal{M} -adhesive categories. The framework of \mathcal{M} -adhesive transformation systems can be instantiated for many practically relevant HLR systems such as various kinds of graphs and Petri nets.

The concept of \mathcal{M} -adhesive categories is based on certain HLR axioms, which are sufficient to derive the main results of graph transformation and HLR systems theory such as Local Church-Rosser, Concurrency, Parallelism, Embedding, Extension, Completeness of Critical Pairs, and Local Confluence Theorems [88, 259]. These HLR axioms rely on the existence of pushouts and pullbacks, where pushouts correspond in general to the union of structures and pullbacks to their intersection, and state their compatibility [88]. The concept of \mathcal{M} -adhesive categories generalizes that of weak adhesive HLR [88], adhesive HLR [89], and adhesive categories [185]. In fact, the mentioned classes of categories are constructed similarly using HLR axioms, which are getting strictly more restrictive from left to right, thus adhesive categories are the most restrictive class and \mathcal{M} -adhesive categories are the least restrictive class of categories. For example, according to [88], the category of sets is contained in each of the four classes, the categories of typed attributed graphs and hypergraphs are adhesive HLR and hence also weak adhesive HLR and \mathcal{M} -adhesive, whereas several categories of Petri nets are not adhesive HLR, but weak adhesive HLR and hence also \mathcal{M} -adhesive.

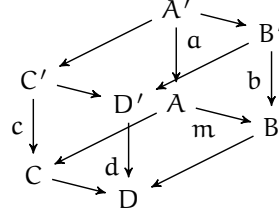
We begin this subsection with the definition of \mathcal{M} -adhesive categories. An *\mathcal{M} -adhesive category* is a short notation for a vertical weak adhesive HLR category introduced in [94]. More precisely, an \mathcal{M} -adhesive category consists of a category \mathbf{C} together with a class \mathcal{M} of monomorphisms² as given in Definition 1 below. Choosing \mathcal{M} to be a suitable subclass of all monomorphisms together with the assumption that pushouts along \mathcal{M} -morphisms are vertical weak van Kampen (VK) squares (subsequently called *\mathcal{M} -VK-squares*) allows to capture, for example, typed attributed graphs and different kinds of Petri nets as instantiations of \mathcal{M} -adhesive categories.

² Note that this does not mean that the category \mathbf{C} is restricted to morphisms from \mathcal{M} as well as that a class \mathcal{M} does not necessarily contain all monomorphisms.

Definition 1 (\mathcal{M} -Adhesive Category [94]).

An \mathcal{M} -adhesive category $(\mathbf{C}, \mathcal{M})$ is a category \mathbf{C} together with a class \mathcal{M} of monomorphisms satisfying the following properties:

- \mathbf{C} has pushouts and pullbacks along \mathcal{M} -morphisms³,
- a class \mathcal{M} is closed under isomorphisms⁴, composition⁵, decomposition⁶, pushouts and pullbacks⁷, and
- pushouts along \mathcal{M} -morphisms are \mathcal{M} -VK-squares (see the diagram below), i.e., the VK-property holds for all commutative cubes, where the given pushout with $m \in \mathcal{M}$ is in the bottom, the back faces are pullbacks and all vertical morphisms a, b, c and d are in \mathcal{M} . The VK-property states that the top face is a pushout iff the front faces are pullbacks.



A set of rules over an \mathcal{M} -adhesive category according to the DPO approach constitutes an \mathcal{M} -adhesive transformation system [94]. In the following, we will use the notions *rule* and *production* as synonym.

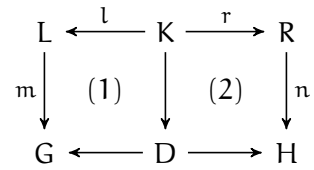
Definition 2 (\mathcal{M} -Adhesive Transformation System [94]).

Consider an \mathcal{M} -adhesive category $(\mathbf{C}, \mathcal{M})$. An \mathcal{M} -adhesive transformation system $AS = ((\mathbf{C}, \mathcal{M}), P)$ ⁸ has in addition a set P of rules of the form $\rho = (L \xleftarrow{l} K \xrightarrow{r} R)$ with $l, r \in \mathcal{M}$.

Applying a rule ρ to some given object G via a match morphism m , we execute a DPO transformation step $G \xrightarrow{\rho, m} H$, also called *direct transformation*, leading to some target object H . The left pushout of the DPO diagram describes the deletion of some elements of G according to the considered rule ρ , while the right pushout describes the addition of some elements to the structure resulting from the deletion. The formal definition of a direct transformation is as follows.

Definition 3 (Direct Transformation [88]).

Consider an \mathcal{M} -adhesive transformation system $(\mathbf{C}, \mathcal{M}, P)$. A direct transformation $G \xrightarrow{\rho, m} H$ via a rule $\rho \in P$ and a match morphism m consists of two pushouts (1) and (2) as shown in the diagram to the right, where $n : R \rightarrow H$ is called a comatch morphism of m . A rule ρ is applicable via m to G , if we have a pushout complement D in (1) such that (1) becomes a pushout. In this case, the target object H of the transformation can be constructed as the gluing of objects D and R via the interface object K .



³ Pushout (pullback) along \mathcal{M} -morphisms means that at least one of the span (cospan) morphisms of the pushout (pullback) diagram is in \mathcal{M} .

⁴ \mathcal{M} is closed under isomorphisms means that for $f \in \mathcal{M}$ and an isomorphism i it holds that also $i \circ f$ and $f \circ i$ are in \mathcal{M} .

⁵ \mathcal{M} is closed under composition means that for $f, g \in \mathcal{M}$ it holds that also $g \circ f \in \mathcal{M}$.

⁶ \mathcal{M} is closed under decomposition means that for $g \circ f \in \mathcal{M}$ and $g \in \mathcal{M}$ it holds that also $f \in \mathcal{M}$.

⁷ \mathcal{M} is closed under pushouts (pullbacks) means that for a pushout (pullback) with one of the span (cospan) morphisms in \mathcal{M} , also the opposite morphism in the pushout (pullback) diagram is in \mathcal{M} .

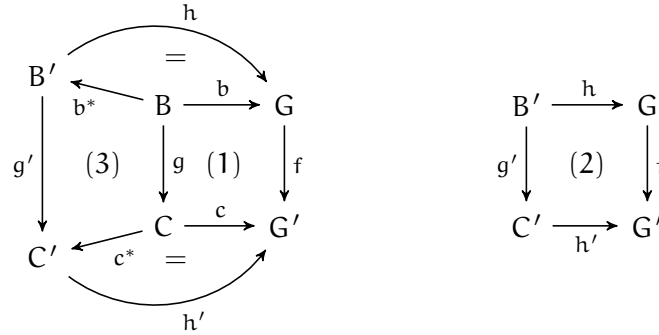
⁸ In the following, we flatten nested tuples to simplify the notation of \mathcal{M} -adhesive transformation systems, i.e., the tuple $((\mathbf{C}, \mathcal{M}), P)$ is written $(\mathbf{C}, \mathcal{M}, P)$.

To check whether a rule ρ is applicable to some object G via a match morphism m , we recall in context of \mathcal{M} -adhesive transformation systems the notion of initial pushouts [88], which represents a universal characterization of the boundary and context constructions. The aim of the initial pushout construction is to define for an arbitrary morphism $f : G \rightarrow G'$ a boundary object B with the morphism $b : B \rightarrow G$ and the context object C with the morphism $c : C \rightarrow G'$ such that the resulting diagram becomes a pushout (see the diagram (1) in the definition below). Intuitively, the object G' in (1) is given as the gluing of objects G and C along the boundary object B .

We need the concept of initial pushouts in the following chapters for the categorical representation of the functorial properties and their proofs. Furthermore, the usage of such general categorical formulations makes the proofs rather concise.

Definition 4 (Initial Pushout [88]).

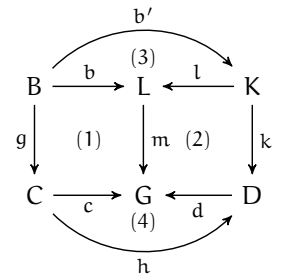
Consider a morphism $f : G \rightarrow G'$ in an \mathcal{M} -adhesive category $(\mathbf{C}, \mathcal{M})^9$. Then (1) is an initial pushout (short IPO) over f with boundary object B , context object C , and $b, c \in \mathcal{M}$ if (1) is a pushout and for every pushout (2) over f with $h, h' \in \mathcal{M}$ it holds that there are unique morphisms $b^* : B \rightarrow B'$ and $c^* : C \rightarrow C'$ in \mathcal{M} such that $h \circ b^* = b$, $h' \circ c^* = c$ and (3) is a pushout.



The notion of initial pushouts allows for the formulation of the gluing condition for rule applicability equivalently to its set-based formulation for graphs (see Definition 74 in Appendix A), but on the abstract level of \mathcal{M} -adhesive transformation systems.

Definition 5 (Gluing Condition in \mathcal{M} -Adhesive Transformation Systems [88]).

Consider an \mathcal{M} -adhesive category $(\mathbf{C}, \mathcal{M})$. For each match $m : L \rightarrow G$ with an initial pushout (1) and $b \in \mathcal{M}$, a rule $\rho = (L \xleftarrow{l} K \xrightarrow{r} R)$ is applicable to an object G with a match morphism $m : L \rightarrow G$ iff the following gluing condition is satisfied: There is a morphism $b' : B \rightarrow K$ in \mathcal{M} with $l \circ b' = b$ (i.e., (3) commutes). Then the pushout complement D in (2) can be constructed as a pushout of morphisms $b' \in \mathcal{M}$ and $g : B \rightarrow C$ leading to morphisms $h : C \rightarrow D, k : K \rightarrow D$ and an induced morphism $d : D \rightarrow G$ such that (2) is a pushout and (4) commutes.



⁹ In general, initial pushouts can also be used in the context of non- \mathcal{M} -adhesive categories with distinguished classes of monomorphisms.

2.2.2 Relevant Semantical Properties in the Framework of \mathcal{M} -Adhesive Transformation Systems

In this subsection, we shortly introduce several semantical properties that are relevant in the context of \mathcal{M} -adhesive transformation systems. In the following chapters, we intend to develop new analysis methods for these semantical properties based on the functorial property transfer between transformation systems with a similar structure.

2.2.2.1 Behavioral Equivalence

The abstract notion of behavioral equivalence is of great importance in the field of computer science since many of its interpretations for concrete formalisms are designed to relate systems of equivalent behavior, i.e., for a definition of behavior and a definition of equivalence (both domain-specific to the concrete formalism) two systems can be compared. An extension of behavioral equivalence is the behavioral congruence where the equivalence must be closed under the application of formalism-specific contexts. The notions of behavioral equivalence and behavioral congruence are, for example, heavily used in the context of process calculi such as CCS [217] and the π -calculus [219]. Based on the notion of borrowed contexts, a bisimulation has been defined for graph transformation systems in [78] and a congruence result has been verified. Behavioral equivalences resp. behavioral congruences allow for the replacement of one system by another in arbitrary contexts. Furthermore, for many interpretations more basic equivalence properties can be derived from behavioral equivalence such as e.g. language equivalence. For the setting of graph transformation systems, our notion of behavioral equivalence preserves important semantical properties as for example termination, confluence, and functional behavior [209, 211]. In fact, the functorial behavior transfer introduced in Section 3.1 constructs a categorical equivalence (i.e., some form of an isomorphism on elements of the considered categories) introduced in Chapter 11 that is sufficient for our interpretation of behavioral equivalence.

The analysis of behavioral equivalence is undecidable for turing-complete formalisms such as graph transformation systems. However, assuming finite reachability graphs, domain-specific, not necessarily terminating, tools may be developed quite easily. Finally, also without tool support, in particular for arbitrary large or even infinite reachability graphs, the notion of bisimilarity [217] is often used (as in this thesis) to ensure or even to define the desired form of behavioral equivalence (even if trace-based behavior equivalence is desired, bisimulations may be used). Moreover, the domain-specific interpretations of bisimulations define a natural proof technique for establishing the desired behavioral equivalence. For analysis reasons, we introduce a definition of bisimulation for graph transformation systems in Subsection 3.2.1 and ensure in Subsections 3.2.1 and 3.2.2 that our functorial behavior transfer is compatible with bisimilarity (in the sense that objects of a source transformation system are bisimilar to their \mathcal{F} -translated counterparts) and preserves bisimulations, respectively.

2.2.2.2 Termination

The semantical property of termination is important in many contexts in computer science. Different from other semantical properties considered in this thesis, there is no context independent answer to whether this property is to be enforced.

Many long running systems, such as embedded systems and arbitrary server processes, are often assumed to be non-terminating, that is, the system should be operational with-

out the need for any predefined time bound or termination condition. While even such long running systems may be terminating, this is usually neither a decision of the system nor the abortion due to a failure: the abortion is, assuming non-faultiness, initiated by some superuser.

On the other hand, many classical software procedures are designed to compute a (possibly) unique result. For these functional procedures the property of termination is crucial as the desired final results are only obtained once the procedure terminates.

Both kinds of systems are, of course, not limited to software applications. Many transformation systems have been used for the modeling of various processes including business processes that also either require termination (finishing of a basic task) or require non-termination (provision of a continuous service).

Irrespective of the kind of a system, if the expectation about termination is not satisfied by the software or process at hand, the system needs to be adapted appropriately.

In the context of HLR systems both kinds of mentioned systems may be modeled [144, 26] and therefore in applications of our theory termination is a semantical property that may be enforced or prevented.

The possible non-determinism of rule applications and match morphism choices implies often the non-termination of the whole transformation system [88]. In this case, the termination can only be guaranteed by carefully designing the transformation rules of the system. The subsequently introduced semantical property of functional behavior, which is meant to ensure that unique results are always computed, also depends on the property of termination [88].

Intuitively, a transformation $G \xRightarrow{*} H$ is called terminating if no rule $\rho \in P$ in the \mathcal{M} -adhesive transformation system $(\mathbf{C}, \mathcal{M}, P)$ is applicable to H anymore [88]. The formal definition of termination is as follows.

Definition 6 (Termination of \mathcal{M} -Adhesive Transformation Systems [88]).

An \mathcal{M} -adhesive transformation system $(\mathbf{C}, \mathcal{M}, P)$ is called terminating if there is no infinite sequence $G_0 \xRightarrow{\rho_1, m_1} G_1 \xRightarrow{\rho_2, m_2} G_2 \xRightarrow{\rho_3, m_3} \dots$ with $\rho_1, \rho_2, \rho_3, \dots \in P$ and matches $m_1, m_2, m_3, \dots \in \text{Mor}_{\mathbf{C}}$.

Many domain specific approaches for determining whether termination is satisfied have been invented. For example, for finite state systems and systems that can be over-/under-approximated by finite state systems the approach of model checking may be sufficient to ensure termination/non-termination [10, 162]. In the domain of graph and model transformation there are several well-established termination analysis techniques like e.g. a classical termination approach constructing a monotonic function that measures graph properties and showing that a value of such a function decreases with every rule application [57], a formulation of concrete criteria for termination like a number of nodes and edges of certain types [8], a general termination approach based on measurement functions [26], sufficient criteria guaranteeing the termination of transformation systems specified by so-called layered graph grammars [25, 86, 194], a combination of termination criteria by isolating deleting graph transformation rules [21, 22], a formulation of a condition stating that a matrix obtained from a graph using a weighted type graph decreases [30], a compositional approach where critical pair analysis is used to determine that the composition of two terminating transformation systems is terminating [257], and a Petri net based analysis method providing a sufficient condition for the termination problem [304].

However, many formalisms are too powerful to be handled this way. That is, the problem of termination is undecidable for graph rewriting systems [256] and any equally powerful formalisms including \mathcal{M} -adhesive transformation systems. The domain specific approaches are often incomplete heuristic algorithms (for example, model checking of infinite state systems may still detect terminating states) or the domains are sufficiently weak w.r.t. their expressive power (for example, termination of primitive recursive functions is immediate).

2.2.2.3 Confluence

Confluence is one of the most important semantical properties of rewriting systems stating that the system has functional behavior in the sense that two distinct transformation sequences starting in the same object can be merged to a common result [88]. In a special case, the confluence property is satisfied for a transformation system if each pair of its rules is parallel independent (i.e., conflict-free) for all possible match morphisms [88]. While some relevant transformation systems are not confluent, they may contain fragments that are confluent. In these cases, the confluence of such fragments can be used to verify other relevant semantical properties of the whole system.

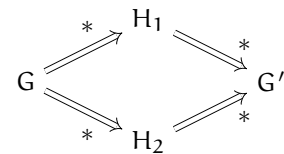
From the applicational point of view, the confluence property is important to use successfully graph transformation techniques, for instance, for the automated translation of UML models into semantic domains [144], for parsing of graph languages [108, 269], for efficiently recognizing graph classes and executing graph algorithms by graph reduction [7, 23], for analyzing the deterministic behavior of programs in graph rewriting languages [104, 283] and for solving the kernelization problem in the domain of the parameterized algorithmics [97].

The notion of confluence is related to the notion of global determinism, meaning that for each pair of terminating transformations with the same source object, the target objects are equal or isomorphic [88]. According to [88], every confluent graph transformation system is globally deterministic, while the single rule applications may be locally non-deterministic. There are two possible reasons for local non-determinism. Firstly, local non-determinism is given when at some point in a transformation sequence several transformation rules are applicable to the target object of the previous transformation step. In this case, one of the rules has to be chosen non-deterministically for application. Secondly, local non-determinism occurs if a concrete rule can be applied to the considered object via several different valid match morphisms. In that case one of the valid match morphisms has to be chosen non-deterministically.

A transformation system is confluent if, starting with some object G , whenever we can transform G into the objects H_1 and H_2 , we can subsequently transform these two objects into the same object G' as given in the diagram below.

Definition 7 (Confluence of \mathcal{M} -Adhesive Transformation Systems [88]).

An \mathcal{M} -adhesive transformation system $(\mathbf{C}, \mathcal{M}, P)$ is confluent if for all transformations $G \xRightarrow{*} H_1$ and $G \xRightarrow{*} H_2$ there is an object G' together with transformations $H_1 \xRightarrow{*} G'$ and $H_2 \xRightarrow{*} G'$.



2.2.2.4 Local Confluence

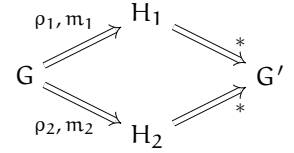
In [234] Newman states that it is sufficient to verify a weaker form of confluence, called *local confluence* (in which only distinct transformation steps have to be merged to a common result), to obtain confluence for terminating transformation systems. This local confluence property stems originally from the domain of term rewriting and was considered in detail for hypergraph rewriting systems in [254] as well as for typed attributed graph transformation systems in [144].

According to [88, 122], local confluence analysis can be furthermore used to verify that in a system all conflicts of rule applications can be resolved. These conflicts can be computed at design time by analyzing the rules of the considered transformation system. A conflict arises, for instance, if one rule deletes an element that is used by another rule. In this case, after the first rule deletes the element used by the second rule, the second rule is not applicable anymore. Moreover, the local confluence analysis identifies independent rule applications as well as determines the non-resolvable conflicting rule applications.

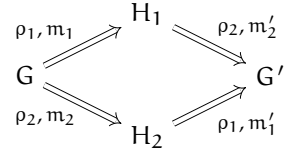
According to [88], the local confluence property for an \mathcal{M} -adhesive transformation system is given as follows.

Definition 8 (Local Confluence of \mathcal{M} -Adhesive Transformation Systems [88]).

An \mathcal{M} -adhesive transformation system $(\mathbf{C}, \mathcal{M}, \mathbf{P})$ is locally confluent if for all pairs of direct transformations $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$ there is an object G' together with transformations $H_1 \xrightarrow{*} G'$ and $H_2 \xrightarrow{*} G'$.



There are two different ways for the verification of local confluence of two direct transformations. We have a special case if two direct transformations $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$ are parallel independent. In this case, we use the Local Church-Rosser Theorem [88], which is an important result in the context of the HLR systems theory, for the verification of local confluence (see the diagram to the right). Thus, the parallel in-



dependence of all two direct transformations is a sufficient condition for local confluence. The second case is a more general case, which is given if two direct transformations $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$ are not necessarily parallel independent. The verification techniques for this case, based on the analysis of critical pairs, will be considered in detail in the following.

The property of independence of transformation steps is one of the semantical properties, whose preservation and reflection will be considered in our results. This property represents a technical characterization used in the Local Church-Rosser Theorem. Intuitively, the notion of independence for two transformation steps means that these steps are neither in conflict nor causally dependent. We say that two transformation steps are in conflict if they are dependent in the sense that one of the rules will disable the other one by its application. Moreover, it is also possible that one of the transformation steps can only be executed after the other has been executed before. In this situation, we have that one step is causal dependent of another one. According to these two possibilities, we distinguish in the following two kinds of independence, namely the parallel

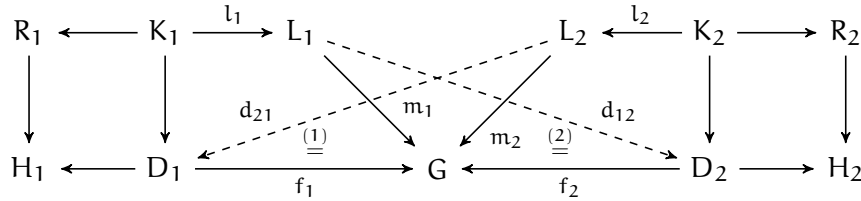
independence (absence of conflicts) and sequential independence (absence of causal dependencies) [141, 88].

According to [88], the formal definition of the parallel and sequential independence is as follows. Technically, this definition means that each of the considered rules does not delete any element, which is a part of the match morphism of the other rule.

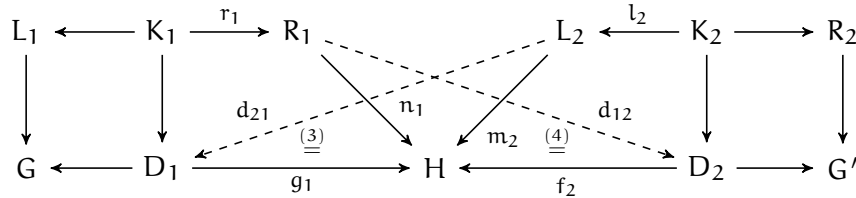
Definition 9 (Parallel and Sequential Independence of Direct Transformations [88]).

Consider an \mathcal{M} -adhesive transformation system $AS = (\mathbf{C}, \mathcal{M}, \mathbf{P})$.

- Two direct transformations $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$ are called parallel independent if there exist morphisms $d_{12} : L_1 \rightarrow D_2$, $d_{21} : L_2 \rightarrow D_1$ ¹⁰ such that $f_1 \circ d_{21} = m_2$ and $f_2 \circ d_{12} = m_1$ (i.e., (1) and (2) commute).



- Two direct transformations $G \xrightarrow{\rho_1, m_1} H$ and $G \xrightarrow{\rho_2, m_2} G'$ are called sequentially independent if there exist morphisms $d_{12} : R_1 \rightarrow D_2$, $d_{21} : L_2 \rightarrow D_1$ such that $g_1 \circ d_{21} = m_2$ and $f_2 \circ d_{12} = m_1$ (i.e., (3) and (4) commute).



For an example, showing parallel and sequential independent transformation steps, see e.g. Example 3.17 from [88].

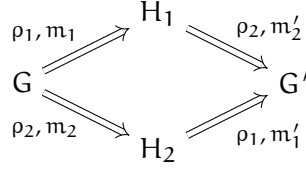
As already mentioned before, the Local Church-Rosser Theorem is based upon the concepts of parallel and sequential independence. Intuitively, this theorem describes under which conditions two rules with given match morphisms can be applied in arbitrary order to the same object such that the corresponding transformation steps lead to the same result. The proof for this theorem is given in [88].

Fact 1 (Local Church-Rosser Theorem [88]).

Given an \mathcal{M} -adhesive transformation system $AS = (\mathbf{C}, \mathcal{M}, \mathbf{P})$.

- Consider two parallel independent direct transformations $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$. Then there are an object G' and direct transformations $H_1 \xrightarrow{\rho_2, m'_2} G'$ and $H_2 \xrightarrow{\rho_1, m'_1} G'$ such that $G \xrightarrow{\rho_1, m_1} H_1 \xrightarrow{\rho_2, m'_2} G'$ and $G \xrightarrow{\rho_2, m_2} H_2 \xrightarrow{\rho_1, m'_1} G'$ are sequentially independent.
- Consider two sequentially independent direct transformations $G \xrightarrow{\rho_1, m_1} H_1 \xrightarrow{\rho_2, m'_2} G'$. Then there are an object H_2 and direct transformations $G \xrightarrow{\rho_2, m_2} H_2 \xrightarrow{\rho_1, m'_1} G'$ such that $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$ are parallel independent.

¹⁰ The potentially existing morphisms d_{12} and d_{21} are depicted by dashed arrows in the corresponding diagrams.

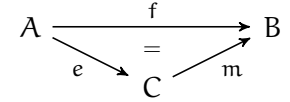


As the next step we recall basic notions and concepts needed for the introduction of the critical pair analysis [88, 292], which is a well-known static analysis technique for the verification of local confluence of parallel dependent transformations. Critical pair analysis stems originally from the area of term rewriting [25] and has been generalized to graph rewriting in [254, 201]. Intuitively, the advantage of using critical pairs for checking local confluence is that we do not have to investigate all possible pairs of rule applications, which are parallel dependent, but only some minimal ones, which are constructed by gluing the left-hand sides of each pair of rules.

We begin with the notions of $\mathcal{E} - \mathcal{M}$ -factorization and $\mathcal{E}' - \mathcal{M}'$ pair factorization, which are essential for the construction of critical pairs and their embedding into direct transformations. We use the notion of $\mathcal{E} - \mathcal{M}$ -factorization, given for example in [88], to be able to decompose a given morphism $f : A \rightarrow B$ into two morphisms $e : A \rightarrow C$ and $m : C \rightarrow B$ with specific properties.

Definition 10 ($\mathcal{E} - \mathcal{M}$ -Factorization [88]).

Consider a category \mathbf{C} and morphisms $f : A \rightarrow B$, $e : A \rightarrow C$, $m : C \rightarrow B$. Then \mathbf{C} has an $\mathcal{E} - \mathcal{M}$ -factorization for given morphism classes \mathcal{E} and \mathcal{M} if for each morphism f there is a decomposition, unique up to isomorphism, $f = m \circ e$ with $e \in \mathcal{E}$ and $m \in \mathcal{M}$. Usually, \mathcal{E} is a subclass of epimorphisms and \mathcal{M} is a subclass of monomorphisms.

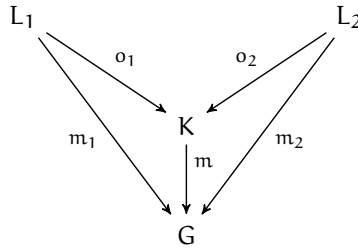


An $\mathcal{E}' - \mathcal{M}'$ pair factorization provides the smallest overlapping of two objects embedded into a given context. In many applications the class \mathcal{E}' is the class of all jointly surjective morphisms and \mathcal{M}' the class of all injective morphisms.

Definition 11 (\mathcal{E}' -Instance, $\mathcal{E}' - \mathcal{M}'$ Pair Factorization [88]).

Consider a category \mathbf{C} with a class of jointly epimorphic morphisms \mathcal{E}' (see Definition 71 in Appendix A) and a class of monomorphisms \mathcal{M}' .

- We call a pair of morphisms $(o_1, o_2) \in \mathcal{E}'$ an \mathcal{E}' -instance.
- A category \mathbf{C} has an $\mathcal{E}' - \mathcal{M}'$ pair factorization if for each pair of morphisms $m_1 : L_1 \rightarrow G$ and $m_2 : L_2 \rightarrow G$ there exist unique up to isomorphism object K and morphisms $o_1 : L_1 \rightarrow K$, $o_2 : L_2 \rightarrow K$, $m : K \rightarrow G$ with $(o_1, o_2) \in \mathcal{E}'$ and $m \in \mathcal{M}'$ such that $m \circ o_i = m_i$ for $i \in \{1, 2\}$. In this case, we say that $((o_1, o_2), m)$ is an $\mathcal{E}' - \mathcal{M}'$ pair factorization of (m_1, m_2) .



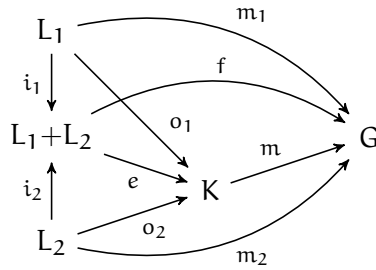
Remark 1 ($\mathcal{E}' - \mathcal{M}$ Pair Factorization).

Note that in our applications the monomorphism class \mathcal{M}'_i contains always all monomorphisms from \mathcal{M}_i for $i \in \{1, 2\}$. For this reason, we use in the following the adapted notion of $\mathcal{E}' - \mathcal{M}'$ pair factorization, which we call $\mathcal{E}' - \mathcal{M}$ pair factorization, when providing new or customized definitions and results. However, we still use the notion of $\mathcal{E}' - \mathcal{M}'$ pair factorization when repeating definitions and results from the literature.

In general, we can construct an $\mathcal{E}' - \mathcal{M}$ pair factorization from an $\mathcal{E} - \mathcal{M}$ -factorization and binary coproducts¹¹. This construction is described in detail in the lemma below.

Lemma 1 ($\mathcal{E}' - \mathcal{M}$ Pair Factorization Based on $\mathcal{E} - \mathcal{M}$ -Factorization).

Consider a category \mathbf{C} with $\mathcal{E} - \mathcal{M}$ -factorizations, as given in Definition 10, and binary coproducts. Let furthermore $(m_1 : L_1 \rightarrow G, m_2 : L_2 \rightarrow G)$ be a morphism pair with the common codomain, $(L_1 + L_2, i_1 : L_1 \rightarrow L_1 + L_2, i_2 : L_2 \rightarrow L_1 + L_2)$ be a binary coproduct of (L_1, L_2) with induced coproduct morphism $f : L_1 + L_2 \rightarrow G$ and $L_1 + L_2 \xrightarrow{e} K \xrightarrow{m} G$ be an $\mathcal{E} - \mathcal{M}$ -factorization of the morphism f as given in Definition 11. Then $((o_1, o_2), m)$ with $o_1 = e \circ i_1$ and $o_2 = e \circ i_2$ is an $\mathcal{E}' - \mathcal{M}$ pair factorization of (m_1, m_2) .

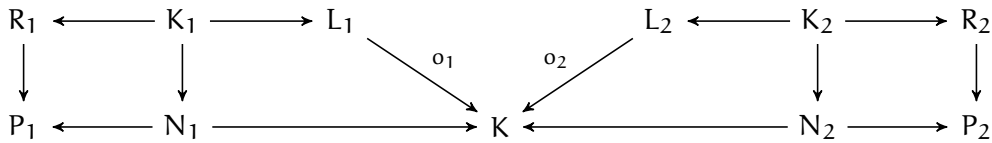
**Proof.**

The detailed proof for this lemma is given in [Appendix A](#) on [page 269](#). □

The critical pair analysis is based upon the notion of critical pairs. Critical pairs represent precisely all potential conflicts of rule applications. A critical pair $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ is a pair of parallel dependent transformations with a minimal overlapping K of the left-hand sides of the rules. The formal definition of a critical pair is given in the following.

Definition 12 (Critical Pair [88]).

Consider an $\mathcal{E}' - \mathcal{M}'$ pair factorization. A pair $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ of direct transformations is called a critical pair if it is parallel dependent and minimal in the sense that the pair (o_1, o_2) of match morphisms $o_1 : L_1 \rightarrow K$ and $o_2 : L_2 \rightarrow K$ is in \mathcal{E}' .



¹¹ Coproducts can be constructed from initial objects and pushouts. For the categories considered subsequently, we assume the existence of initial objects.

Another important well-known result, that we use in our approach, is the *Completeness of Critical Pairs Lemma* [88]. This lemma, stating that each pair of parallel dependent direct transformations embeds a critical pair, justifies why it is sufficient to consider the local confluence property only for certain parallel dependent rule applications, which are minimal in the sense of critical pairs.

Fact 2 (Completeness of Critical Pairs Lemma [88]).

Consider an \mathcal{M} -adhesive transformation system $(\mathbf{C}, \mathcal{M}, \mathbf{P})$ with an $\mathcal{E}' - \mathcal{M}'$ pair factorization, where the $\mathcal{M} - \mathcal{M}'$ pushout-pullback decomposition property¹² holds.¹³ The critical pairs are then complete, i.e., for each pair of parallel dependent direct transformations $H_1 \xrightarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ there is a critical pair $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ with extension diagrams¹⁴ (1), (2) and $m \in \mathcal{M}'$.

$$\begin{array}{ccccc}
 P_1 & \xleftarrow{\rho_1, o_1} & K & \xrightarrow{\rho_2, o_2} & P_2 \\
 \downarrow & & \downarrow m & & \downarrow \\
 H_1 & \xleftarrow{\rho_1, m_1} & G & \xrightarrow{\rho_2, m_2} & H_2
 \end{array}
 \begin{array}{c}
 (1) \qquad (2)
 \end{array}$$

According to [88], the *Completeness of Critical Pairs Lemma* already implies the local confluence for a transformation system containing no critical pairs. Otherwise, we have to show the additional strict confluence property given in Definition 14 for all critical pairs of the considered transformation system. Strictness means intuitively that the maximal substructure of K , that is preserved by the critical pair, has also to be preserved by the merging transformation steps $P_1 \xrightarrow{*} K'$ and $P_2 \xrightarrow{*} K'$ (see the diagram in Definition 14). In order to formulate the strict confluence property, we use the notion of a derived span $\text{der}(t) = (G_0 \leftarrow D \rightarrow G_n)$ for some transformation $t : G_0 \xrightarrow{*} G_n$ connecting the first and the last objects of the transformation [88]. The formal definition of a derived span is as follows.

Definition 13 (Derived Span [88]).

Consider an \mathcal{M} -adhesive transformation system $(\mathbf{C}, \mathcal{M}, \mathbf{P})$.

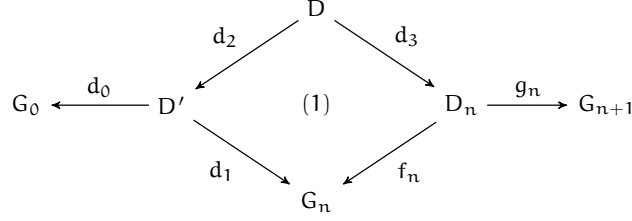
- The derived span of an identical transformation $t : G \xrightarrow{id} G$ is defined by $\text{der}(t) = (G \leftarrow G \rightarrow G)$ with identical morphisms.
- The derived span of a direct transformation $G \xrightarrow{\rho, m} H$ is the span $(G \leftarrow D \rightarrow H)$ of the corresponding double pushout (see the diagram in Definition 3).
- For a transformation $t : G_0 \xrightarrow{*} G_n \Rightarrow G_{n+1}$ the derived span is the composition via the pullback (1) of the derived spans $\text{der}(G_0 \xrightarrow{*} G_n) = (G_0 \xleftarrow{d_0} D' \xrightarrow{d_1} G_n)$ and $\text{der}(G_n \Rightarrow G_{n+1}) = (G_n \xleftarrow{e_0} E' \xrightarrow{e_1} G_{n+1})$.

¹² The $\mathcal{M} - \mathcal{M}'$ pushout-pullback decomposition property is a technical property needed for decomposition of pushouts consisting of a pullback and special morphisms in \mathcal{M} and \mathcal{M}' . For a formal definition of the $\mathcal{M} - \mathcal{M}'$ pushout-pullback decomposition property see Definition 72 in Appendix A.

¹³ In our applications the monomorphism class \mathcal{M}' contains always all monomorphisms from \mathcal{M} , as already mentioned in Remark 1, which leads to the trivial satisfaction of the $\mathcal{M} - \mathcal{M}'$ pushout-pullback decomposition property.

¹⁴ According to [88], an extension diagram describes how transformations $K \xrightarrow{\rho_1, o_1} P_1$ and $K \xrightarrow{\rho_2, o_2} P_2$ can be extended to transformations $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$, respectively, via an extension morphism $m : K \rightarrow G$ mapping K to G . For a formal definition of an extension diagram see Definition 73 in Appendix A.

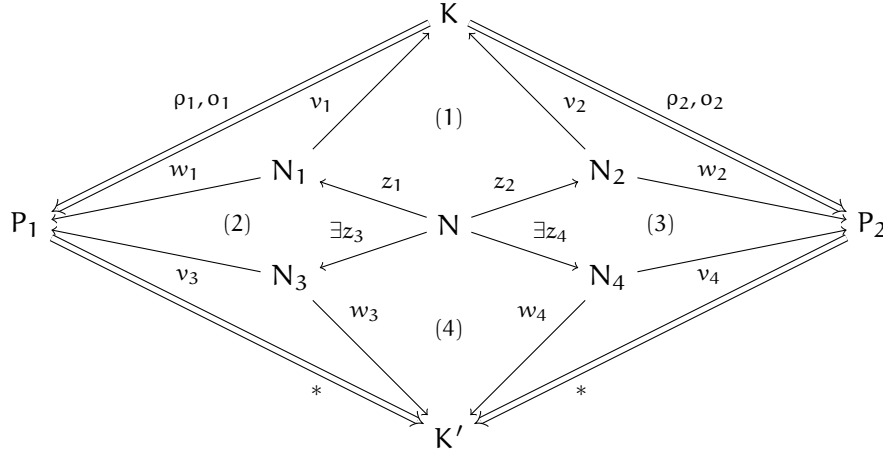
$G_{n+1}) = (G_n \xleftarrow{f_n} D_n \xrightarrow{g_n} G_{n+1})$. This construction leads to the derived span $\text{der}(t) = (G_0 \xleftarrow{d_0 \circ d_2} D \xrightarrow{g_n \circ d_3} G_{n+1})$ shown below.



Definition 14 (Strict Confluence of Critical Pairs [88]).

Consider an \mathcal{M} -adhesive transformation system $(\mathcal{C}, \mathcal{M}, P)$. A critical pair $P_1 \xleftarrow{\rho_1, \sigma_1} K \xrightarrow{\rho_2, \sigma_2} P_2$ is called strictly confluent if we have the following:

- **Confluence:** The critical pair $P_1 \xleftarrow{\rho_1, \sigma_1} K \xrightarrow{\rho_2, \sigma_2} P_2$ of direct transformations is confluent, i.e., there exist an object K' and transformations $P_1 \xrightarrow{*} K'$ and $P_2 \xrightarrow{*} K'$ with derived spans $\text{der}(P_i \xrightarrow{*} K') = (P_i \xleftarrow{v_{i+2}} N_{i+2} \xrightarrow{w_{i+2}} K')$ for $i \in \{1, 2\}$.
- **Strictness:** Let $\text{der}(K \xrightarrow{\rho_i, \sigma_i} P_i) = (K \xleftarrow{v_i} N_i \xrightarrow{w_i} P_i)$ for $i \in \{1, 2\}$ and let N be the pullback object of the pullback (1). Then there exist morphisms z_3 and z_4 such that (2), (3), and (4) commute.



As already mentioned before, in order to show local confluence of an \mathcal{M} -adhesive transformation system, it is sufficient to show the strict confluence of all its critical pairs (see the Local Confluence Theorem and Critical Pair Lemma from [88] below and [255], where Plump has shown that it does not suffice to verify only the *Confluence condition* for all critical pairs without the proof of their *Strictness condition*.)

Fact 3 (Local Confluence Theorem and Critical Pair Lemma [88]).

An \mathcal{M} -adhesive transformation system is locally confluent if all its critical pairs are strictly confluent.

For concrete \mathcal{M} -adhesive transformation systems the graph transformation tool AGG [292] can be used for conflict detection. Subsequently, AGG allows for the semiautomatic proof of the *Confluence condition* for the computed critical pairs. However, the analysis of

the *Strictness condition* is not supported by AGG yet and has to be done manually outside of that tool environment.

2.2.2.5 Functional Behavior

As pointed out in [88], a transformation sequence $G \xRightarrow{*} H$ can be seen in general as a computation leading to some result H if the considered sequence is terminating. But, due to the non-determinism of rule applications and match morphism choices, another terminating transformation sequence starting with the same object G can lead to a different result as the object H . This case cannot occur if the considered transformation system is confluent, i.e., for a confluent transformation system we have that the result of different terminating transformation sequences starting with the same object is unique up to isomorphism. This semantical property is called functional behavior [88] and is very advantageous in the domains of graph and model transformation [144, 269, 7, 104, 283].

Besides single-process systems, for which functional behavior is desirable, it is important to realize that many parallel programs compute functions collaboratively requiring often functional behavior as well. Furthermore, submodules contained in the system at hand may have functional behavior, while the overall system is inherently not observing functional behavior on purpose.

According to [88], a locally confluent and terminating transformation system is confluent and has functional behavior as given in the following remark.

Remark 2 (Functional Behavior of \mathcal{M} -Adhesive Transformation Systems [88]).

Consider a terminating and locally confluent \mathcal{M} -adhesive transformation system $AS = (\mathbf{C}, \mathcal{M}, P)$. Then AS is confluent and has functional behavior in the following sense:

- For each object G there is an object H together with a terminating transformation $G \xRightarrow{*} H$ in AS , and H is unique up to isomorphism.
- Each pair of transformations $G \xRightarrow{*} H_1$ and $G \xRightarrow{*} H_2$ can be extended to terminating transformations $G \xRightarrow{*} H_1 \xRightarrow{*} H$ and $G \xRightarrow{*} H_2 \xRightarrow{*} H$ with the same object H .

2.3 ADVANCED CONCEPTS FOR RULE-BASED TRANSFORMATIONS AND THEIR ANALYSIS

The aim of this section is to give an overview for already existing notions and general results concerning transformations with nested application conditions and local confluence analysis for this kind of transformations. In the following chapters, we build our new theory for verification of local confluence based on so-called \mathcal{M} -functors using these existing notions and results.

2.3.1 Transformations with Nested Application Conditions

Application conditions were first introduced in [77] and can be used in a big variety of application areas to strengthen the descriptive expressiveness of transformation systems by restricting the application of rules based on more complex conditions on the current structure.

Different kinds of application conditions were already studied in detail e.g. in [77, 139, 84, 135, 88, 136]. Application conditions can be nested, may contain Boolean expressions [135, 136] and are expressively equivalent to first-order formulas on graphs [51] as shown in [136, 270]. We generally use the term “nested application condition” whenever we refer to the most general case.

Nested application conditions in this thesis are defined in terms of application conditions for rules in [135, 136]. We use this kind of nested application conditions to restrict the applicability of rules to some given object. For this reason we equip the left hand side or right hand side of a rule with some nested application condition. If the match morphism to some given object G satisfies this nested application condition, we have that the considered rule is applicable to the object G . The formal definition of a nested application condition is given in the following.

Definition 15 (Nested Application Condition [135]).

A nested application condition ac_P over an object P ¹⁵ is inductively defined as follows:

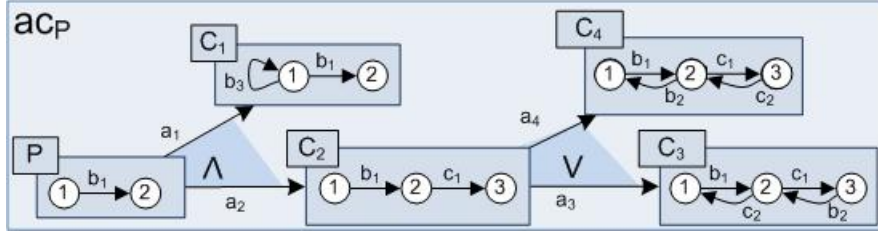
- *true* is an application condition over the object P .
- For every morphism $\alpha : P \rightarrow C$ and every nested application condition ac_C over the object C , $\exists(\alpha, ac_C)$ is a nested application condition over the object P .
- A nested application condition can also be a Boolean formula over nested application conditions. This means that also $\neg ac_P$, $\bigwedge_{i \in J} ac_{P,i}$, and $\bigvee_{i \in J} ac_{P,i}$ are nested application conditions over the object P with nested application condition $ac_{P,i}$ and $i \in J$ for some index set J .

For better understanding of the notion introduced above, we give an example for a nested application condition on graphs also used in [280].

Example 1 (Nested Application Condition [280]).

Consider the nested application condition ac_P from Figure 3, where all morphisms are inclusions. ac_P means that the source of every b -edge has a b -self-loop and must be followed by some c -edge such that subsequently there is a path in the reverse direction visiting the source and the target of the first b -edge with precisely one c -edge and one b -edge in an arbitrary order. We denote this nested application condition by $ac_P = \exists(\alpha_1, true) \wedge \exists(\alpha_2, \exists(\alpha_3, true) \vee \exists(\alpha_4, true))$.

¹⁵ For our considerations, P is usually the left-hand side of a rule.

Figure 3: Nested application condition ac_P

$$\begin{array}{ccc}
 ac_P \triangleright P & \xrightarrow{a} & C \triangleleft ac_C \\
 & \searrow p & \swarrow q \in \mathcal{M} \\
 & G &
 \end{array}$$

Figure 4: Diagram for satisfaction of a nested application condition $ac_P = \exists(a, ac_C)$

A rule with nested application condition is applicable to some given object only if the considered match morphism satisfies the nested application condition. As a next step, we define according to [135], whether a morphism satisfies a nested application condition.

Definition 16 (Satisfaction of Nested Application Conditions [135]).

We define inductively when a morphism $p : P \rightarrow G^{16}$ satisfies a nested application condition ac_P over an object P :

- Every morphism satisfies the nested application condition true.
- A morphism $p : P \rightarrow G$ satisfies a nested application condition $\exists(a, ac_C)$, written $p \models \exists(a, ac_C)$, if there exists an \mathcal{M} -morphism q such that $q \circ a = p$ and $q \models ac_C$ (see Figure 4)¹⁷.
- $p \models \neg ac_P$ means that $p \not\models ac_P$.
- $p \models \bigwedge_{i \in \mathcal{I}} ac_{P,i}$ means that for all $i \in \mathcal{I}$ it holds $p \models ac_{P,i}$ for some index set \mathcal{I} .
- $p \models \bigvee_{i \in \mathcal{I}} ac_{P,i}$ means that for some $i \in \mathcal{I}$ it holds $p \models ac_{P,i}$ for some index set \mathcal{I} .

We demonstrate in the following example the satisfaction of a nested application condition ac_P from Example 1 by some morphism p .

Example 2 (Satisfaction of Nested Application Conditions [280]).

The inclusion $p : P \rightarrow G_A$ from Figure 5 satisfies ac_P from the Example 1. This holds, because there are inclusions $q_i : C_i \rightarrow G_A$ such that $q_i \circ a_i = p$ for $i \in \{1, 2\}$ and $q_1 \models true$, $q_2 \models \exists(a_3, true) \vee \exists(a_4, true)$ since there is the inclusion $q_3 : C_3 \rightarrow G_A$ such that $q_3 \circ a_3 = q_2$ and $q_3 \models true$.

Remark 3 (Negative and Positive Application Conditions [213]).

Nested application conditions as described before represent a generalization of negative and positive application conditions (short NACs and PACs) introduced in [143, 139] and [87], respectively. A NAC (resp. PAC) over some object P is defined in terms of a morphism $a : P \rightarrow C$ (see Figure 4) and is denoted by $\neg \exists a$ (resp. $\exists a$). A NAC $\neg \exists a$ (resp. PAC $\exists a$) is satisfied by a morphism $p : P \rightarrow G$ if there is no \mathcal{M} -morphism $q : C \rightarrow G$ (resp. there is an \mathcal{M} -morphism $q : C \rightarrow G$) making the diagram in Figure 4 commute.

¹⁶ For our considerations, p corresponds usually to a match morphism $m : L \rightarrow G$.

¹⁷ The notation $ac_P \triangleright P$ in Figure 4 indicates that ac_P is a nested application condition over the object P .

Remark 4 (Shift of Nested Application Conditions over Morphisms [95]).

We use the **Shift**-transformation for shifting of nested application conditions over a morphism

b. The **Shift**-transformation is inductively defined as follows:

- For $ac_P = \text{true}$: $\text{Shift}(b, ac_P) = \text{true}$,
- For $ac_P = \exists(a, ac_C)$:
 - $\text{Shift}(b, ac_P) = \bigvee_{(a', b') \in F} \exists(a', \text{Shift}(b', ac_C))$
 if $F = \{(a', b') \in \mathcal{E}' \mid b' \in \mathcal{M} \wedge a' \circ b = b' \circ a\} \neq \emptyset^{18}$,
 - $\text{Shift}(b, ac_P) = \text{false}$ if $F = \emptyset$,
- For $ac_P = \neg ac'_P$: $\text{Shift}(b, ac_P) = \neg(\text{Shift}(b, ac'_P))$,
- For $ac_P = \bigwedge_{i \in J} ac_{P,i}$: $\text{Shift}(b, ac_P) = \bigwedge_{i \in J} (\text{Shift}(b, ac_{P,i}))$,
- For $ac_P = \bigvee_{i \in J} ac_{P,i}$: $\text{Shift}(b, ac_P) = \bigvee_{i \in J} (\text{Shift}(b, ac_{P,i}))$,

where F is the possibly infinite index set, b' is an inclusion, \mathcal{E}' is a class of jointly epimorphic morphisms, and $ac_{P,i}$ is a nested application condition with $i \in J$ for some index set J .

$$\begin{array}{ccc}
 ac_P \triangleright P & \xrightarrow{b} & P' \\
 \downarrow a & = & \downarrow a' \\
 ac_C \triangleright C & \xrightarrow{b'} & C'
 \end{array}
 \qquad
 \begin{array}{ccc}
 ac_P \triangleright P & \xrightarrow{b} & P' \triangleleft \text{Shift}(b, ac_P) \\
 \searrow n \circ b & & \swarrow n \\
 & H &
 \end{array}$$

Moreover, as shown in [95], for each \mathcal{M} -adhesive category $(\mathbf{C}, \mathcal{M})$ with the transformation **Shift**, as given before, and an \mathcal{E}' - \mathcal{M}' pair factorization, where \mathcal{M}' is a monomorphism subclass of \mathcal{M} , it holds that the **Shift**-transformation and the satisfaction relation are compatible, technically formulated as follows:

$$\forall ac_P, \forall b : P \rightarrow P', n : P' \rightarrow H. (n \circ b \models ac_P) \Leftrightarrow (n \models \text{Shift}(b, ac_P))$$

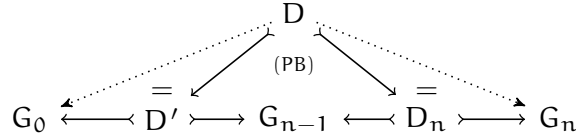
For an example, how the **Shift**-transformation can be applied, see Figure 15 in Example 8 from Subsection 2.4.2.

As a next step, we recall the notions of plain derived rule, derived nested application condition and derived rule of a transformation $t : G_0 \xRightarrow{*} G_n$ from [96], which we need later to define a special kind of compatibility of AC-disregarding transformations, the so-called AC-compatibility, as well as to specify the shifting of nested application conditions over rules. A plain derived rule $\rho(t)$ is a single rule from G_0 to G_n containing all changes that should be done during the transformation t . A derived nested application condition $ac(t)$ combines all nested application conditions of the transformation t into a single nested application condition over the object G_0 . Finally, a derived rule consists of the corresponding plain derived rule and derived nested application condition of t .

Definition 21 (Plain Derived Rule [96]).

For a transformation $t : G_0 \xRightarrow{*} G_n$, the plain derived rule $p(t)$ is defined by the span $G_0 \leftarrow D_0 \rightarrow G_1$ for $n = 1$ and by iterated pullback construction leading to the span $G_0 \leftarrow D \rightarrow G_n$ for $n \geq 2$ as depicted in the diagram below with dotted arrows, where all morphisms are inclusions.

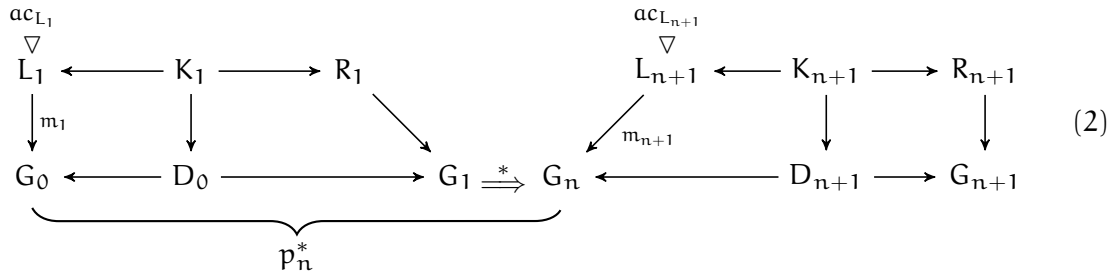
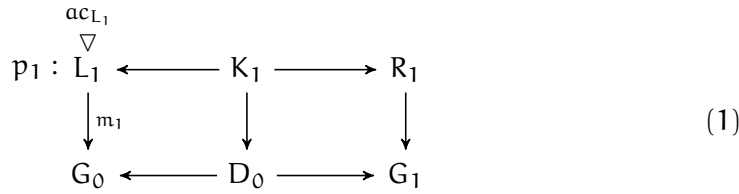
¹⁸ Intuitively, the index set F (containing the pairs of morphisms (a', b') leading to graphs C') can be understood as a case distinction of situations given by overlappings of C and P' (see the diagram to the left in Remark 4), where both nested application conditions expressed by $\exists(a, \text{true})$ and $\exists(b, \text{true})$ are satisfied. Note that the shifting is applied recursively.



Definition 22 (Derived Nested Application Condition [96]).

Consider an AC-disregarding transformation $t : G_0 \xRightarrow{*} G_n$. The derived nested application condition $ac(t)$ over the object G_0 is inductively defined as follows:

- For t of length 0 with $G_0 \cong G'_0$ and G'_0 from the extension diagram given in Definition 73 in Appendix A, let $ac(t) = \text{true}$.
- For $t : G_0 \xRightarrow{\rho_1, m_1} G_1$, let $ac(t) = \text{Shift}(m_1, ac_{L_1})$ (see the diagram (1) below).
- For $t : G_0 \xRightarrow{*} G_n \Rightarrow G_{n+1}$ with $n \geq 1$, let $ac(t) = ac(G_0 \xRightarrow{*} G_n) \wedge L(p_n^*, ac(G_n \Rightarrow G_{n+1}))$ where $p_n^* = (G_0 \leftarrow D \rightarrow G_n)$ is the plain derived rule $p(G_0 \xRightarrow{*} G_n)$ and $ac(G_n \Rightarrow G_{n+1}) = \text{Shift}(m_{n+1}, ac_{L_{n+1}})$ (see the diagram (2) below).



Definition 23 (Derived Rule [96]).

The derived rule of an AC-disregarding transformation $t : G_0 \xRightarrow{*} G_n$ has the form $\rho(t) = (p(t), ac(t))$ where $p(t)$ is the plain derived rule and $ac(t)$ is the derived nested application condition of t .

We need also a possibility to shift nested application conditions over rules to be able to compute an application condition of a transformation sequence. This kind of shifting is called *L-transformation* and is used to shift application conditions of a rule from its right-hand side to its left-hand side and vice versa [95].

Remark 5 (Shift of Nested Application Conditions over Rules [95]).

We use the *L-transformation* for shifting of nested application conditions over rules. The *L-transformation* is inductively defined as follows:

- For $ac_R = \text{true}$: $L(\rho, ac_R) = \text{true}$,
- For $ac_R = \exists(a, ac_H)$:

- $\mathbf{L}(\rho, \text{ac}_R) = \exists(b, \mathbf{L}(\rho^*, \text{ac}_H))$ if (r, a) has a pushout complement in (1) and $\rho^* = (G \leftarrow D \rightarrow H)$ is the derived rule by constructing the pushout (2),
 - $\mathbf{L}(\rho, \text{ac}_R) = \text{false}$ otherwise,
 - For $\text{ac}_R = \neg \text{ac}'_R$: $\mathbf{L}(\rho, \text{ac}_R) = \neg(\mathbf{L}(\rho, \text{ac}'_R))$,
 - For $\text{ac}_R = \bigwedge_{i \in \mathcal{I}} \text{ac}_{R,i}$: $\mathbf{L}(\rho, \text{ac}_R) = \bigwedge_{i \in \mathcal{I}} (\mathbf{L}(\rho, \text{ac}_{R,i}))$,
 - For $\text{ac}_R = \bigvee_{i \in \mathcal{I}} \text{ac}_{R,i}$: $\mathbf{L}(\rho, \text{ac}_R) = \bigvee_{i \in \mathcal{I}} (\mathbf{L}(\rho, \text{ac}_{R,i}))$,
- where l, r, l^*, r^* are inclusions and \mathcal{I} is some index set.

$$\begin{array}{ccc}
 \rho : L \xleftarrow{l} K \xrightarrow{r} \overset{\text{ac}_R}{\nabla} R & & \rho : \overset{\text{ac}_L}{\nabla} L \xleftarrow{l} K \xrightarrow{r} R \\
 \downarrow b \quad (2) \quad \downarrow \quad (1) \quad \downarrow a & & \downarrow a \quad (1) \quad \downarrow \quad (2) \quad \downarrow b \\
 \rho^* : G \xleftarrow{l^*} D \xrightarrow{r^*} H & & \rho^* : G \xleftarrow{l^*} D \xrightarrow{r^*} H \\
 \mathbf{L}(\rho^*, \text{ac}_H) & & \mathbf{R}(\rho^*, \text{ac}_G)
 \end{array}$$

The shifting of nested application conditions by **R**-transformations from left to right is defined symmetrically corresponding to the diagram above to the right.

Moreover, for every nested application condition ac_R (ac_L) over the object R (L) of a derived rule ρ , the **L**-transformation (**R**-transformation) transforms ac_R (ac_L) via ρ into the corresponding nested application condition $\mathbf{L}(\rho, \text{ac}_R)$ ($\mathbf{R}(\rho, \text{ac}_L)$) over the object L (R) such that we have for every direct transformation $G \xRightarrow{\rho, m} H$ that $m \models \mathbf{L}(\rho, \text{ac}_R) \Leftrightarrow m^* \models \text{ac}_R$ ($m^* \models \mathbf{R}(\rho, \text{ac}_L) \Leftrightarrow m \models \text{ac}_L$) for the corresponding comatch m^* .

$$\begin{array}{ccc}
 (\text{ac}_L) & & (\mathbf{R}(\rho, \text{ac}_L)) \\
 \mathbf{L}(\rho, \text{ac}_R) & & \text{ac}_R \\
 \rho : \overset{\nabla}{L} \xleftarrow{\quad} K \xrightarrow{\quad} \overset{\nabla}{R} & & \overset{\nabla}{L} \xleftarrow{\quad} K \xrightarrow{\quad} \overset{\nabla}{R} \\
 m \downarrow \quad \quad \downarrow \quad \quad \downarrow m^* & & \\
 G \xleftarrow{\quad} D \xrightarrow{\quad} H & &
 \end{array}$$

For an example showing the shifting of a nested application condition over a rule see e.g. Example 3 from [95].

2.3.2 Local Confluence Analysis for Transformations with Nested Application Conditions

An appropriate theory allowing the verification of local confluence for \mathcal{M} -adhesive transformation systems containing rules with nested application conditions was already introduced in [96]. In this subsection, we recall some fundamental concepts of this theory needed in the following to build up our theoretical results concerning local confluence analysis based on functorial property transfer.

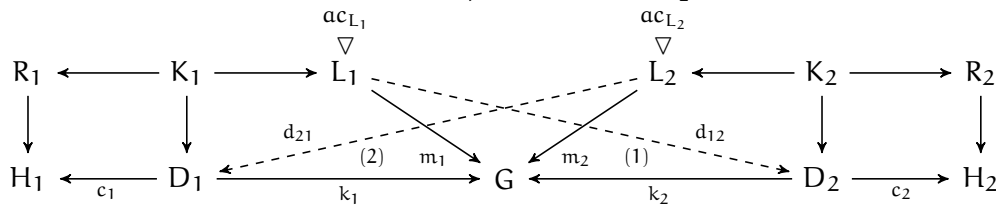
For the verification of local confluence for transformations with nested application conditions, we distinguish, similar to the plain case, two different ways. We use the Local Church-Rosser Theorem adapted to the case of transformations with nested application conditions given in Fact 4 for merging of parallel independent transformation steps as is already shown in [95], while we apply the adapted critical pair analysis from [96] for pairs of direct transformations, which are parallel dependent.

We consider again two kinds of independence property, namely the parallel and the sequential independence of transformation steps. As introduced before, we speak of parallel independence if two transformation steps can be applied in any order yielding the same result, while we speak of sequential independence if two transformation steps in a transformation sequence can be exchanged without affecting the overall result of this sequence [141, 88]. Intuitively, the definition of parallel and sequential independence of transformation steps with nested application conditions [98] means not only that each rule does not delete any element which is a part of the match morphism of the other rule, but also that after the application of one of the rules, the application condition of the other rule is still satisfiable.

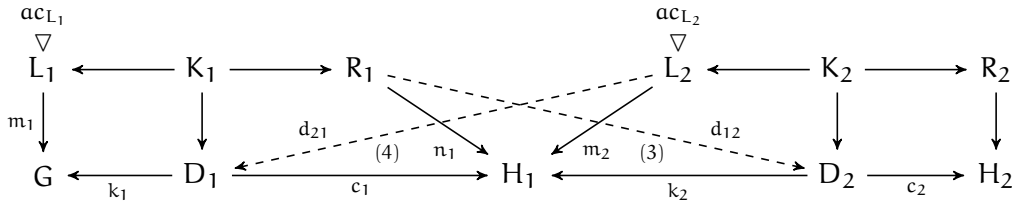
Definition 24 (Parallel and Sequential Independence of Direct Transformations with Nested Application Conditions [98]).

Consider an \mathcal{M} -adhesive transformation system $AS = (\mathbf{C}, \mathcal{M}, P)$.

- Two direct transformations $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ with nested application conditions are parallel independent if there exist morphisms $d_{12} : L_1 \rightarrow D_2$, $d_{21} : L_2 \rightarrow D_1$ such that (1) and (2) commute and $c_2 \circ d_{12} \models ac_{L_1}$, $c_1 \circ d_{21} \models ac_{L_2}$.



- Two direct transformations $G \xleftarrow{\rho_1, m_1} H_1 \xrightarrow{\rho_2, m_2} H_2$ with nested application conditions are sequentially independent if there exist morphisms $d_{12} : R_1 \rightarrow D_2$, $d_{21} : L_2 \rightarrow D_1$ such that (3) and (4) commute and $c_2 \circ d_{12} \models \mathbf{R}(\rho_1, ac_{L_1})$, $k_1 \circ d_{21} \models ac_{L_2}$.



For an example showing two parallel independent transformations for rules with nested application conditions see e.g. Example 4 from [95].

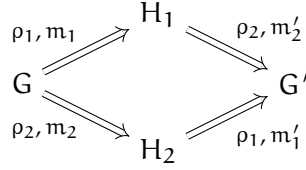
The corresponding Local Church-Rosser Theorem for transformation steps with nested application conditions is then formulated according to [95] as follows.

Fact 4 (Local Church-Rosser Theorem for Transformations with Nested Application Conditions [95]).

Given an \mathcal{M} -adhesive transformation system $AS = (\mathbf{C}, \mathcal{M}, P)$ where P are rules with nested application conditions.

- Consider two parallel independent direct transformations with nested application conditions $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$. Then there are an object G' and direct transformations $H_1 \xrightarrow{\rho_2, m'_2} G'$ and $H_2 \xrightarrow{\rho_1, m'_1} G'$ such that $G \xrightarrow{\rho_1, m_1} H_1 \xrightarrow{\rho_2, m'_2} G'$ and $G \xrightarrow{\rho_2, m_2} H_2 \xrightarrow{\rho_1, m'_1} G'$ are sequentially independent.

- Consider two sequentially independent direct transformations with nested application conditions $G \xrightarrow{\rho_1, m_1} H_1 \xrightarrow{\rho_2, m'_2} G'$. Then there are an object H_2 and direct transformations $G \xrightarrow{\rho_2, m_2} H_2 \xrightarrow{\rho_1, m'_1} G'$ such that $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$ are parallel independent.



To be able to consider critical pairs of transformations with nested application conditions, we first need to define the notion of a weak critical pair leading to the specific nested application conditions, called *extension* and *conflict inducing* nested application conditions. According to [96], the notion of a weak critical pair of transformations with nested application conditions gives us intuitively all minimal contexts of all pairs of AC-disregarding rule applications. This is necessary since all pairs of rule applications are potentially not confluent, even if they are parallel independent for the case if we disregard nested application conditions.

Definition 25 (Weak Critical Pair of Transformations with Nested Application Conditions [96]).

Consider derived rules ρ_1 and ρ_2 with nested application conditions ac_{L_1} and ac_{L_2} over their corresponding left-hand sides. A weak critical pair of ρ_1 and ρ_2 is a pair $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ of AC-disregarding transformations, where the pair of morphisms (o_1, o_2) is in \mathcal{E}'_1 . Every weak critical pair induces nested application conditions ac_K and ac_K^* on K defined by

$$ac_K = \mathbf{Shift}(o_1, ac_{L_1}) \wedge \mathbf{Shift}(o_2, ac_{L_2}),$$

called *extension nested application condition*, and

$$ac_K^* = \neg(ac_{K, d_{21}}^* \wedge ac_{K, d_{12}}^*),$$

called *conflict-inducing nested application condition*, with $ac_{K, d_{12}}^*$ and $ac_{K, d_{21}}^*$ given as follows:

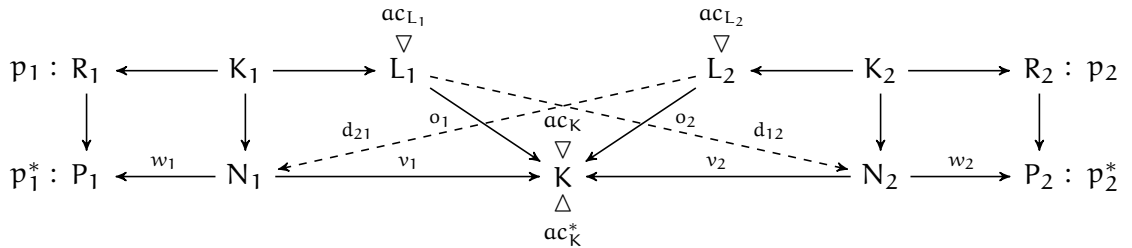
if $(\exists d_{12} : L_1 \rightarrow N_2. v_2 \circ d_{12} = o_1)$ then

$$ac_{K, d_{12}}^* = \mathbf{L}(p_2^*, \mathbf{Shift}(w_2 \circ d_{12}, ac_{L_1})) \text{ else } ac_{K, d_{12}}^* = \text{false},$$

if $(\exists d_{21} : L_2 \rightarrow N_1. v_1 \circ d_{21} = o_2)$ then

$$ac_{K, d_{21}}^* = \mathbf{L}(p_1^*, \mathbf{Shift}(w_1 \circ d_{21}, ac_{L_2})) \text{ else } ac_{K, d_{21}}^* = \text{false},$$

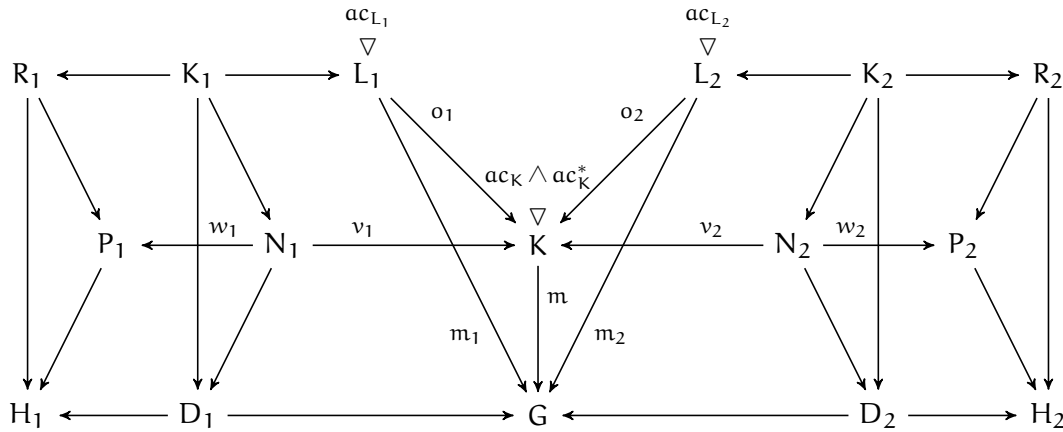
where the plain derived rules $p_1^* = (K \xleftarrow{v_1} N_1 \xrightarrow{w_1} P_1)$ and $p_2^* = (K \xleftarrow{v_2} N_2 \xrightarrow{w_2} P_2)$ are defined by the corresponding double pushouts (see the diagram below).



We use extension and conflict-inducing nested application conditions in the sense of [96] to characterize possible conflicts for the application of rules with nested application conditions. The satisfaction of an extension nested application condition (ac_K) means that by extension of K to some context G , the matches $m \circ o_i$ for $i \in \{1, 2\}$ satisfy their associated nested application conditions ac_{L_1} and ac_{L_2} . Furthermore, if we obtain by the described extension two parallel independent direct transformations $H_1 \xrightarrow{\rho_1, m \circ o_1} G \xrightarrow{\rho_2, m \circ o_2} H_2$ for the case if we disregard nested application conditions, then the corresponding conflict-inducing nested application condition (ac_K^*) leads to the parallel dependency of $H_1 \xrightarrow{\rho_1, m \circ o_1} G \xrightarrow{\rho_2, m \circ o_2} H_2$ in the case if we consider nested application conditions. The existence of the morphisms d_{12} or d_{21} means then that one of the two considered transformation rules is not applicable anymore when considering the nested application conditions ac_{L_1} and ac_{L_2} . The reason for this is that the nested application condition ac_K^* is not satisfiable by any embedding morphism m when determining whether this weak critical pair is also a critical pair according to the following definition.

Definition 26 (Critical Pair of Transformations with Nested Application Conditions [96]).

Consider derived rules ρ_1 and ρ_2 with nested application conditions ac_{L_1} and ac_{L_2} over their corresponding left-hand sides. A critical pair of ρ_1 and ρ_2 is a weak critical pair $P_1 \xrightarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ with induced extension and conflict-inducing nested application conditions on K , ac_K and ac_K^* , respectively, if there exists a morphism $m : K \rightarrow G$ in \mathcal{M} such that $m \models ac_K \wedge ac_K^*$ and $m_i = m \circ o_i$ for $i \in \{1, 2\}$ satisfies the gluing condition, i.e., m_i has a pushout complement D_i with respect to the plain derived rule p_i .



From [96] we know that a weak critical pair is also a critical pair if and only if we can extend it to a pair of parallel dependent transformations with nested application conditions, i.e., a weak critical pair $P_1 \xrightarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ disregards nested application conditions while the extension to $H_1 \xrightarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ satisfies ac_{L_1} of ρ_1 and ac_{L_2} of ρ_2 . Moreover, it holds according to [96] that critical pairs as introduced before are complete, i.e., each of them can be embedded in at least one span of parallel dependent transformations.

In the following, we introduce the general notion of compatibility of nested application conditions as well as the special kind of such compatibility, the AC-compatibility, that we especially need to be able to define the so-called strict AC-confluence of critical pairs [96].

The strict AC-confluence is the corresponding notion of strict confluence adapted to the case of transformations with nested application conditions.

Definition 27 (Compatibility of Nested Application Conditions).

Two nested application conditions are compatible if for every match morphism that satisfies ac_P we have that this match morphism also satisfies ac'_P , i.e.,

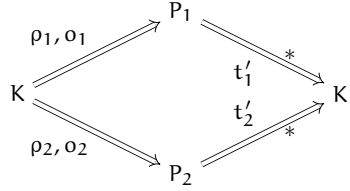
$$(ac_P \Rightarrow ac'_P) \Leftrightarrow \forall p : P \rightarrow G. ((p \models ac_P) \Rightarrow (p \models ac'_P))$$

Definition 28 (AC-Compatibility of AC-Disregarding Transformations [96]).

Consider a critical pair $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ of derived rules ρ_1 and ρ_2 with induced extension and conflict-inducing nested application conditions on K , ac_K and ac_K^* , respectively. Then the AC-compatibility is defined by the following implication

$$(ac_K \wedge ac_K^*) \Rightarrow (ac(t_1) \wedge ac(t_2))$$

where $t_i \triangleq K \xrightarrow{\rho_i, o_i} P_i \xrightarrow{t'_i} K'$ are extended AC-disregarding transformations with derived nested application conditions $ac(t_i)$ on K for $i \in \{1, 2\}$.



The following definition recalls the notion of strict AC-confluence, which is based on the plain case of strict confluence given in Definition 14 and is important for the verification of local confluence for transformations with nested application conditions according to [96].

Definition 29 (Strict AC-Confluence of Critical Pairs [96]).

A critical pair $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ of derived rules ρ_1 and ρ_2 with induced extension and conflict-inducing nested application conditions on K , ac_K and ac_K^* , respectively, is called strictly AC-confluent, if it is plain strictly confluent, i.e., strictly confluent in the sense of [88] (as recalled in Definition 14) with AC-disregarding transformations t'_1 and t'_2 such that the extended AC-disregarding transformations $t_i \triangleq K \xrightarrow{\rho_i, o_i} P_i \xrightarrow{t'_i} K'$ with derived nested application conditions $ac(t_i)$ on K for $i \in \{1, 2\}$ are AC-compatible.

An example for constructing a critical pair in the category of hypergraphs and showing its strict AC-confluence is given in Example 8 in Subsection 2.4.2.

We use the adapted Local Confluence Theorem given in the following to analyze whether an \mathcal{M} -adhesive transformation system is locally confluent for the case of transformations with nested application conditions. The proof for this theorem is given in [96].

Fact 5 (Local Confluence Theorem for Transformation Systems with Nested Application Conditions [96]).

An \mathcal{M} -adhesive transformation system with nested application conditions is locally confluent if all its critical pairs are strictly AC-confluent.

2.4 CONCRETE INSTANTIATIONS OF \mathcal{M} -ADHESIVE TRANSFORMATION SYSTEMS

In this section, we recall several well-known instantiations for the framework of \mathcal{M} -adhesive transformation systems, to which we apply in the following our new analysis approach. For our applications, we are interested especially in the transformation systems over typed attributed graphs, hypergraphs and a special kind of Petri nets, called Petri nets with individual tokens (short PTI nets).

2.4.1 *Typed Attributed Graph Transformation Systems*

Graphs are a well suitable formalism for various application domains. Different kinds of graphs are used for the representation of a particular state of a system and the abstract syntax of visual models [122]. Thereby graphs can be used on the type resp. on the instance level to define types of graph elements resp. the instances of these types. A given type graph determines the structure as well as the node and edge types allowed to occur in the graphs at the instance level. Moreover, for many modeling purposes it should be possible to equip the nodes and/or edges of graphs with attributes.

Another application domain is the modeling and verification of graph based algorithms using graph programs, as introduced in [137, 283, 258], allowing to reason about graph algorithms, expressed on the domain of graphs directly, without introducing syntactical programming languages to this problem. Furthermore, graph-based techniques are successfully used e.g. for modeling of inter-networking systems [295] as well as for the development of knowledge-based design tools [282].

The modification of system states is mostly defined as rules of a certain form, describing pre- and post-conditions for the modifications to be applied. For systems, where states are modeled using a graph formalism, these rules are graph transformation rules belonging to the same graph formalism. As already discussed in Section 2.1, graph transformation is often used as a modeling technique in software engineering and as a metalanguage to specify and implement visual modeling techniques such as UML [144]. Other applications include, for example, parsing of visual languages [25] and the automated transfer of visual models into code or into different semantical domains [102, 303, 183]. The approach presented in this thesis also provides a semantics for hypergraph and PTI net transformation systems by translating them on functorial way into the semantical domain of typed attributed graph transformation systems.

There are several approaches formalizing (typed) attributed graphs. For instance, in [202] Löwe et al. represent the graph part of an attributed graph as an algebra that extends the given data type algebra. In [258] Plump and Steinert define attributed graphs using labeled graphs and transform them by application of rule schemata dealing with calculations on labels. In [144] resp. [88] a typed attributed graph with node resp. with node and edge attribution is seen as a pair consisting of a graph and a data algebra, whose values are contained as nodes in the graph. Finally, the so-called symbolic graph transformation approach [240] allows for the separation of the graph and the algebra parts of typed attributed graphs. In this thesis, we follow the approach introduced in [88].

The aforementioned graph formalisms consist not only of definitions for the description of static graphs belonging to that formalism. They are also equipped with defini-

$\Sigma\text{-nat}$	$\text{sorts : } \text{nat}$ $\text{opns : } z : \rightarrow \text{nat}$ $\text{succ : nat} \rightarrow \text{nat}$
NAT	$\text{NAT}_{\text{nat}} = \mathbb{N}$ $z_{\text{nat}} = 0 \in \mathbb{N}$ $\text{succ}_{\text{nat}} : \mathbb{N} \rightarrow \mathbb{N} \quad x \mapsto x + 1$

Figure 7: Data type signature $\Sigma\text{-nat}$ and algebra NAT

tions for the transformation of the included graphs based on the DPO [88] resp. SPO [265, 200, 199] approach.

In this subsection, we introduce first two different kinds of graphs, namely, E-graphs and attributed graphs, leading together to the notion of typed attributed graphs. Afterwards, we review the \mathcal{M} -adhesive category $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$ of typed attributed graphs with the class \mathcal{M} of all injective typed attributed graph morphisms with isomorphism on the data type part, which is shown to be adhesive HLR and hence also \mathcal{M} -adhesive in [88]. The customized categories of typed attributed graphs $\mathbf{AGraphs}_{\text{HGTG}}$ resp. $\mathbf{AGraphs}_{\text{PNTG}}$ with the special hypergraph resp. PTI net attributed type graphs HGTG resp. PNTG , we will use in the following chapters as the image categories for the definition of \mathcal{M} -functors \mathcal{F}_{HG} resp. \mathcal{F}_{PTI} . Finally, we recall the definitions of boundary and context objects in $\mathbf{AGraphs}_{\text{ATG}}$ leading to the initial pushout construction in $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$.

According to [88], the basis for modeling of attributed graphs with attributes for nodes and edges is a definition of E-graphs, which are used to determine for each edge and node occurring in concrete instances the possibly assigned attributes. An E-graph consists of two different kinds of nodes (*graph* and *data nodes*) and three different kinds of edges (*graph edges* as well as *edges for node* and *edge attribution*). The formal definition of E-graphs and the corresponding morphisms is given in the following.

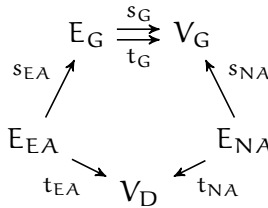
Definition 30 (E-Graphs and E-Graph Morphisms [88]).

Consider a signature E declaring the sets and functions shown in the diagram below.

- An E-graph G is defined as

$$G = (V_G^G, V_D^G = \mathbb{N}, E_G^G, E_{\text{NA}}^G, E_{\text{EA}}^G, (s_j^G, t_j^G)_{j \in \{G, \text{NA}, \text{EA}\}}) \text{ where}$$

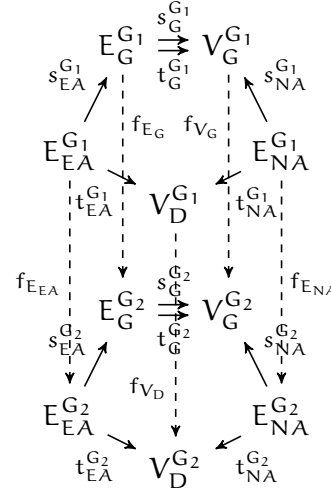
- V_G^G resp. V_D^G are sets of graph resp. data nodes of G ,
- $E_G^G, E_{\text{NA}}^G, E_{\text{EA}}^G$ are sets of graph edges as well as node attribute and edge attribute edges of G ,
- s_j^G, t_j^G for $j \in \{G, \text{NA}, \text{EA}\}$ are the corresponding source and target functions for the edges.



- Consider two E-graphs G_1 and G_2 with $G_k = (V_G^{G_k}, V_D^{G_k} = \mathbb{N}, E_G^{G_k}, E_{\text{NA}}^{G_k}, E_{\text{EA}}^{G_k}, (s_j^{G_k}, t_j^{G_k})_{j \in \{G, \text{NA}, \text{EA}\}})$ for $k \in \{1, 2\}$. An E-graph morphism $f : G_1 \rightarrow G_2$ is a tuple $(f_{V_G}, f_{V_D}, f_{E_G}, f_{E_{\text{NA}}}, f_{E_{\text{EA}}})$ ¹⁹ with $f_{V_i} : V_i^{G_1} \rightarrow V_i^{G_2}$ and $f_{E_j} : E_j^{G_1} \rightarrow E_j^{G_2}$ for $i \in \{G, D\}$ and $j \in \{G, \text{NA}, \text{EA}\}$ such that f commutes with all source and target functions, i.e.,

¹⁹ For clarity of the diagram below, the typed attributed graph morphism components are depicted with dashed arrows.

- $f_{V_D} \circ t_{EA}^{G_1} = t_{EA}^{G_2} \circ f_{EEA}$,
- $t_{NA}^{G_2} \circ f_{ENA} = f_{V_D} \circ t_{NA}^{G_1}$,
- $s_{NA}^{G_2} \circ f_{ENA} = f_{V_G} \circ s_{NA}^{G_1}$,
- $s_{EA}^{G_2} \circ f_{EEA} = f_{E_G} \circ s_{EA}^{G_1}$,
- $s_G^{G_2} \circ f_{E_G} = f_{V_G} \circ s_G^{G_1}$,
- $t_G^{G_2} \circ f_{E_G} = f_{V_G} \circ t_G^{G_1}$.



- All E-graphs as objects and all E-graph morphisms between them define the category of E-graphs **EGraphs**.

In order to define attributed graphs with attributes for nodes and edges, we need according to [88] an extension of E-graphs to attributed graphs by combining an E-graph with an algebra over a given data signature. The data sets of the algebra, which are defined for the sort symbols of the given signature, can then be used for node and edge attribution. In the following, we consider for our purpose only the signature $\Sigma\text{-nat}$ and the corresponding algebra NAT given in Figure 7.

Definition 31 (Attributed Graphs and Attributed Graph Morphisms [88]).

- An attributed graph is a pair (G, D) of an E-graph G over the signature E and a $\Sigma\text{-nat}$ algebra D , where in the following we only use $D = T_{\Sigma\text{-nat}} \cong NAT$ (with the term algebra $T_{\Sigma\text{-nat}}$ and the ordinary natural numbers algebra NAT from Figure 7) for the occurring attribute values.
- For two attributed graphs $AG^1 = (G_1, D^1)$ and $AG^2 = (G_2, D^2)$, an attributed graph morphism $f : AG^1 \rightarrow AG^2$ is a pair $f = (f_G, f_D)$ with an E-graph morphism $f_G : G_1 \rightarrow G_2$ and an algebra homomorphism $f_D : D^1 \rightarrow D^2$ such that the diagram (1) below commutes with inclusions $f_i : D_{nat}^i \rightarrow V_D^{G_i}$ for $i \in \{1, 2\}$ and an identity morphism $f_{D, nat} : D_{nat}^1 \rightarrow D_{nat}^2$.

$$\begin{array}{ccc}
 D_{nat}^1 & \xrightarrow{f_{D, nat}} & D_{nat}^2 \\
 f_1 \downarrow & (1) & \downarrow f_2 \\
 V_D^{G_1} & \xrightarrow{f_{G, V_D}} & V_D^{G_2}
 \end{array}$$

- All attributed graphs as objects and all attributed graph morphisms between them define the category of attributed graphs **AGraphs**.

According to [88], the notion of attributed graphs combined with the typing concept leads to the notion of typed attributed graphs, where attributed graphs are typed over an attributed type graph ATG . The formal definitions of ATG and typed attributed graphs over ATG with the corresponding morphisms are given below.

Definition 32 (Attributed Type Graph ATG [88]).

An attributed type graph $ATG = (TG, D_{fin})$ consists of a type graph TG and the final $\Sigma\text{-nat}$

algebra D_{fin}^{20} . ATG is attributed over the final Σ -nat algebra D_{fin} and defines the set of all possible types that can be used.

Definition 33 (Typed Attributed Graphs over Type Graph ATG and Typed Attributed Graph Morphisms [88]).

Consider an attributed type graph ATG as given in Definition 32 above.

- A typed attributed graph is a pair $(AG, type)$ of an attributed graph $AG = (G, D)$, where G is an E-graph and D is the Σ -nat algebra NAT (see Figure 7), and an **AGraphs**-morphism $type : AG \rightarrow ATG$.
- For two typed attributed graphs $TAG^1 = (AG^1, type^1)$ and $TAG^2 = (AG^2, type^2)$, a typed attributed graph morphism $f' : TAG^1 \rightarrow TAG^2$ is an attributed graph morphism $f : AG^1 \rightarrow AG^2$ such that the compatibility with typing morphisms holds, i.e., $type^2 \circ f = type^1$ (see the diagram below).

$$\begin{array}{ccc} AG^1 & \xrightarrow{f} & AG^2 \\ & \searrow \scriptstyle type^1 & \swarrow \scriptstyle type^2 \\ & & ATG \end{array} \quad =$$

- All typed attributed graphs as objects and all typed attributed graph morphisms between them define the category of typed attributed graphs **AGraphs**_{ATG}.

To be able to use **AGraphs**_{ATG} over specific attributed type graphs as the target category for our functor constructions in the following chapters, we need the category **AGraphs**_{ATG} together with the suitable class of monomorphisms to be \mathcal{M} -adhesive. According to [88], this is the case as given in the following remark.

Remark 6 ((AGraphs**_{ATG}, \mathcal{M}) is an \mathcal{M} -adhesive Category).**

According to [88], the category $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ of typed attributed graphs with the class \mathcal{M} of all injective typed attributed graph morphisms with isomorphism on the data type part is adhesive HLR and hence also \mathcal{M} -adhesive, where pushouts and pullbacks along \mathcal{M} -morphisms are constructed componentwise in the graph part.

Considering special cases of $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ from our applications with $ATG = HGTG$ for the description of hypergraphs (see Figure 29) resp. $ATG = PNTG$ for the description of PTI nets (see Figure 64), some examples for the typed attributed graph representation and typed attributed graph morphisms are given in Figure 34 resp. Figure 68²¹. Furthermore, two examples for the corresponding *type*-morphism components $type_{V_G}$ and $type_{E_G}$ are given in Figure 31 and Figure 66 for **AGraphs**_{HGTG} and **AGraphs**_{PNTG}, respectively, where $type_{V_G}$ and $type_{E_G}$ map each component of the given graphs to their corresponding components in the type graph. Finally, Figure 34 and Figure 68 show examples for pushouts and pullbacks²² in **AGraphs**_{HGTG} and **AGraphs**_{PNTG}, respectively.

For rule-based transformation of typed attributed graphs we use again the DPO approach introduced in detail in [88]. As already discussed before, in order to apply a rule via a match morphism to some typed attributed graph, we have to check whether the gluing condition is satisfied. The gluing condition can be formulated on the abstract level of \mathcal{M} -adhesive transformation systems using initial pushouts as already introduced

²⁰ In general, D_{fin} is a final algebra over a given data signature.

²¹ The meaning of the depicted elements of the two special type graphs will be explained in detail in Sections 5.1 and 8.1.

²² Both diagrams are pushouts and pullbacks at once.

in Section 2.2. According to [88], the \mathcal{M} -adhesive category $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$ has initial pushouts, which can be constructed using the boundary and context objects given in Facts 6 and 7, respectively.

Fact 6 (Boundary Object in $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$ [88]).

Consider a typed attributed graph morphism $f' : L' \rightarrow G'$. According to [88] (see Definition 10.5 there ²³), the boundary object $B' = ((B'_0, \text{NAT}), \text{type}^{B'})$ with the boundary points B'_0 is given by the intersection of suitable attributed subgraphs B'' of L' :

$$B'_0 = (V_G^{B'_0}, V_D^{B'_0} = \mathbb{N}, E_G^{B'_0}, E_{\text{NA}}^{B'_0}, E_{\text{EA}}^{B'_0}, (s_j^{B'_0}, t_j^{B'_0})_{j \in \{G, \text{NA}, \text{EA}\}}) \text{ defined by}$$

$$B'_0 = \bigcap \{B'' \subseteq L' \mid V_D^{L'} = V_D^{B''} \wedge V_G^{B''} \subseteq V_G^{B''} \wedge E_G^{B''} \subseteq E_G^{B''} \wedge E_{\text{NA}}^{B''} \subseteq E_{\text{NA}}^{B''} \wedge E_{\text{EA}}^{B''} \subseteq E_{\text{EA}}^{B''}\}.$$

The components of $B''_0 = (V_G^{B''_0}, V_D^{B''_0} = \mathbb{N}, E_G^{B''_0}, E_{\text{NA}}^{B''_0}, E_{\text{EA}}^{B''_0}, (s_j^{B''_0}, t_j^{B''_0})_{j \in \{G, \text{NA}, \text{EA}\}})$ are built up by the dangling and identification points as follows:

$$\begin{aligned} V_G^{B''_0} &= \{a \in V_G^{L'} \mid [\exists a' \in E_{\text{NA}}^{G'} \setminus f'_{\text{EA}}(E_{\text{NA}}^{L'}) . f'_{V_G}(a) = s_{\text{NA}}^{G'}(a')] \\ &\quad \vee [\exists a' \in E_G^{G'} \setminus f'_{E_G}(E_G^{L'}) . f'_{V_G}(a) = s_G^{G'}(a') \vee f'_{V_G}(a) = t_G^{G'}(a')] \\ &\quad \vee [\exists a' \in V_G^{L'} . a \neq a' \wedge f'_{V_G}(a) = f'_{V_G}(a')]\}, \\ E_G^{B''_0} &= \{a \in E_G^{L'} \mid [\exists a' \in E_{\text{EA}}^{G'} \setminus f'_{E_{\text{EA}}}(E_{\text{EA}}^{L'}) . f'_{E_G}(a) = s_{\text{EA}}^{G'}(a')] \\ &\quad \vee [\exists a' \in E_G^{L'} . a \neq a' \wedge f'_{E_G}(a) = f'_{E_G}(a')]\}, \\ E_{\text{NA}}^{B''_0} &= \{a \in E_{\text{NA}}^{L'} \mid \exists a' \in E_{\text{NA}}^{L'} . a \neq a' \wedge f'_{E_{\text{NA}}}(a) = f'_{E_{\text{NA}}}(a')\}, \\ E_{\text{EA}}^{B''_0} &= \{a \in E_{\text{EA}}^{L'} \mid \exists a' \in E_{\text{EA}}^{L'} . a \neq a' \wedge f'_{E_{\text{EA}}}(a) = f'_{E_{\text{EA}}}(a')\}, \\ s_G^{B''_0}, t_G^{B''_0} : E_G^{B''_0} \rightarrow V_G^{B''_0} &\text{ are restrictions of } s_G^{L'}, t_G^{L'} : E_G^{L'} \rightarrow V_G^{L'}, \\ s_{\text{NA}}^{B''_0} : E_{\text{NA}}^{B''_0} \rightarrow V_G^{B''_0} &\text{ is a restriction of } s_{\text{NA}}^{L'} : E_{\text{NA}}^{L'} \rightarrow V_G^{L'}, \\ t_{\text{NA}}^{B''_0} : E_{\text{NA}}^{B''_0} \rightarrow V_D^{B''_0} &\text{ is a restriction of } t_{\text{NA}}^{L'} : E_{\text{NA}}^{L'} \rightarrow V_D^{L'}, \\ s_{\text{EA}}^{B''_0} : E_{\text{EA}}^{B''_0} \rightarrow E_G^{B''_0} &\text{ is a restriction of } s_{\text{EA}}^{L'} : E_{\text{EA}}^{L'} \rightarrow E_G^{L'}, \\ t_{\text{EA}}^{B''_0} : E_{\text{EA}}^{B''_0} \rightarrow V_D^{B''_0} &\text{ is a restriction of } t_{\text{EA}}^{L'} : E_{\text{EA}}^{L'} \rightarrow V_D^{L'}, \end{aligned}$$

and $b' : B' \rightarrow L'$ is an inclusion.

$$\begin{array}{ccc} B' & \xrightarrow{b'} & L' \\ & & \downarrow f' \\ & & G' \end{array}$$

Fact 7 (Context Object in $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$ [88]).

Consider a typed attributed graph morphism $f' : L' \rightarrow G'$ and the boundary object B' constructed according to Fact 6 above. The context object C' is a typed attributed subgraph of G' defined as follows:

$$C' = ((C'_0, \text{NAT}), \text{type}^{C'}) \text{ with}$$

$$C'_0 = (V_G^{C'_0}, V_D^{C'_0} = \mathbb{N}, E_G^{C'_0}, E_{\text{NA}}^{C'_0}, E_{\text{EA}}^{C'_0}, (s_j^{C'_0}, t_j^{C'_0})_{j \in \{G, \text{NA}, \text{EA}\}}) \text{ where}$$

²³ In Fact 6 we corrected some typing errors concerning indices of the morphism f' in the end part of some conditions of components $V_G^{B''_0}$ and $E_G^{B''_0}$.

$$\begin{aligned}
V_G^{C'_0} &= (V_G^{G'} \setminus f'_{V_G}(V_G^{L'})) \cup f'_{V_G}(V_G^{B'}), \\
V_D^{C'_0} &= V_D^{G'}, \\
E_G^{C'_0} &= (E_G^{G'} \setminus f'_{E_G}(E_G^{L'})) \cup f'_{E_G}(E_G^{B'}), \\
E_{NA}^{C'_0} &= (E_{NA}^{G'} \setminus f'_{E_{NA}}(E_{NA}^{L'})) \cup f'_{E_{NA}}(E_{NA}^{B'}), \\
E_{EA}^{C'_0} &= (E_{EA}^{G'} \setminus f'_{E_{EA}}(E_{EA}^{L'})) \cup f'_{E_{EA}}(E_{EA}^{B'}), \\
s_G^{C'_0}, t_G^{C'_0} : E_G^{C'_0} &\rightarrow V_G^{C'_0} \text{ are restrictions of } s_G^{G'}, t_G^{G'} : E_G^{G'} \rightarrow V_G^{G'}, \\
s_{NA}^{C'_0} : E_{NA}^{C'_0} &\rightarrow V_G^{C'_0} \text{ is a restriction of } s_{NA}^{G'} : E_{NA}^{G'} \rightarrow V_G^{G'}, \\
t_{NA}^{C'_0} : E_{NA}^{C'_0} &\rightarrow V_D^{C'_0} \text{ is a restriction of } t_{NA}^{G'} : E_{NA}^{G'} \rightarrow V_D^{G'}, \\
s_{EA}^{C'_0} : E_{EA}^{C'_0} &\rightarrow E_G^{C'_0} \text{ is a restriction of } s_{EA}^{G'} : E_{EA}^{G'} \rightarrow E_G^{G'}, \\
t_{EA}^{C'_0} : E_{EA}^{C'_0} &\rightarrow V_D^{C'_0} \text{ is a restriction of } t_{EA}^{G'} : E_{EA}^{G'} \rightarrow V_D^{G'},
\end{aligned}$$

and $c' : C' \rightarrow G'$ is an inclusion.

$$\begin{array}{ccc}
B' & \xrightarrow{b'} & L' \\
\downarrow & & \downarrow f' \\
C' & \xrightarrow{c'} & G'
\end{array}$$

The following fact from [88] summarizes the construction of initial pushouts in the category $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$. For the proof of the fact see [88].

Fact 8 (Initial Pushout in $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$ [88]).

Consider a typed attributed graph morphism $f' : L' \rightarrow G'$. The diagram (1) given below with the boundary object B' constructed according to Fact 6, the context object C' constructed according to Fact 7, inclusions $b' : B' \rightarrow L'$, $c' : C' \rightarrow G'$ and the morphism $g' : B' \rightarrow C'$ given by $g'_j(x) = (f'_j \circ b'_j)(x)$ for $j \in \{V_G, V_D, E_G, E_{NA}, E_{EA}\}$ is well-defined and is an initial pushout over f' in $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$.

$$\begin{array}{ccc}
B' & \xrightarrow{b'} & L' \\
g' \downarrow & (1) & \downarrow f' \\
C' & \xrightarrow{c'} & G'
\end{array}$$

Two examples for the initial pushout construction in the customized \mathcal{M} -adhesive typed attributed graph categories $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ resp. $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ ²⁴ are given in Figure 40 resp. Figure 73, where the outer diagram to the right consisting of typed attributed graphs B' , $\mathcal{F}_{\text{HG}}(L)$ resp. $\mathcal{F}_{\text{PTI}}(L)$, $\mathcal{F}_{\text{HG}}(G)$ resp. $\mathcal{F}_{\text{PTI}}(G)$, C' and morphisms between them forms an initial pushout in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ resp. $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$.

²⁴ The \mathcal{M} -adhesive categories $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ and $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ are subcategories of $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$ for the monomorphism subclass $\mathcal{M} = \mathcal{M}_2$ of all injective typed attributed graph morphisms with identical algebra homomorphism.

2.4.2 Hypergraph Transformation Systems

Hypergraphs are a widely used graph-based formalism introduced for the modeling of various kinds of systems. The formalism of hypergraphs is quite versatile and its main strength is its built-in capability of succinctly representing relationships between an unbounded number of nodes, that is, an edge (also called a *hyperedge*) in a hypergraph connects an unbounded number of source nodes with an unbounded number of target nodes. In this sense, hypergraphs where each edge connects precisely one source node with one target node can be understood as ordinary graphs.

There are many areas where hypergraphs can be successfully deployed. Hypergraphs have shown to be appropriate e.g. for the evaluation of functional expressions where they allow a function with n arguments to be modeled by a hyperedge with one source node and n target nodes [253]. Moreover, hypergraphs are a suitable formalism for the modeling of inter-networking systems, where components of a system are represented by hyperedges and hypergraph nodes representing communication ports between different components model the network environment of these components [149, 148, 295]. In this case, the fact that a node is shared by several hyperedges means that the involved components have a possibility to interact using the network communication infrastructure [295]. In [196] hypergraphs have been used in the context of declustering problems for the implementation of I/O parallelization algorithms in the context of parallel databases and high performance systems, while in [120] hypergraphs have been employed for the analysis of relational databases. Furthermore, in [37, 40] hypergraphs have been deployed in the context of service-oriented architecture for business processes for modeling services, their mutual dependencies as well as their resource and capacity requirements.

Hypergraph transformation systems have been used in many application domains to model the behavior of distributed or concurrent systems [271, 147, 295, 188], to model machine learning processes [279], and, furthermore, hyperedge replacement systems, as a restricted form of hypergraph transformation systems, can be seen as graphical context-free Chomsky grammars [181]. In the domain of *Architectural Design Rewriting* (short *ADR*) [33, 34], hypergraphs and hypergraph rewriting rules are extensively used to model the dynamism of architectural designs. In the context of *Synchronized Hyperedge Replacement* (short *SHR*) [39, 54], frequently deployed for the modeling of software architectures and distributed systems (encodes amongst others the π -calculus [147], Ambient and Klam [295] as well as Fusion [188]), hypergraphs and their transformations are used for modeling systems and their behavior. In [163] a formalism, in fact a process calculus, for the stochastic modeling and simulation of biochemical reactions is introduced. This formalism extends the κ -calculus based on the idea of hyperedge replacement. Further application areas of hypergraphs and hypergraph transformations are already pointed out in the introduction.

In this subsection, we recall the concept of hypergraphs, discuss some of their characterization properties, review the \mathcal{M} -adhesive category (**HyperGraphs**, \mathcal{M}) of hypergraphs with the class \mathcal{M} of all injective hypergraph morphisms, which is shown to be \mathcal{M} -adhesive in [88], and consider such constructions on hypergraphs as pushouts, pullbacks, and initial pushouts. Finally, in the end of this subsection, we give an example for a small hypergraph transformation system, which we will use in the following as a running example for the application of our new theoretical results to hypergraph transformation systems.

Definition 34 (Hypergraphs and Hypergraph Morphisms [88]).

- A hypergraph G is defined as $G = (V_G, E_G, s_G, t_G)$ where
 - V_G is a set of hypergraph nodes,
 - E_G is a set of hyperedges, and
 - $s_G, t_G : E_G \rightarrow V_G^*$ are functions assigning the string $s_G(e)$ of source nodes resp. $t_G(e)$ of target nodes to each hyperedge e .
- Consider hypergraphs $G = (V_G, E_G, s_G, t_G)$ and $H = (V_H, E_H, s_H, t_H)$. A hypergraph morphism $f : G \rightarrow H$ is given by a tuple of functions $f = (f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H)$ such that the diagram below commutes with source and target functions, i.e.,

$$s_H \circ f_E = f_V^* \circ s_G \quad \text{and} \quad t_H \circ f_E = f_V^* \circ t_G$$

where the function $f_V^* : V_G^* \rightarrow V_H^*$ maps $\lambda \mapsto \lambda$ and $x_1 \dots x_n \mapsto f_V(x_1) \dots f_V(x_n)$ for the empty word λ .

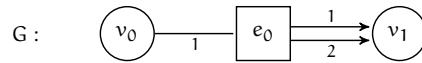
$$\begin{array}{ccc} E_G & \xrightarrow{s_G} & V_G^* \\ f_E \downarrow & \begin{array}{c} t_G \\ = \\ s_H \end{array} & \downarrow f_V^* \\ E_H & \xrightarrow{s_H} & V_H^* \\ & \begin{array}{c} t_H \end{array} & \end{array}$$

Note that we use a more general kind of hypergraphs with source and target functions, where hyperedges with incoming and outgoing arcs are allowed. From the theoretical point of view, it is not necessary to distinguish the source and the target functions. It is sufficient to assign one sequence of nodes to each hyperedge, which can be already done considering only one corresponding function. But, from the applicational point of view, for usage of labeled hypergraphs in some of our examples, it makes sense to distinguish the source and the target functions. Therefore, we use the definition of hypergraphs as given above.

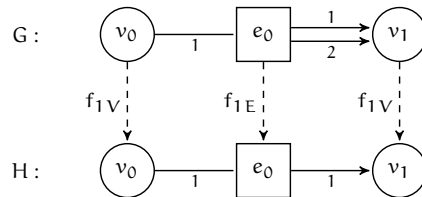
An example for the hypergraph representation and the meaning of hypergraph morphisms is given in the following.

Example 3 (Hypergraphs and Hypergraph Morphisms).

The picture below shows a hypergraph G with hypergraph nodes v_0 and v_1 and a hyperedge e_0 . As usual in the hypergraph notation, only the target nodes of a hyperedge are connected by arrows. We call the connectors between hypergraph nodes and hyperedges "hyperedge tentacles". The numbers on hyperedge tentacles denote the position of the node in the corresponding source resp. target string of the considered hyperedge.



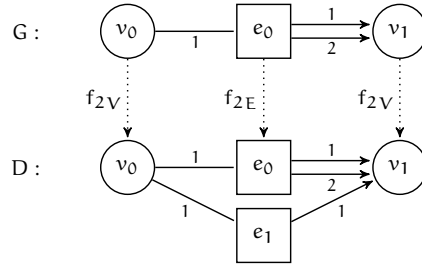
Consider another hypergraph H . The intuitive mapping $f_1 : G \rightarrow H$, as depicted in the picture below with dashed arrows, is no hypergraph morphism since the hyperedge e_0 in both cases has different strings of target nodes: $t_G(e_0) = v_1 \cdot v_1 \neq v_1 = t_H(e_0)$, which is not valid for hypergraph morphisms.



Consider another hypergraph D . The intuitive mapping $f_2 : G \rightarrow D$, as depicted in the picture below with dotted arrows, is a valid hypergraph morphism since the compatibility of f_2 with source and target functions s_G, s_D, t_G, t_D holds:

$$s_D(f_{2E}(e_0)) = s_D(e_0) = v_0 = f_{2V}^*(v_0) = f_{2V}^*(s_G(e_0))$$

$$t_D(f_{2E}(e_0)) = t_D(e_0) = v_1 \cdot v_1 = f_{2V}^*(v_1 \cdot v_1) = f_{2V}^*(t_G(e_0))$$



Note that the mapping $f_1 : G \rightarrow H$ given above would be a correct typed attributed graph morphism. Therefore, it is not advisable to use an intuitive translation of hypergraphs and hypergraph morphisms into typed attributed graphs and typed attributed graph morphisms by just representing hypergraph nodes and hyperedges by different types of graph nodes and hyperedge tentacles by graph edges. In this case, there is a risk that the graph morphisms, which have to be considered, do not correspond to the valid hypergraph morphisms.

To ensure the validity of hypergraph morphisms in our applications, we use the following characterization, which is easily implementable for a tool-supported identification.

Lemma 2 (Characterization of Hypergraph Morphisms).

1. Consider a hypergraph morphism $f = (f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H)$ according to Definition 34. Then the following two properties hold:
 - a) The number of incoming resp. outgoing arrows for each edge remains equal, i.e.,
 - i. $\forall e \in E_G, \forall n \in \mathbb{N}. (|s_G(e)|=n) \Leftrightarrow (|s_H(f_E(e))|=n),$
 - ii. $\forall e \in E_G, \forall n \in \mathbb{N}. (|t_G(e)|=n) \Leftrightarrow (|t_H(f_E(e))|=n)$ and
 - b) The morphism preserves/reflects the source and target components of every edge, i.e.,
 - i. $\forall v \in V_G, \forall e \in E_G, \forall n \leq |s_G(e)|. (s_G^n(e)=v) \Rightarrow (s_H^n(f_E(e))=f_V(v)),$
 - ii. $\forall v \in V_G, \forall e \in E_G, \forall n \leq |s_G(e)|. (s_H^n(f_E(e))=f_V(v)) \Leftrightarrow (s_G^n(e)=v)$ if f_V is injective,
 - iii. $\forall v \in V_G, \forall e \in E_G, \forall n \leq |t_G(e)|. (t_G^n(e)=v) \Rightarrow (t_H^n(f_E(e))=f_V(v)),$
 - iv. $\forall v \in V_G, \forall e \in E_G, \forall n \leq |t_G(e)|. (t_H^n(f_E(e))=f_V(v)) \Leftrightarrow (t_G^n(e)=v)$ if f_V is injective.
2. According to Definition 34, $f = (f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H)$ is a hypergraph morphism if the properties 1(b)i and 1(b)iii hold.

Proof.

The detailed proof for this lemma is given in [Appendix C](#) on [page 311](#). □

Hypergraph objects and morphisms as given in Definition 34 lead to the definition of the category **HyperGraphs**.

Definition 35 (Category HyperGraphs [88]).

All hypergraphs as objects and all hypergraph morphisms between them define the category of hypergraphs denoted as **HyperGraphs**.

Remark 7 ((HyperGraphs, \mathcal{M}) is an \mathcal{M} -adhesive Category).

According to [88, 94], the category $(\mathbf{HyperGraphs}, \mathcal{M})$ of hypergraphs with the class \mathcal{M} of all injective hypergraph morphisms is an adhesive HLR and hence also an \mathcal{M} -adhesive category, where pushouts and pullbacks are constructed componentwise (see the following definitions and examples in Figure 8 and Figure 9) and there exist initial pushouts as follows from Lemma 7 (see an example in Figure 10).

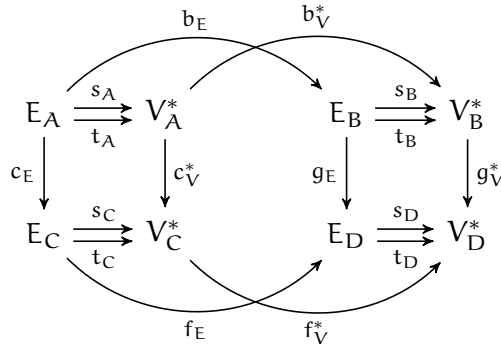
In the following, we recall the definition of hypergraph pushouts, which we adopt from [253] in slightly different notation. Subsequently, we show in Lemma 3 that the considered construction leads to a valid pushout diagram in the category **HyperGraphs**.

Definition 36 (Construction of Hypergraph Pushouts).

Let $b : A \rightarrow B$ and $c : A \rightarrow C$ be hypergraph morphisms. Define $D = (V_D, E_D, s_D, t_D)$ from the diagram below as follows:

$$\begin{array}{ccc} A & \xrightarrow{b} & B \\ c \downarrow & (PO) & \downarrow g \\ C & \xrightarrow{f} & D \end{array}$$

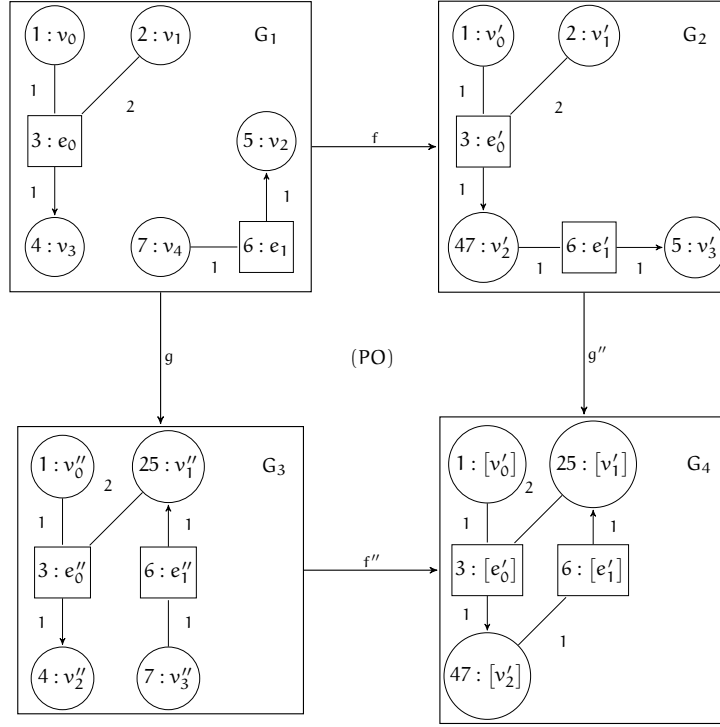
- (V_D, f_V, g_V) is a pushout in **Sets** of (b_V, c_V) with $V_D = (V_B \uplus V_C)|_{\equiv_V}$, $\forall v \in V_C$. $f_V(v) = [v]_{\equiv_V}$, and $\forall v \in V_B$. $g_V(v) = [v]_{\equiv_V}$ where \equiv_V is the smallest equivalence relation with $\forall n \in V_A$. $(b_V(n), c_V(n)) \in \equiv_V$.
- (E_D, f_E, g_E) is a pushout in **Sets** of (b_E, c_E) with $E_D = (E_B \uplus E_C)|_{\equiv_E}$, $\forall e \in E_C$. $f_E(e) = [e]_{\equiv_E}$, and $\forall e \in E_B$. $g_E(e) = [e]_{\equiv_E}$ where \equiv_E is the smallest equivalence relation with $\forall m \in E_A$. $(b_E(m), c_E(m)) \in \equiv_E$.
- $\forall e \in E_D$. $s_D(e) = \begin{cases} f_V^*(s_C(e')) & \text{if } e = [e'] \wedge e' \in E_C, \\ g_V^*(s_B(e')) & \text{if } e = [e'] \wedge e' \in E_B \end{cases}$
- $\forall e \in E_D$. $t_D(e) = \begin{cases} f_V^*(t_C(e')) & \text{if } e = [e'] \wedge e' \in E_C, \\ g_V^*(t_B(e')) & \text{if } e = [e'] \wedge e' \in E_B \end{cases}$



Lemma 3 (Pushout in HyperGraphs).

Consider the pushout construction from Definition 36. Then the diagram given below is a hypergraph pushout.

$$\begin{array}{ccc} A & \xrightarrow{b} & B \\ c \downarrow & (1) & \downarrow g \\ C & \xrightarrow{f} & D \end{array}$$

Figure 8: Pushout in **HyperGraphs****Proof.**

The detailed proof of this lemma is given in [Appendix C](#) on [page 312](#). \square

An example for the concrete pushout construction is given below, where the graphical representation of the constructed pushout object G_4 is given in [Figure 8](#).

Example 4 (Construction of Hypergraph Pushouts).

Consider hypergraphs G_1 , G_2 , G_3 and hypergraph morphisms $f : G_1 \rightarrow G_2$, $g : G_1 \rightarrow G_3$ as given in [Figure 8](#). We construct the hypergraph $G_4 = (V_{G_4}, E_{G_4}, s_{G_4}, t_{G_4})$ together with the hypergraph morphisms $f'' : G_3 \rightarrow G_4$ with $f'' = (f''_V : V_{G_3} \rightarrow V_{G_4}, f''_E : E_{G_3} \rightarrow E_{G_4})$ and $g'' : G_2 \rightarrow G_4$ with $g'' = (g''_V : V_{G_2} \rightarrow V_{G_4}, g''_E : E_{G_2} \rightarrow E_{G_4})$ according to [Definition 36](#).

- $V_{G_4} = (V_{G_2} \uplus V_{G_3})|_{\equiv_V} = \{[v'_0], [v'_1], [v'_2]\}$ where \equiv_V is the smallest equivalence relation with $\forall v \in V_{G_1}. (f_V(v), g_V(v)) \in \equiv_V$. The corresponding morphisms for the node-components are given by $f''_V = \{(v''_0, [v'_0]), (v'_1, [v'_1]), (v''_2, [v'_2]), (v'_3, [v'_1])\}$ and $g''_V = \{(v'_0, [v'_0]), (v'_1, [v'_1]), (v'_2, [v'_2]), (v'_3, [v'_1])\}$.
- $E_{G_4} = (E_{G_2} \uplus E_{G_3})|_{\equiv_E} = \{[e'_0], [e'_1]\}$ where \equiv_E is the smallest equivalence relation with $\forall e \in E_{G_1}. (f_E(e), g_E(e)) \in \equiv_E$. The corresponding morphisms for the edge-components are given by $f''_E = \{(e''_0, [e'_0]), (e'_1, [e'_1])\}$ and $g''_E = \{(e'_0, [e'_0]), (e'_1, [e'_1])\}$.
- $s_{G_4}([e'_0]) = g''^*_V(s_{G_2}(e'_0)) = g''^*_V(v'_0 \cdot v'_1) = [v'_0] \cdot [v'_1]$
- $s_{G_4}([e'_1]) = g''^*_V(s_{G_2}(e'_1)) = g''^*_V(v'_2) = [v'_2]$
- $t_{G_4}([e'_0]) = g''^*_V(t_{G_2}(e'_0)) = g''^*_V(v'_2) = [v'_2]$
- $t_{G_4}([e'_1]) = g''^*_V(t_{G_2}(e'_1)) = g''^*_V(v'_3) = [v'_1]$

In the next step, we recall the definition of hypergraph pullbacks given in [\[169\]](#). The subsequent [Lemma 4](#) states that the construction described in [Definition 37](#) results in a valid hypergraph pullback.

Definition 37 (Construction of Hypergraph Pullbacks [169]).

Let $g : B \rightarrow D$ and $f : C \rightarrow D$ be hypergraph morphisms. Define $A = (V_A, E_A, s_A, t_A)$ from the diagram below as follows:

$$\begin{array}{ccc} A & \xrightarrow{b} & B \\ c \downarrow & \text{(PB)} & \downarrow g \\ C & \xrightarrow{f} & D \end{array}$$

- (V_A, b_V, c_V) is a pullback in **Sets** of (f_V, g_V) with $V_A = \{(v_1, v_2) \in V_B \times V_C \mid g_V(v_1) = f_V(v_2)\}$, $\forall (v_1, v_2) \in V_A$. $b_V(v_1, v_2) = v_1$ and $\forall (v_1, v_2) \in V_A$. $c_V(v_1, v_2) = v_2$.
- (E_A, b_E, c_E) is a pullback in **Sets** of (f_E, g_E) with $E_A = \{(e_1, e_2) \in E_B \times E_C \mid g_E(e_1) = f_E(e_2)\}$, $\forall (e_1, e_2) \in E_A$. $b_E(e_1, e_2) = e_1$ and $\forall (e_1, e_2) \in E_A$. $c_E(e_1, e_2) = e_2$.
- $\forall (e_1, e_2) \in E_A$. $s_A(e_1, e_2) = (v_1, v'_1) \dots (v_n, v'_n)$ for $s_B(e_1) = v_1 \dots v_n$, $s_C(e_2) = v'_1 \dots v'_n$, and $n \in \mathbb{N}$.
- $\forall (e_1, e_2) \in E_A$. $t_A(e_1, e_2) = (v_1, v'_1) \dots (v_n, v'_n)$ for $t_B(e_1) = v_1 \dots v_n$, $t_C(e_2) = v'_1 \dots v'_n$, and $n \in \mathbb{N}$.

$$\begin{array}{ccccc} & & b_E & & b_V^* \\ & \nearrow & & \searrow & \\ E_A & \xrightarrow{s_A} & V_A^* & & E_B & \xrightarrow{s_B} & V_B^* \\ & \searrow & & \nearrow & \\ & & c_V^* & & g_E & & g_V^* \\ c_E \downarrow & & & & \downarrow & & \downarrow \\ E_C & \xrightarrow{s_C} & V_C^* & & E_D & \xrightarrow{s_D} & V_D^* \\ & \nearrow & & \searrow & \\ & & f_E & & f_V^* \end{array}$$

Lemma 4 (Pullback in HyperGraphs).

Consider the pullback construction from Definition 37. Then the diagram given below is a hypergraph pullback.

$$\begin{array}{ccc} A & \xrightarrow{b} & B \\ c \downarrow & (1) & \downarrow g \\ C & \xrightarrow{f} & D \end{array}$$

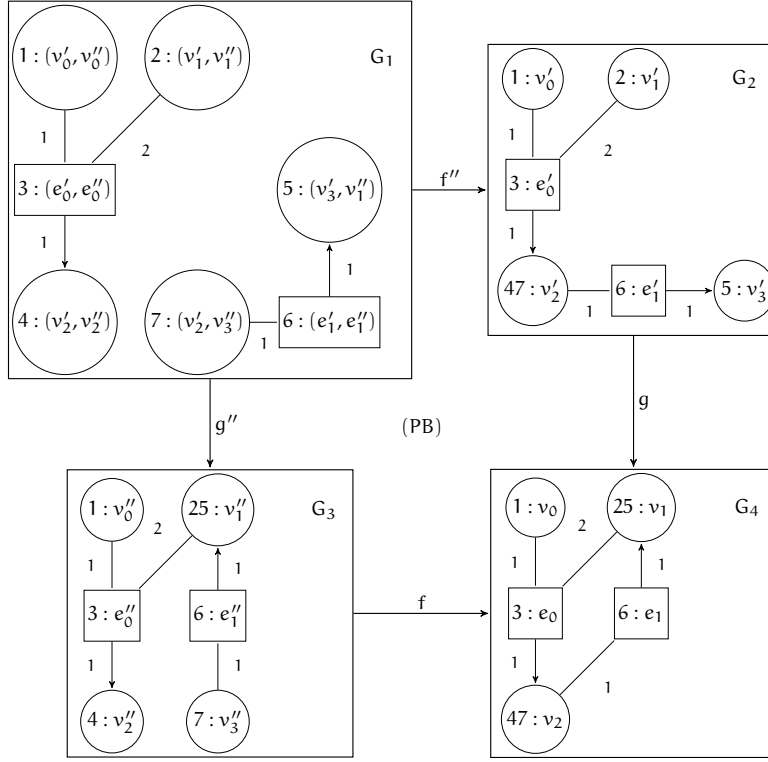
Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 314](#). □

In the following example, we construct a pullback in the category **HyperGraphs** according to Definition 37. The graphical representation of the constructed pullback object G_1 is given in [Figure 9](#).

Example 5 (Construction of Hypergraph Pullbacks).

Consider hypergraphs G_2, G_3, G_4 and hypergraph morphisms $f : G_3 \rightarrow G_4, g : G_2 \rightarrow G_4$ as given in [Figure 9](#). We construct the hypergraph $G_1 = (V_{G_1}, E_{G_1}, s_{G_1}, t_{G_1})$ together with the hypergraph morphisms $f'' : G_1 \rightarrow G_2$ with $f'' = (f_V'' : V_{G_1} \rightarrow V_{G_2}, f_E'' : E_{G_1} \rightarrow E_{G_2})$ and $g'' : G_1 \rightarrow G_3$ with $g'' = (g_V'' : V_{G_1} \rightarrow V_{G_3}, g_E'' : E_{G_1} \rightarrow E_{G_3})$ according to Definition 37.

Figure 9: Pullback in **HyperGraphs**

- $V_{G_1} = \{(v_1, v_2) \in V_{G_2} \times V_{G_3} \mid g_V(v_1) = f_V(v_2)\} = \{(v'_0, v''_0), (v'_2, v''_2), (v'_2, v''_3), (v'_1, v''_1), (v'_3, v''_3)\}$. The corresponding morphisms for the node-components are given by $f''_V = \{((v'_0, v''_0), v'_0), ((v'_2, v''_2), v'_2), ((v'_2, v''_3), v'_2), ((v'_1, v''_1), v'_1), ((v'_3, v''_3), v'_3)\}$ and $g''_V = \{((v'_0, v''_0), v''_0), ((v'_2, v''_2), v''_2), ((v'_2, v''_3), v''_3), ((v'_1, v''_1), v''_1), ((v'_3, v''_3), v''_3)\}$.
- $E_{G_1} = \{(e_1, e_2) \in E_{G_2} \times E_{G_3} \mid g_E(e_1) = f_E(e_2)\} = \{(e'_0, e''_0), (e'_1, e''_1)\}$. The corresponding morphisms for the edge-components are given by $f''_E = \{((e'_0, e''_0), e'_0), ((e'_1, e''_1), e'_1)\}$ and $g''_E = \{((e'_0, e''_0), e''_0), ((e'_1, e''_1), e''_1)\}$.
- $s_{G_1}(e'_0, e''_0) = (v'_0, v''_0) \cdot (v'_1, v''_1)$ for $s_{G_2}(e'_0) = v'_0 \cdot v'_1$ and $s_{G_3}(e''_0) = v''_0 \cdot v''_1$
- $s_{G_1}(e'_1, e''_1) = (v'_2, v''_3)$ for $s_{G_2}(e'_1) = v'_2$ and $s_{G_3}(e''_1) = v''_3$
- $t_{G_1}(e'_0, e''_0) = (v'_2, v''_2)$ for $t_{G_2}(e'_0) = v'_2$ and $t_{G_3}(e''_0) = v''_2$
- $t_{G_1}(e'_1, e''_1) = (v'_3, v''_3)$ for $t_{G_2}(e'_1) = v'_3$ and $t_{G_3}(e''_1) = v''_3$

For rule-based transformation of hypergraphs, we use the classical DPO approach with hypergraph transformation rules defined as spans of injective hypergraph morphisms [253]. To ensure the rule applicability for hypergraph transformations, we use again the notion of initial pushouts. Similar to initial pushouts in the category of graphs (see the end of Subsection 2.4.1), we also have initial pushouts in the \mathcal{M} -adhesive category (**HyperGraphs**, \mathcal{M}) using the boundary construction based on dangling and identification points (see Lemma 7). The detailed constructions of the boundary and the context objects over a general morphism $f : L \rightarrow G$ for the \mathcal{M} -adhesive category (**HyperGraphs**, \mathcal{M}) are given in Lemmas 5 and 6 below.

Lemma 5 (Boundary Object in (HyperGraphs**, \mathcal{M})).**

Consider two hypergraphs $L = (V_L, E_L, s_L, t_L)$ and $G = (V_G, E_G, s_G, t_G)$. The boundary object

$B \subseteq L$ of the initial pushout over a general morphism $f : L \rightarrow G$ in the \mathcal{M} -adhesive category $(\mathbf{HyperGraphs}, \mathcal{M})$ can be constructed as follows with an inclusion $b : B \rightarrow L$:

$B = (V_B, E_B, s_B, t_B)$ where

$V_B = DP_V \cup IP_V \cup IP_{VE}$ with dangling points

$$DP_V = \{v \in V_L \mid \exists e \in E_G \setminus f_E(E_L). (f_V(v) \notin s_G(e)) \vee (f_V(v) \notin t_G(e))\}$$

where $x \in w \Leftrightarrow \exists w_1, w_2. w = w_1 \cdot x \cdot w_2$ and identification points

$$IP_V = \{v \in V_L \mid \exists v' \neq v. v' \in V_L \wedge f_V(v) = f_V(v')\},$$

$$IP_{VE} = \{\bar{v} \in V_L \mid \exists e \in IP_E. \bar{v} \in s_L(e) \vee \bar{v} \in t_L(e)\}^{25},$$

$$E_B = IP_E = \{e \in E_L \mid \exists e' \neq e. e' \in E_L \wedge f_E(e) = f_E(e')\},$$

$$s_B(e) = s_L(e),$$

$$t_B(e) = t_L(e).$$

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ & & \downarrow f \\ & & G \end{array}$$

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 316](#). □

Lemma 6 (Context Object in $(\mathbf{HyperGraphs}, \mathcal{M})$).

Consider a hypergraph morphism $f : L \rightarrow G$ and the boundary object B constructed according to Lemma 5 above. Then the context object C can be constructed in the \mathcal{M} -adhesive category $(\mathbf{HyperGraphs}, \mathcal{M})$ as follows with inclusion $c : C \rightarrow G$:

$C = (V_C, E_C, s_C, t_C)$ with

$$V_C = (V_G \setminus f_V(V_L)) \cup f_V(b_V(V_B)),$$

$$E_C = (E_G \setminus f_E(E_L)) \cup f_E(b_E(E_B)),$$

$$s_C(e) = s_G(e),$$

$$t_C(e) = t_G(e).$$

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ \downarrow & & \downarrow f \\ C & \xrightarrow{c} & G \end{array}$$

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 316](#). □

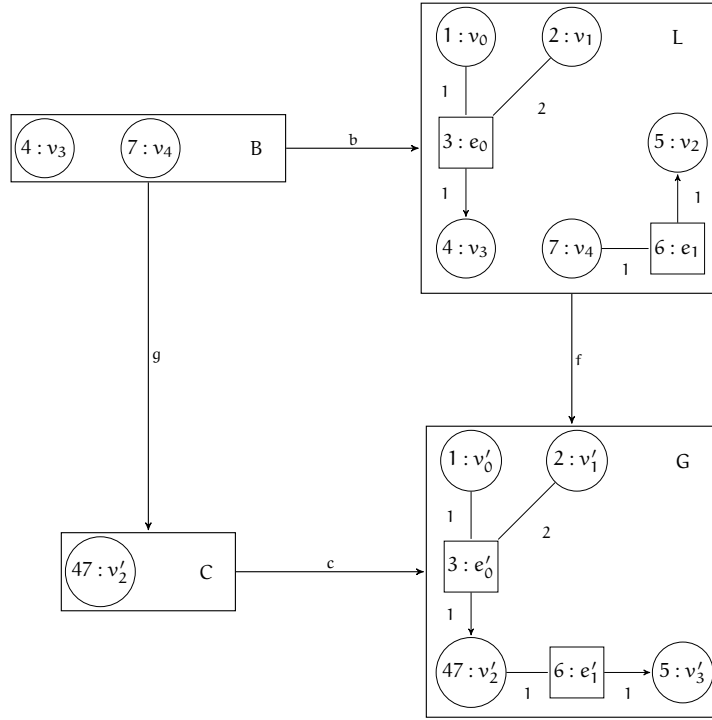
Lemma 7 (Initial Pushout in $(\mathbf{HyperGraphs}, \mathcal{M})$).

Consider a hypergraph morphism $f : L \rightarrow G$, a boundary object B constructed according to Lemma 5, a context object C constructed according to Lemma 6, and inclusions $b : B \rightarrow L$, $c : C \rightarrow G$. Then the diagram (1) given below is an initial pushout in $(\mathbf{HyperGraphs}, \mathcal{M})$ with the hypergraph morphism $g : B \rightarrow C$ defined as $g = f|_B$ ²⁶.

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ g \downarrow & (1) & \downarrow f \\ C & \xrightarrow{c} & G \end{array}$$

²⁵ Note that IP_{VE} is needed in order to make sure that the boundary object B is a well-defined hypergraph to avoid an intersection construction for B .

²⁶ Since b and c are inclusions, the morphism g can be constructed as the restriction of f to the domain of g .

Figure 10: Initial Pushout in $(\mathbf{HyperGraphs}, \mathcal{M})$ **Proof.**

The detailed proof of this lemma is given in [Appendix C](#) on [page 318](#). \square

In the example below, we construct an initial pushout in the \mathcal{M} -adhesive category $(\mathbf{HyperGraphs}, \mathcal{M})$ according to Lemmas 5, 6, and 7. Its graphical representation is given in [Figure 10](#).

Example 6 (Construction of Initial Pushouts in $(\mathbf{HyperGraphs}, \mathcal{M})$).

Consider hypergraphs L , G , and a hypergraph morphism $f : L \rightarrow G$ as given in [Figure 10](#). We construct the boundary object $B = (V_B, E_B, s_B, t_B)$ with the hypergraph morphism $b : B \rightarrow L$ according to Lemma 5, the context object $C = (V_C, E_C, s_C, t_C)$ with the hypergraph morphism $c : C \rightarrow G$ according to Lemma 6 and the hypergraph morphism $g : B \rightarrow C$ according to Lemma 7 as follows:

- $B = (V_B, E_B, s_B, t_B)$ with

$$V_B = \text{DP}_V \cup \text{IP}_V \cup \text{IP}_{VE} = \emptyset \cup \{v_3, v_4\} \cup \emptyset = \{v_3, v_4\},$$

$$E_B = \text{IP}_E = \emptyset,$$

$$s_B = t_B = \emptyset, \text{ and}$$

$$b = (b_V : V_B \rightarrow V_L, b_E : E_B \rightarrow E_L) = (\{(v_3, v_3), (v_4, v_4)\}, \emptyset).$$

- $C = (V_C, E_C, s_C, t_C)$ with

$$V_C = (\{v'_0, v'_1, v'_2, v'_3\} \setminus f_V(\{v_0, v_1, v_2, v_3, v_4\})) \cup f_V(b_V(\{v_3, v_4\})) = \{v'_2\}$$

$$E_C = (\{e'_0, e'_1\} \setminus f_E(\{e_0, e_1\})) \cup f_E(b_E(\emptyset)) = \emptyset$$

$$s_C = t_C = \emptyset, \text{ and}$$

$$c = (c_V : V_C \rightarrow V_G, c_E : E_C \rightarrow E_G) = (\{(v'_2, v'_2)\}, \emptyset).$$

- $g = (g_V : V_B \rightarrow V_C, g_E : E_B \rightarrow E_C) = f|_B = (\{(v_3, v'_2), (v_4, v'_2)\}, \emptyset)$

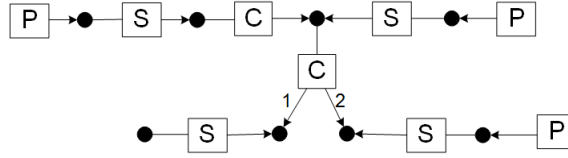


Figure 11: Hypergraph defining a network with distributed processes

We now introduce our running example, which we want to analyze later on for the case of hypergraph transformation systems without or with nested application conditions by applying the theoretical results of our approach. Note that for this example, we use hypergraphs extended by a labeling function for hyperedges. Objects in this slightly extended category have the form: $G = (V_G, E_G, s_G, t_G, l_G)$ with the labeling function $l_G : E_G \rightarrow A$ where A is some alphabet.

Example 7 (Mobile Processes [211, 213]).

We consider a simple distributed system with mobility, inspired by [12], with servers connected by channels, and processes moving through the network and running on the servers. In our system model, servers, connections and processes are represented as labeled hyperedges. The meaning of the hyperedge labels is as follows: P denotes a process before it is executed, S stands for server, and C for connection. A running process is represented by label R . Note that, on the one hand, we simplify the network model in [12] by disregarding firewalls and secure servers; on the other hand, we allow for connections between three servers modeled by hyperedges with three tentacles, and we distinguish between traveling processes P and running processes R .

The hypergraph in Figure 11 models a network with four servers, different kinds of connections between them, and three processes. A process P is located at a server S when the process hyperedge is connected to the source node of the server hyperedge.

The behavior of the system is modeled by the hypergraph transformation rules in Figure 12 for the case if we consider this system without any additional constraints expressed by nested application conditions. Rules `enterServer` [`leaveServer`] allow a process to enter [leave] a server location. Both rules are inverse to each other (indicated by the double arrow). Rules `crossC` [`backC`] model the traveling of a process via a connection. We have different rules for process traveling, depending on the kind of connection hyperedge that is crossed. When a process finally has found a suitable server, it switches into the **running** state by applying the rule `runP`. A process that has finished its execution, is removed from the system by the rule `removeR`.

To give later an example for local confluence analysis of hypergraph transformation systems containing rules with nested application conditions, we extend the rules `enterServer`, `runP` and `removeR` by nested application conditions. The changed rules are given in Figure 13. In the extended scenario, we assume for the sake of the example that a server can locate at most two processes, which is modeled as a negative application condition (NAC) for the rule `enterServer`. Furthermore, rules `runP` and `removeR` require by their positive application conditions (PACs) that the process is located at a server.

After our running example for a hypergraph transformation system is introduced, we give an additional example for understanding the strict AC-confluence of a critical pair in the context of the Mobile Processes scenario.

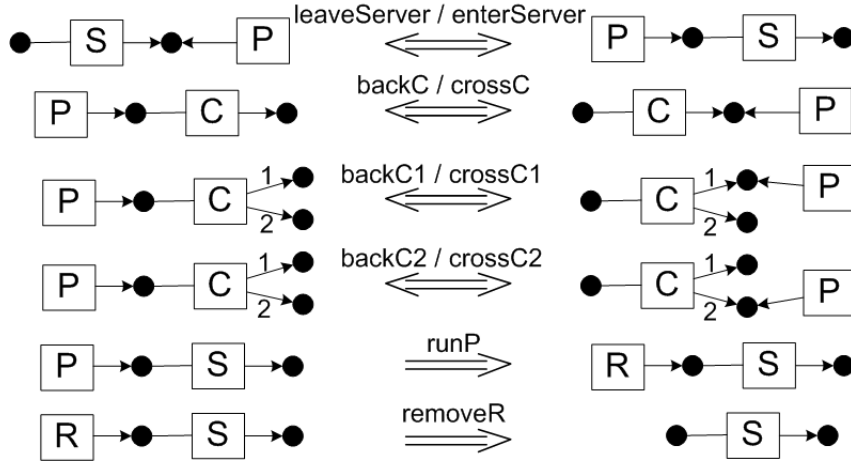


Figure 12: Hypergraph transformation rules without nested application conditions modeling the behavior of the Mobile Processes scenario

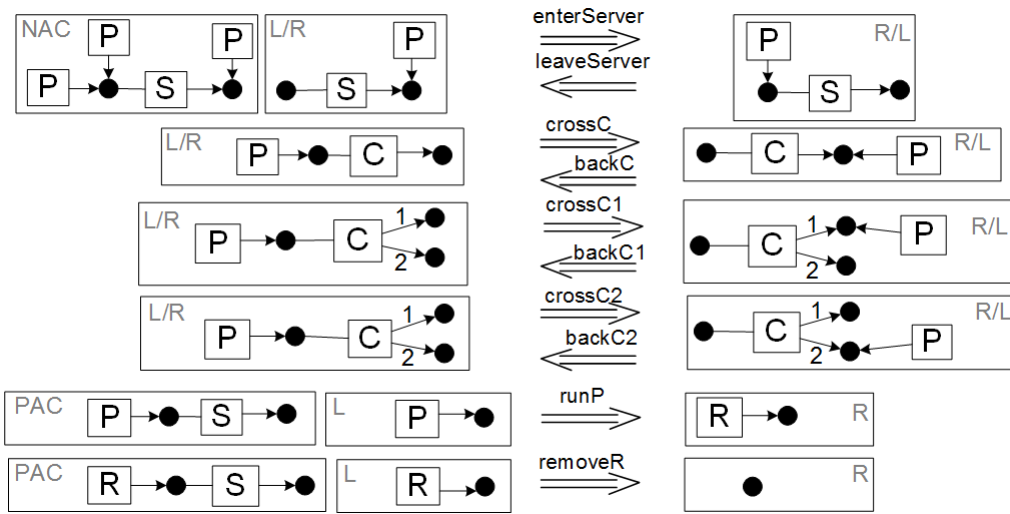


Figure 13: Hypergraph transformation rules with nested application conditions modeling the behavior of the extended Mobile Processes scenario

Example 8 (Strict AC-Confluence of a Critical Pair in the Context of the Mobile Processes Scenario [213]).

In this example, we construct a critical pair of rules *runP* and *enterServer* (see Figure 13) and show that this critical pair is strictly AC-confluent according to Definition 29. We choose this critical pair as an example for showing strict AC-confluence since it represents the most interesting case combining rules with a NAC and a PAC.²⁷

We proceed as follows. First, we construct a weak critical pair of both rules, i.e., a pair of AC-disregarding transformations with jointly surjective hypergraph morphisms α_1 and α_2 as given in Figure 14. Note that this weak critical pair satisfies the given NAC but does not satisfy the PAC.

²⁷ In Section 7.3, where we later describe the AGG-based local confluence analysis of our concrete hypergraph transformation system with NACs and PACs, we can exclude all \mathcal{F}_{HG} -reachable critical pairs of the rules $\mathcal{F}_{\text{HG}}(\text{runP})$ and $\mathcal{F}_{\text{HG}}(\text{enterServer})$ from the detailed strict $\text{AC}(\mathcal{F}_{\text{HG}})$ -confluence analysis, because the corresponding overlapping graphs cannot be reached from the translated start hypergraph, which means that these \mathcal{F}_{HG} -reachable critical pairs are incompatible with some invariant of the original hypergraph transformation system.

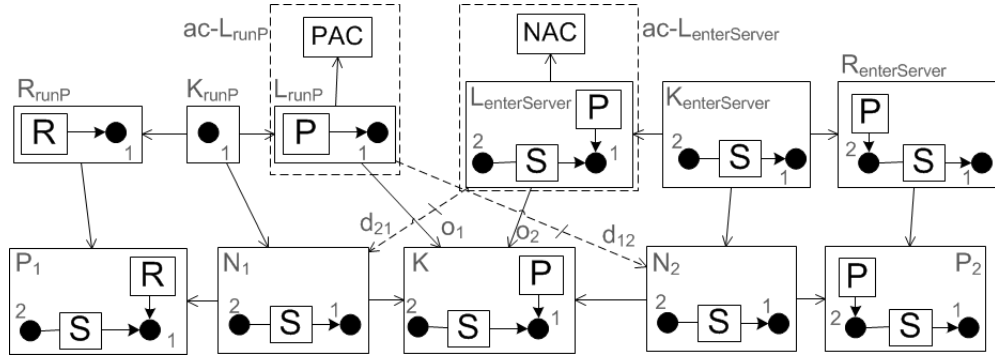
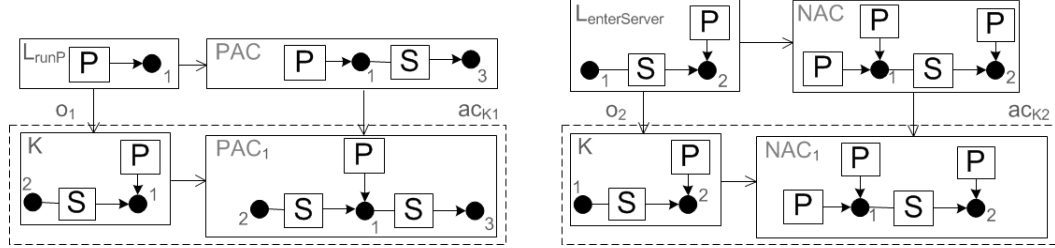


Figure 14: Weak critical pair of rules runP and enterServer

Figure 15: Extension application condition $ac_K = ac_{K_1} \wedge ac_{K_2}$ induced by the weak critical pair of rules runP and enterServer

The weak critical pair in Figure 14 induces the following extension and conflict-inducing application conditions ac_K and ac_K^* on K according to Definition 25:

- $ac_K = \text{Shift}(o_1, ac_{L_{runP}}) \wedge \text{Shift}(o_2, ac_{L_{enterServer}}) = ac_{K_1} \wedge ac_{K_2}$ and
- $ac_K^* = \neg(ac_{K,d_{21}}^* \wedge ac_{K,d_{12}}^*) = \neg(false \wedge false) = true$ ²⁸

The extension application condition ac_K consists of the application conditions ac_{K_1} and ac_{K_2} marked by dashed rectangles in Figure 15, which we obtain by shifting of $ac_{L_{runP}}$ and $ac_{L_{enterServer}}$ (see parts marked by dashed rectangles in Figure 14) along morphisms o_1 and o_2 to object K . The conflict-inducing application condition ac_K^* is true, because there are no morphisms $d_{12} : L_{runP} \rightarrow N_2$ and $d_{21} : L_{enterServer} \rightarrow N_1$ (depicted by scratched dashed arrows in Figure 14) making the corresponding triangles commute.

The considered weak critical pair is a critical pair, because it can be embedded into a pair of AC-regarding transformations $H_1 \xrightarrow{runP, m_1} G \xrightarrow{enterServer, m_2} H_2$ (see Figure 16) such that the embedding morphism $m : K \rightarrow G$ satisfies $ac_K \wedge ac_K^*$ and the match morphisms $m_1 : L_{runP} \rightarrow G$, $m_2 : L_{enterServer} \rightarrow G$ make the corresponding triangles between L_{runP} , K , and G resp. between $L_{enterServer}$, K , and G commute.

Figure 17 shows that the embedding morphism $m : K \rightarrow G$ satisfies the application conditions ac_{K_1} , ac_{K_2} and hence also ac_K , because there is an injective hypergraph morphism $q_1 : PAC_1 \rightarrow G$ making the triangle (1) commute and there is no injective hypergraph morphism $q_2 : NAC_1 \rightarrow G$ (depicted by the scratched dashed arrow) making the triangle (2) commute. Moreover, $m \models ac_K^*$ since every morphism satisfies true. Thus, we obtain that $P_1 \xrightarrow{runP, o_1} K \xrightarrow{enterServer, o_2} P_2$ is a critical pair.

²⁸ $ac_K^* = true$, because the morphisms d_{12} and d_{21} do not exist. Consideration of inverse rules will lead to the existence of d_{12} and d_{21} .

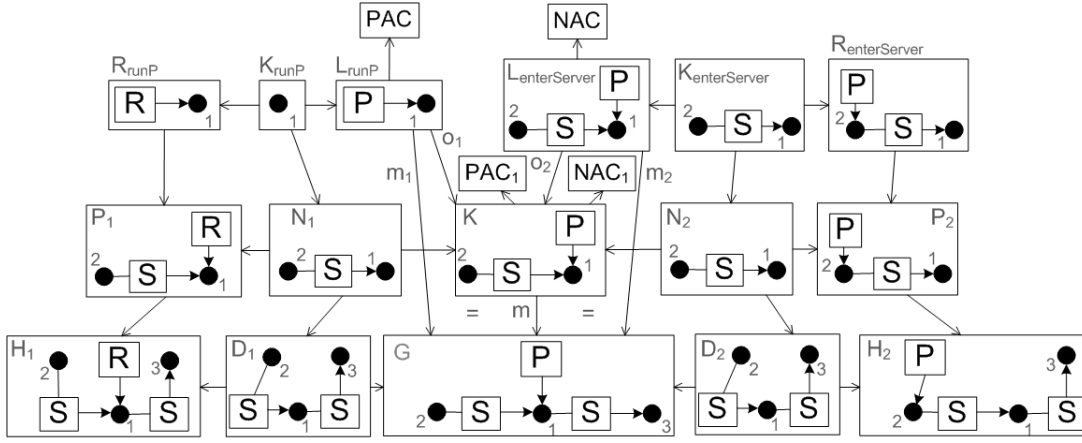
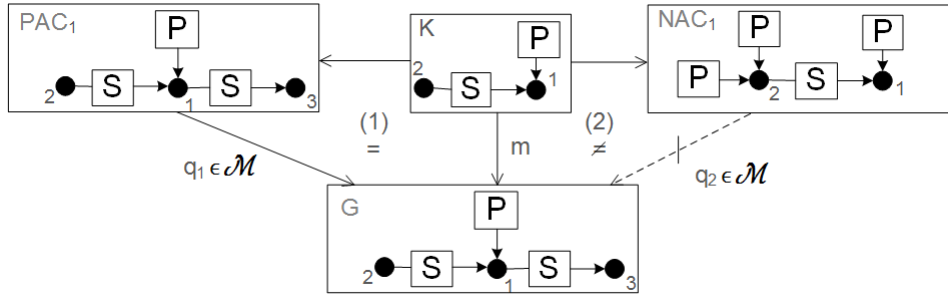
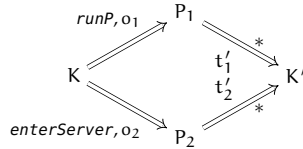


Figure 16: Critical pair of rules runP and enterServer

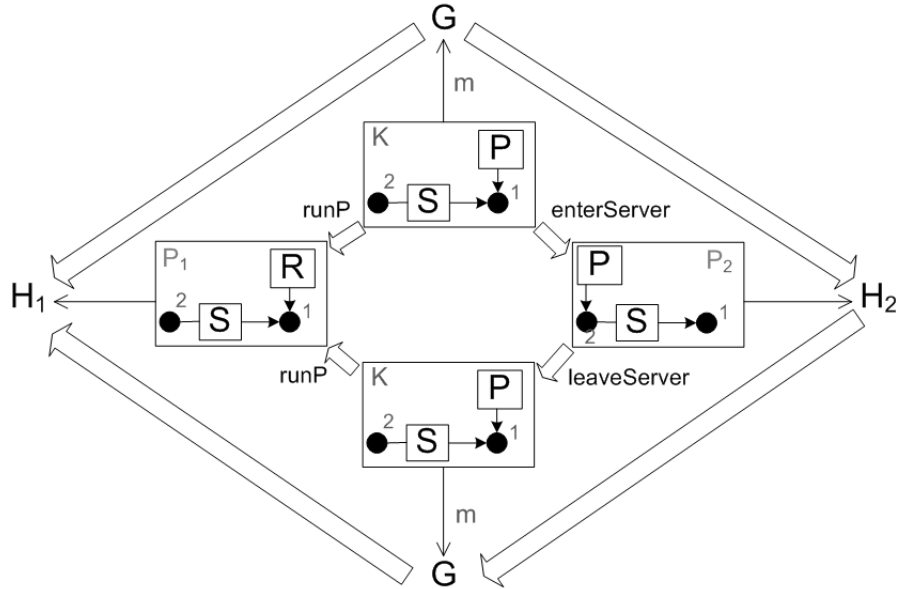
Figure 17: Embedding morphism $m : K \rightarrow G$ satisfies ac_K induced by the weak critical pair of rules runP and enterServer

As the next step we have to check whether the critical pair constructed before is strictly AC-confluent, i.e., the merging transformation steps t'_1, t'_2 are strictly confluent for the case if we disregard application conditions as well as the extended AC-disregarding transformations $t_1 \triangleq K \xrightarrow{runP, o_1} P_1 \xrightarrow{t'_1 *} K', t_2 \triangleq K \xrightarrow{enterServer, o_2} P_2 \xrightarrow{t'_2 *} K'$ are AC-compatible.



In Figure 18²⁹, we can see that the transformation steps $K \xrightarrow{runP, o_1} P_1$ and $K \xrightarrow{enterServer, o_2} P_2$ can be joined together to the hypergraph P_1 . This is possible, because after applying the rule enterServer, we can reverse the effect by applying the corresponding inverse rule leaveServer getting again the hypergraph K ; applying then the rule runP, we obtain a hypergraph, which is isomorphic to the hypergraph P_1 . Additionally, the maximal substructure of K (consisting of the two nodes 1 and 2 with a server between them), that is preserved by the critical pair, is also preserved by the merging steps described before. Therefore, the strictness condition holds as well.

²⁹ Note that the inner square in Figure 18 including the critical pair is AC-disregarding, but the outer square with G, H_1 , and H_2 is AC-regarding.

Figure 18: Plain strict confluence of the critical pair of rules `runP` and `enterServer`

Now it remains to show that the extended transformations t_1 and t_2 are AC-compatible, i.e., $(ac_K \wedge ac_K^*) \Rightarrow (ac(t_1) \wedge ac(t_2))$. For this reason, we calculate the derived application conditions $ac(t_1)$ and $ac(t_2)$ obtaining the following:

$$\begin{aligned}
 ac(t_1) &= ac(K \xrightarrow{runP, o_1} P_1) = ac_{K_1}, \\
 ac(t_2) &= ac(K \xrightarrow{enterServer, o_2} P_2 \xrightarrow{leaveServer, o_3} K \xrightarrow{runP, o_1} P_1) = ac_{K_2} \wedge true \wedge ac_{K_1}, \\
 &\Rightarrow ac(t_1) \wedge ac(t_2) = ac_{K_1} \wedge ac_{K_2}
 \end{aligned}$$

Since we already know that $ac_K \wedge ac_K^* = ac_{K_1} \wedge ac_{K_2}$, we obtain the AC-compatibility for t_1 and t_2 , because $(ac_{K_1} \wedge ac_{K_2}) \Rightarrow (ac_{K_1} \wedge ac_{K_2})$. Thus, we have that the constructed critical pair $P_1 \xleftarrow{runP, o_1} K \xrightarrow{enterServer, o_2} P_2$ is strictly AC-confluent.

2.4.3 PTI Net Transformation Systems

Petri nets, initially introduced in 1962 by Carl Adam Petri [252], are a formal specification language useful for modeling concurrent, distributed, parallel, asynchronous, non-deterministic and stochastic systems [268] supporting powerful techniques for qualitative and quantitative analysis [232]. The formalism of Petri nets combines a well-founded mathematical theory with a graphical representation of the dynamic system behavior. A strength of the Petri net approach is its intuitive visual notation and the variety of implemented analysis techniques considering, e.g., reachability, liveness, and safeness properties. According to [308], there are many examples for application of different kinds of Petri nets to model a variety of dynamic event-driven and concurrent systems such as computer networks [205], communication systems [214, 307], real-time computing systems [294], workflows [299, 195], and logistic networks [301].

The formalism of Reconfigurable Petri Nets [154, 260] generalized to the notion of Petri net transformation systems [90, 92, 80, 81, 246] allows to combine formal modeling of dynamic systems and controlled model adaptation. The main idea is the stepwise development of Place/Transition nets (short P/T nets) by applying net transformation rules [90, 92, 260]. This approach makes Petri nets more expressive and allows additionally to the well-known “token game” for a formal description and analysis of structural changes [90, 92, 260]. The application areas for the Petri net transformation systems formalism comprise for instance medical information systems [103], train control systems [247], logistics [90], emergency scenarios [92], reconfigurable manufacturing systems [164], complex dynamic structures [130] and component technology [243].

We introduce in this subsection Petri nets with individual tokens (short PTI nets) as a slightly different formalization of classical P/T nets. Intuitively, each PTI net is explicitly equipped with a marking included into the structure of the net, which is no longer defined as a sum of a monoid, but as a set of individuals. Then, considering a PTI net as a graph-like structure, the marking (i.e., the tokens assigned to places) is also a subject of transformation rules. We use PTI nets instead of the classical P/T net formalism, because they allow for two kinds of transformations on such nets using the DPO approach, namely, transformations of the net structure as well as marking-changing transformations of the net modeling its firing behavior. Unfortunately, the marking-changing transformations are not possible without restrictions for the reconfigurable Place/Transition systems (short P/T systems)³⁰ [223]. The reason for this problem is that the category $(\mathbf{P\!T\!Sys}, \mathcal{M}_{inj})$ of P/T systems together with the class \mathcal{M}_{inj} of all injective morphisms allowing to increase the number of tokens on corresponding places is not \mathcal{M} -adhesive. The adapted category $(\mathbf{P\!T\!Sys}, \mathcal{M}_{strict})$ of P/T systems together with the class \mathcal{M}_{strict} of all injective morphisms where the number of tokens on the corresponding places stays equal is indeed \mathcal{M} -adhesive, but is inconvenient regarding the usability of the transformation approach. Therefore the usage of PTI net transformation systems strictly increases the expressiveness compared to the classical reconfigurable P/T systems formalism. Moreover, considering tokens as individuals makes the formalism of PTI nets even more similar to typed attributed graphs, which gives a possibility to simulate and analyze Petri net transformations by existing graph transformation tools like AGG [20, 206, 285]. The prac-

³⁰ A reconfigurable P/T system consists of a marked P/T net (also called P/T system) and a set of net transformation rules [153].

ticability of the Petri net transformation approach with individual tokens is shown in [105, 223, 116] introducing case studies on applications to communication spaces and communication platforms.

For a classical P/T net N , we adapt the approach of Meseguer and Montanari [216] using free commutative monoids P^\oplus over P , where $N = (P, T, \text{pre}, \text{post})$ with places P , transitions T , pre- and post-domain functions $\text{pre}, \text{post} : T \rightarrow P^\oplus$, and markings $M \in P^\oplus$. A Petri net with individual tokens [224] consists of a classical P/T net together with a set I of individual tokens and a marking function $m : I \rightarrow P$ assigning each token to the corresponding place in the net. Therefore, two (or more) different individual tokens $x, y \in I$ may be on the same place, i.e., $m(x) = m(y)$, while in the standard “collective token approach” the marking $M \in P^\oplus$ tells us only how many tokens we have on each place, but we are not able to distinguish between two tokens on the same place.

Different notions of individuality have been introduced in [266, 300] where tokens are equipped with additional information influencing the firing of transitions. This additional information is used to distinguish tokens, to store relevant data, or to store information about the history of a token. This concept of high-level tokens with an additional inner structure is orthogonal to the PTI net concept since PTI nets only extend the standard Petri net formalism by explicit inclusion of a marking into the structure of the net. These orthogonal approaches are combined in Algebraic High-Level nets (short AHL nets) where tokens are individual tokens representing data elements from a given algebra.

Note that in the following we are interested primarily in structure changing transformations of PTI nets in the context of our running example. However, our theoretical results concerning the functorial behavior transfer (see Subsection 3.1.1) are also applicable to marking changing transformations of PTI nets considered in detail in [224].

We start this subsection with the introduction of some important definitions and concepts concerning PTI nets, which together with the corresponding morphisms form the category **PTINet**. This category completed by the class \mathcal{M} of all injective PTI net morphisms is an \mathcal{M} -adhesive category, as is already shown in [224]. We use the category $(\mathbf{PTINet}, \mathcal{M})$ in the following chapters as the source category for the construction of the concrete restricted \mathcal{M} -functor \mathcal{F}_{PTI} translating PTI nets into the corresponding typed attributed graphs. Furthermore, we consider in this subsection two kinds of boundary construction for PTI nets (depending on the considered morphism class) leading to an initial pushout in $(\mathbf{PTINet}, \mathcal{M})$ as well as give examples for a PTI net pushout, pullback and initial pushout. Finally, in the end of this subsection, we give an example for a PTI net transformation system, which we will use in the following as a running example for the application of our theoretical results to PTI net transformation systems.

According to [224], PTI nets and morphisms between them are defined as follows.

Definition 38 (PTI Nets and PTI Net Morphisms [224]).

- A Petri net with individual tokens $NI = (P, T, \text{pre}, \text{post}, I, m)$ is given by a classical P/T net $N = (P, T, \text{pre} : T \rightarrow P^\oplus, \text{post} : T \rightarrow P^\oplus)$ where P^\oplus is the free commutative monoid over P , I is a (possibly infinite) set of individual tokens, and $m : I \rightarrow P$ is the marking function assigning to each individual token $x \in I$ the corresponding place $m(x) \in P$.

- A PTI net morphism $f : \text{NI}_1 \rightarrow \text{NI}_2$ is given by a triple of functions $f = (f_P : P_1 \rightarrow P_2, f_T : T_1 \rightarrow T_2, f_I : I_1 \rightarrow I_2)^{31}$ such that the following diagrams commute with pre and post, respectively.

$$\begin{array}{ccc}
 T_1 & \xrightleftharpoons[\text{post}_1]{\text{pre}_1} & P_1^\oplus \\
 f_T \downarrow & = & \downarrow f_P^\oplus \\
 T_2 & \xrightleftharpoons[\text{post}_2]{\text{pre}_2} & P_2^\oplus
 \end{array}
 \qquad
 \begin{array}{ccc}
 I_1 & \xrightarrow{m_1} & P_1 \\
 f_I \downarrow & = & \downarrow f_P \\
 I_2 & \xrightarrow{m_2} & P_2
 \end{array}$$

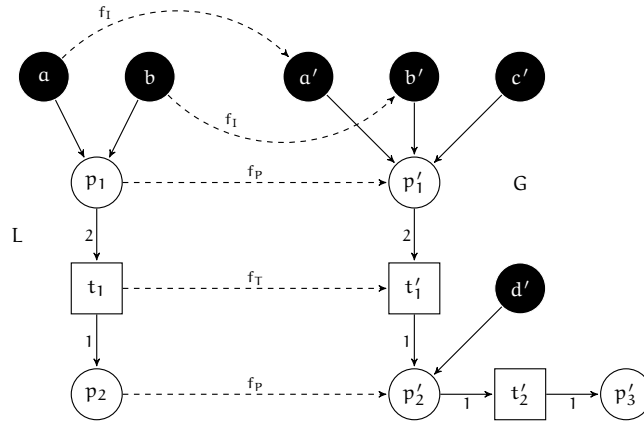
The following example introduces the graphical representation of PTI nets as well as demonstrates a valid morphism between two PTI nets.

Example 9 (PTI Nets and PTI Net Morphisms).

In the picture below to the left, a PTI net L is given with places p_1, p_2 , transition t_1 and individual tokens a, b colored in black. The marking function $m_L : I_L \rightarrow P_L$ assigns both individual tokens to the place p_1 , i.e., $m_L(a) = m_L(b) = p_1$. As usual for P/T nets, the numbers on arrows between places and transitions denote how many individual tokens on places in the pre-domain of some transition are needed to fire this transition, while the numbers between transitions and places determine the number of tokens, which will be produced on the corresponding places in the post-domain of the considered transition after its firing.

Consider another PTI net G given in the picture below to the right. The mapping $f : L \rightarrow G$ as depicted in the picture below with dashed arrows is a valid PTI net morphism, because the compatibility of f with pre- and post-domain functions as well as with the marking function holds:

$$\begin{aligned}
 f_P^\oplus(\text{pre}_L(t_1)) &= f_P^\oplus(2p_1) = 2p'_1 = \text{pre}_G(t'_1) = \text{pre}_G(f_T(t_1)) \\
 f_P^\oplus(\text{post}_L(t_1)) &= f_P^\oplus(p_2) = p'_2 = \text{post}_G(t'_1) = \text{post}_G(f_T(t_1)) \\
 f_P(m_L(a)) &= f_P(p_1) = p'_1 = m_G(a') = m_G(f_I(a)) \\
 f_P(m_L(b)) &= f_P(p_1) = p'_1 = m_G(b') = m_G(f_I(b))
 \end{aligned}$$



³¹ Note, that we do not require for general PTI net morphisms to have an injective f_I component, in order to have pushouts and pullbacks in the category of PTI nets and to be able to show that this category is \mathcal{M} -adhesive. Nevertheless, we use the restriction to the injective f_I component in the following for match and rule morphisms. Furthermore, as given in Remark 8, token-injective PTI net morphisms preserve firing steps of PTI nets, which can be expressed by rule application in the framework of DPO approach.

The firing behavior of PTI nets is defined for marked Petri nets, also called *Petri systems*, as firing steps (see [224]). Since tokens can be distinguished, a possible firing step for a transition have to be considered in the context of a token selection. The token selection gives us explicitly sets of tokens that should be consumed and produced in the firing step.

Definition 39 (Firing Behavior of PTI Nets [224]).

A transition $t \in T$ in a PTI net $NI = (P, T, \text{pre}, \text{post}, I, m)$ is enabled under a token selection (M, m, N, n) where

- $M \subseteq I$ is a set of tokens to be consumed by the firing step,
- $m : I \rightarrow P$ is a marking function of NI ,
- N is a set of tokens to be produced by the firing step such that holds $(I \setminus M) \cap N = \emptyset$,
- $n : N \rightarrow P$ is a token mapping after firing the transition t ,

if it meets the following token selection condition:

$$\left[\sum_{i \in M} m(i) = \text{pre}(t) \right] \wedge \left[\sum_{i \in N} n(i) = \text{post}(t) \right]$$

If an enabled transition t fires, the follower marking (I', m') is given by $I' = (I \setminus M) \cup N$ and $m' : I' \rightarrow P$ with

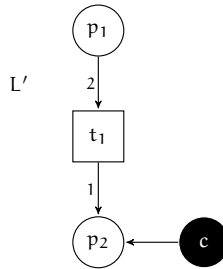
$$m'(x) = \begin{cases} m(x) & \text{if } x \in I \setminus M \\ n(x) & \text{if } x \in N \end{cases}$$

leading to the new PTI net $NI' = (P, T, \text{pre}, \text{post}, I', m')$ in the firing step $NI \xrightarrow{t} NI'$ via the token selection (M, m, N, n) .

An example for firing a transition is given in the following.

Example 10 (Firing of Transitions in PTI Nets).

Consider the PTI net L given in the left picture in Example 9. The transition t_1 of L is obviously enabled under the token selection $(M_L, m_L, N_L, n_L) = (\{a, b\}, \{(a, p_1), (b, p_1)\}, \{c\}, \{(c, p_2)\})$. The follower marking after firing the transition t_1 is $(I'_L, m'_L) = (\{c\}, \{(c, p_2)\})$. Thus, the PTI net $L' = (P_{L'}, T_{L'}, \text{pre}_{L'}, \text{post}_{L'}, I'_L, m'_L)$ given in the picture below is the result of the firing step $L \xrightarrow{t_1} L'$.



PTI net objects and morphisms, as given in Definition 38 above, form the category \mathbf{PTINet} .

Definition 40 (Category PTINet [224]).

All PTI nets as objects and all PTI net morphisms between them define the category of Petri nets with individual tokens **PTINet**.

The following remark summarizes several important results, which hold for the category **PTINet**.

Remark 8 (Relevant Facts about the Category of PTI Nets).

1. It is shown in [224] that token-injective PTI net morphisms preserve firing steps of PTI nets, i.e., for each PTI net morphism $f : NI_1 \rightarrow NI_2$ with injective f_I -component and each firing step $NI_1 \xrightarrow{t} NI'_1$ there is a firing step $NI_2 \xrightarrow{f_T(t)} NI'_2$ and a PTI net morphism $f' : NI'_1 \rightarrow NI'_2$.

$$\begin{array}{ccc} NI_1 & \xrightarrow{t} & NI'_1 \\ f \downarrow & & \downarrow f' \\ NI_2 & \xrightarrow{f_T(t)} & NI'_2 \end{array}$$

2. The behavior described before leads to the fact that firing of transitions can be expressed by rule application.
3. It is shown in [224] that $(\mathbf{PTINet}, \mathcal{M})$ with the class \mathcal{M} of all injective PTI net morphisms is an \mathcal{M} -adhesive category (see Theorem 5.2 there) where pushouts and pullbacks are constructed componentwise³² (see Facts 2.4 and 2.5 there). The diagram in Figure 19 in the end of this subsection is an example for a pushout and pullback in the category **PTINet**.

As introduced in [224], we use the DPO approach for rule-based transformation of PTI nets. PTI net transformation rules are spans of injective PTI net morphisms. Similar to the case of hypergraphs, the applicability of PTI net transformation rules at some match can be characterized by initial pushouts in the \mathcal{M} -adhesive category $(\mathbf{PTINet}, \mathcal{M})$.

In order to construct an initial pushout over a match morphism in $(\mathbf{PTINet}, \mathcal{M})$, we give a definition of a boundary object over a general match morphism $f : L \rightarrow G$ and afterwards adapt this definition to the case of a match morphism from \mathcal{M} . According to [224], the boundary object in $(\mathbf{PTINet}, \mathcal{M})$ is a minimal subnet containing all places, transitions, and individual tokens that must not be deleted by the application of a rule such that there is a pushout complement. The boundary object over a general match morphism $f : L \rightarrow G$ in $(\mathbf{PTINet}, \mathcal{M})$ is given as follows.

Fact 9 (Boundary Object over a General Match Morphism in $(\mathbf{PTINet}, \mathcal{M})$ [224]).

Consider two PTI nets $L = (P_L, T_L, \text{pre}_L, \text{post}_L, I_L, m_L)$ and $G = (P_G, T_G, \text{pre}_G, \text{post}_G, I_G, m_G)$. The boundary object B of an initial pushout over a general match morphism $f : L \rightarrow G$ in the category $(\mathbf{PTINet}, \mathcal{M})$ can be constructed according to [224] as follows:

$$\begin{aligned} B &= (P_B, T_B, \text{pre}_B, \text{post}_B, I_B, m_B) \text{ where} \\ P_B &= DP_T \cup DP_I \cup IP_P \cup IP_I \cup P_{IP_T} \cup P_{IP_I} \\ &\text{with dangling points} \\ DP_T &= \{p \in P_L \mid \exists t \in T_G \setminus f_T(T_L). f_P(p) \in (\bullet t \cup t \bullet)\}, \\ DP_I &= \{p \in P_L \mid \exists x \in I_G \setminus f_I(I_L). f_P(p) = m_G(x)\} \text{ with } m_G : I_G \rightarrow P_G, \end{aligned}$$

³² Note that only pullbacks along injective PTI net morphisms are constructed componentwise in $(\mathbf{PTINet}, \mathcal{M})$. Pullbacks over non-injective morphisms should be constructed according to Fact A.24 in [88].

and identification points

$$IP_P = \{p \in P_L \mid \exists p' \neq p. p' \in P_L \wedge f_P(p) = f_P(p')\},$$

$$IP_I = \{x \in I_L \mid \exists x' \neq x. x' \in I_L \wedge f_I(x) = f_I(x')\},$$

$$P_{IP_T} = \{p \in P_L \mid \exists t \in IP_T. p \in (\bullet t \cup t \bullet)\},$$

$$P_{IP_I} = \{p \in P_L \mid \exists x \in IP_I. p = m_L(x)\} \text{ with } m_L : I_L \rightarrow P_L,$$

$$T_B = IP_T = \{t \in T_L \mid \exists t' \neq t. t' \in T_L \wedge f_T(t) = f_T(t')\},$$

$$pre_B(t) = pre_L(t),$$

$$post_B(t) = post_L(t),$$

$$I_B = IP_I,$$

$$m_B(x) = m_L(x),$$

and $b : B \rightarrow L$ is an inclusion.

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ & & \downarrow f \\ & & G \end{array}$$

For our applications, we only consider the restriction of the category **PTINet** to \mathcal{M} -morphisms, denoted as $\mathbf{PTINet}|_{\mathcal{M}}$ ³³, in order to define in Section 8.2 the restricted \mathcal{M} -functor \mathcal{F}_{PTI} . The mentioned restriction is necessary since \mathcal{F}_{PTI} is not well-defined on general morphisms. But we use the \mathcal{M} -adhesive category $(\mathbf{PTINet}, \mathcal{M})$ in order to have pushouts because the category $(\mathbf{PTINet}|_{\mathcal{M}}, \mathcal{M})$ is not \mathcal{M} -adhesive due to the well-known fact that the induced morphism of \mathcal{M} -morphisms in a pushout diagram is in general not necessarily an \mathcal{M} -morphism.

Considering this restriction, we give a simplified construction of a boundary object in the setting of \mathcal{M} -morphisms. Obviously, the constructions in Fact 9 and Remark 9 coincide for \mathcal{M} -morphisms.

Remark 9 (Boundary Object over an Injective Match Morphism in $(\mathbf{PTINet}, \mathcal{M})$).

According to Fact 9 above, the boundary object B of an initial pushout over an injective match morphism $f : L \rightarrow G$ in the category $(\mathbf{PTINet}, \mathcal{M})$ is given by the corresponding PTI net with empty identification points, i.e., $IP_P = IP_T = IP_I = \emptyset$, which also implies that $P_{IP_T} = P_{IP_I} = \emptyset$ and $T^B = I^B = \emptyset$. This simplifies the construction of the boundary object B in the following way:

$B = (P_B, T_B, pre_B, post_B, I_B, m_B)$ where

$P_B = DP_T \cup DP_I$ with

$$DP_T = \{p \in P_L \mid \exists t \in T_G \setminus f_T(T_L). f_P(p) \in (\bullet t \cup t \bullet)\},$$

$$DP_I = \{p \in P_L \mid \exists x \in I_G \setminus f_I(I_L). f_P(p) = m_G(x)\} \text{ with } m_G : I_G \rightarrow P_G,$$

$$T_B = \emptyset, I_B = \emptyset, pre_B = post_B = m_B = \emptyset,$$

and $b : B \rightarrow L$ is an inclusion.

³³ For the purpose of the restricted \mathcal{M} -functor definition in Part iv, we denote the class of all injective PTI net morphisms by \mathcal{M}_1 instead of \mathcal{M} to emphasize that the category $\mathbf{PTINet}|_{\mathcal{M}_1}$ is the source category of the restricted \mathcal{M} -functor.

After the suitable boundary object is constructed, we now recall in the following fact the general construction of a context object in the category $(\mathbf{PTINet}, \mathcal{M})$ given in [224].

Fact 10 (Context Object in $(\mathbf{PTINet}, \mathcal{M})$ [224]).

Consider a PTI net match morphism $f : L \rightarrow G$ and the boundary object B over f constructed according to Fact 9 above. The context object C is then defined as follows:

$C = (P_C, T_C, \text{pre}_C, \text{post}_C, I_C, m_C)$ where

$$P_C = (P_G \setminus f_P(P_L)) \cup f_P(b_P(P_B)),$$

$$T_C = (T_G \setminus f_T(T_L)) \cup f_T(b_T(T_B)),$$

$$I_C = (I_G \setminus f_I(I_L)) \cup f_I(b_I(I_B)),$$

$$\text{pre}_C(t) = \text{pre}_G(t),$$

$$\text{post}_C(t) = \text{post}_G(t),$$

$$m_C(x) = m_G(x),$$

and $c : C \rightarrow G$ is an inclusion.

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ \downarrow & & \downarrow f \\ C & \xrightarrow{c} & G \end{array}$$

We use in the following also the fact that the diagram constructed over an injective morphism $f : L \rightarrow G$ with boundary and context objects, defined as described before, results in an initial pushout over f , which is obviously a special case of Fact 2.10 from [224] for the case of general morphisms.

Fact 11 (Initial Pushout over an Injective Match Morphism in $(\mathbf{PTINet}, \mathcal{M})$ [224]).

Consider a PTI net morphism $f : L \rightarrow G$ in \mathcal{M} , a boundary object B constructed according to Remark 9, a context object C constructed according to Fact 10, and inclusions $b : B \rightarrow L$, $c : C \rightarrow G$. Then the diagram (1) given below is an initial pushout in $(\mathbf{PTINet}, \mathcal{M})$ with PTI net morphism $g : B \rightarrow C$ defined as $g = f|_B$ ³⁴.

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ g \downarrow & (1) & \downarrow f \\ C & \xrightarrow{c} & G \end{array}$$

In the example below we construct an initial pushout in the category $(\mathbf{PTINet}, \mathcal{M})$ according to Remark 9 and Facts 10, 11. Graphically this initial pushout is given in Figure 19.

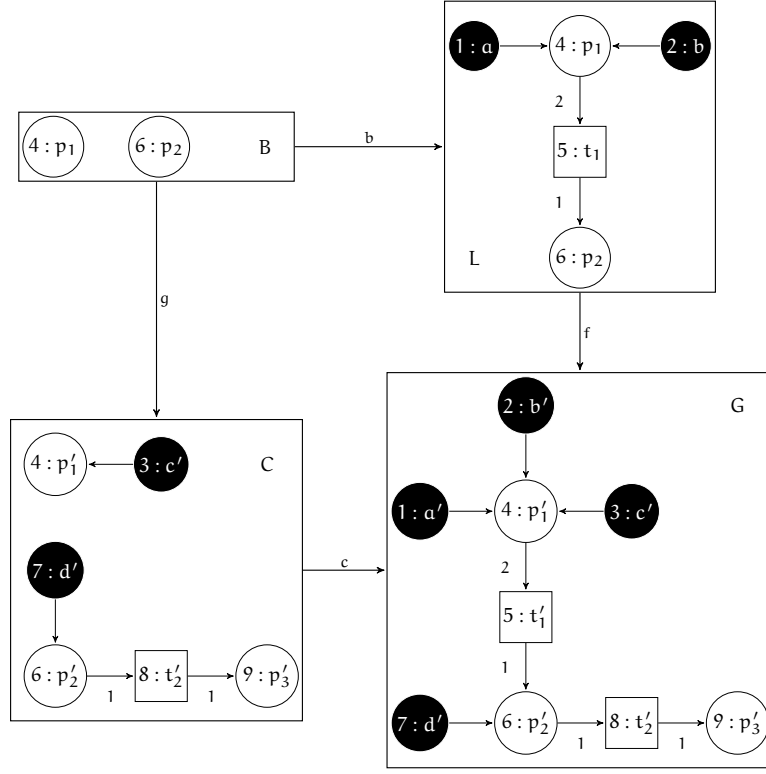
Example 11 (Construction of Initial Pushouts over Injective Match Morphisms in $(\mathbf{PTINet}, \mathcal{M})$).

Consider PTI nets L, G and an injective PTI net morphism $f : L \rightarrow G$ as given in Figure 19. We construct the boundary object $B = (P_B, T_B, \text{pre}_B, \text{post}_B, I_B, m_B)$ with the PTI net morphism $b : B \rightarrow L$ according to Remark 9, the context object $C = (P_C, T_C, \text{pre}_C, \text{post}_C, I_C, m_C)$ with the PTI net morphism $c : C \rightarrow G$ according to Fact 10 and the PTI net morphism $g : B \rightarrow C$ according to Fact 11 as follows:

- $B = (P_B, T_B, \text{pre}_B, \text{post}_B, I_B, m_B)$ with

$$P_B = DP_T \cup DP_I = \{p_2\} \cup \{p_1, p_2\} = \{p_1, p_2\},$$

³⁴ Since b and c are inclusions, the morphism g can be constructed as the restriction of f to the domain of g .

Figure 19: Pushout, pullback, and initial pushout in $(\mathbf{PTINet}, \mathcal{M})$

$$T_B = \emptyset,$$

$$\text{pre}_B = \text{post}_B = \emptyset,$$

$$I_B = \emptyset,$$

$$m_B = \emptyset, \text{ and}$$

$$b = (b_P : P_B \rightarrow P_L, b_T : T_B \rightarrow T_L, b_I : I_B \rightarrow I_L) = (\{(p_1, p_1), (p_2, p_2)\}, \emptyset, \emptyset).$$

- $C = (P_C, T_C, \text{pre}_C, \text{post}_C, I_C, m_C)$ with

$$P_C = (\{p'_1, p'_2, p'_3\} \setminus f_P(\{p_1, p_2\})) \cup f_P(b_P(\{p_1, p_2\})) = \{p'_1, p'_2, p'_3\},$$

$$T_C = (\{t'_1, t'_2\} \setminus f_T(\{t_1\})) \cup f_T(b_T(\emptyset)) = \{t'_2\},$$

$$\text{pre}_C(t'_2) = \text{pre}_G(t'_2) = p'_2,$$

$$\text{post}_C(t'_2) = \text{post}_G(t'_2) = p'_3,$$

$$I_C = (\{a', b', c', d'\} \setminus f_I(\{a, b\})) \cup f_I(b_I(\emptyset)) = \{c', d'\},$$

$$m_C(c') = m_G(c') = p'_1, \quad m_C(d') = m_G(d') = p'_2, \text{ and}$$

$$c = (c_P : P_C \rightarrow P_G, c_T : T_C \rightarrow T_G, c_I : I_C \rightarrow I_G)$$

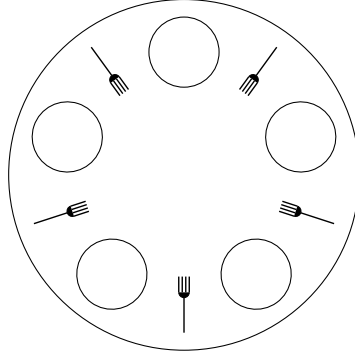
$$= (\{(p'_1, p'_1), (p'_2, p'_2), (p'_3, p'_3)\}, \{(t'_2, t'_2)\}, \{(c', c'), (d', d')\}).$$

- $g = (g_P : P_B \rightarrow P_C, g_T : T_B \rightarrow T_C, g_I : I_B \rightarrow I_C) = f|_B = (\{(p_1, p'_1), (p_2, p'_2)\}, \emptyset, \emptyset)$

We now introduce our running example of a concrete PTI net transformation system, for which we want to analyze the confluence property later on by applying the corresponding theoretical results of our approach.

Example 12 (Mobile Dining Philosophers [207]).

This example is a cutout of the “House of Philosophers” model introduced in [153]. We consider only the restaurant part of the house of philosophers with only one table. Philosophers can take place at the table or leave the table. To introduce a philosopher at the table, the sitting arrangement has to be changed. The activities of philosophers are defined as by the classical “Dining Philosophers” model [58] where five philosophers sit at a round table with plates of spaghetti (see the picture below). Between each two adjacent philosophers there is a fork lying on the table. The philosophers must in turn think and eat spaghetti. To start eating, a philosopher needs two forks to be available, one to the left and one to the right directly next to his plate. Each fork can be used by only one philosopher at the same time. After a philosopher finishes eating spaghetti, he puts both forks down, so that they become available to other philosophers.



In the mentioned cutout, called Mobile Dining Philosophers, we model the dynamic change of the seating arrangement at a table using PTI net transformation rules. The firing of the PTI net transitions models the traditional behavior of the philosophers, switching between thinking and eating. The PTI net G_{init} given in Figure 20 depicts five philosophers sitting at a table and five forks lying between the philosophers on the table. A philosopher is represented by a subnet consisting of places th_i , ea_i and transitions t_i , t'_i with the corresponding edges between them for $i \in \{1, 2, 3, 4, 5\}$. Each philosopher is initially in the thinking state, which is indicated by a token on each th_i place. If two lf_i places in the direct neighborhood of one of the philosophers are marked with a token (e.g. lf_1 and lf_2 for the philosopher 1) and this philosopher is currently in a thinking state, then the philosopher can take both forks and change into the eating state represented by firing the corresponding t_i transition. After the philosopher finishes eating, the t'_i transition can be fired such that the philosopher puts down both forks by marking the corresponding two fork places lf_i in his neighborhood and returns back into the thinking state by marking the corresponding thinking place th_i .

The structural changes of the net occurring when a philosopher joins or leaves a table are modeled by the corresponding PTI net transformation rules JoinTable and LeaveTable depicted in Figure 21 considering first the case of rules without nested application conditions. The upper part of Figure 21 shows the PTI net transformation rule JoinTable for reconfiguring the table when another philosopher joins it, which is realized by integrating the subnet representing a philosopher with his own fork into the current table structure. The transformation rule LeaveTable, depicted in the lower part of Figure 21, represents the situation when one philosopher decides to leave the table. In this case, the philosopher subnet with the corresponding fork is removed from the current table structure. Obviously, the transformation rule LeaveTable is inverse to the rule JoinTable, i.e., left-hand and right-hand sides of the rules are interchanged.

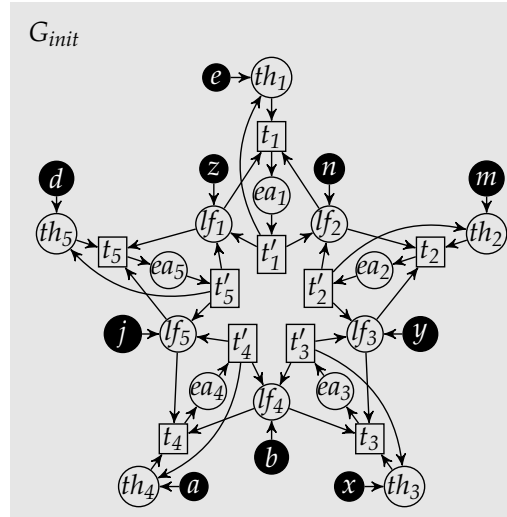


Figure 20: PTI net modeling five philosophers sitting at a table

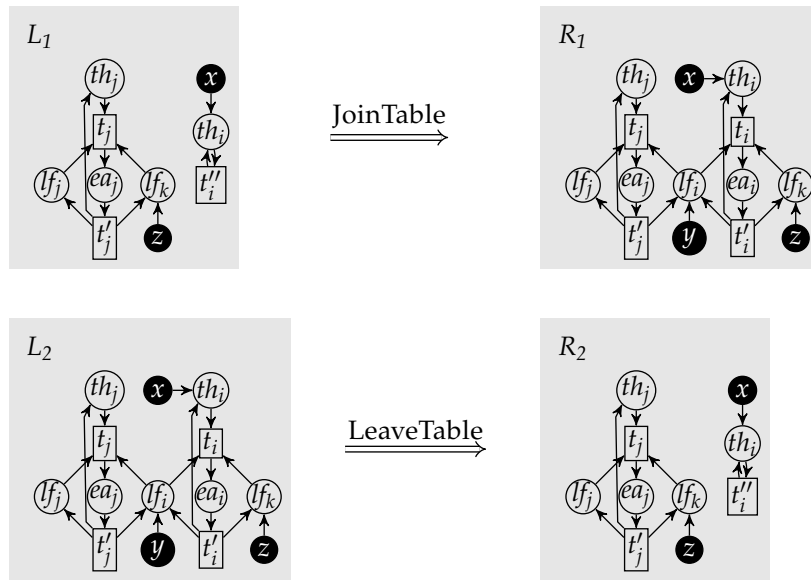


Figure 21: PTI net transformation rules without nested application conditions modeling the behavior of the Mobile Dining Philosophers scenario

To give later an example for local confluence analysis of PTI net transformation systems containing rules with nested application conditions, we extend the rule `LeaveTable` by an application condition. Using this application condition, given by a disjunction of two PACs, we ensure that a philosopher can leave the table only if there are at least two other philosophers sitting at the table. The adapted rule is given in [Figure 22](#). Since the left hand side already ensures that three philosophers are sitting at the table, these two additional PACs do not alter the behavior of the transformation system for the start PTI net at hand. We do not use alternative reasonable nested application conditions restricting the behavior of the Mobile Dining Philosophers scenario since their usage would result in diagrams too large for presentation purposes. However, the provided example is adequate for showing the impact on the (local) confluence analysis of PTI net transformation systems containing rules with nested application conditions presented in [Part iv](#).

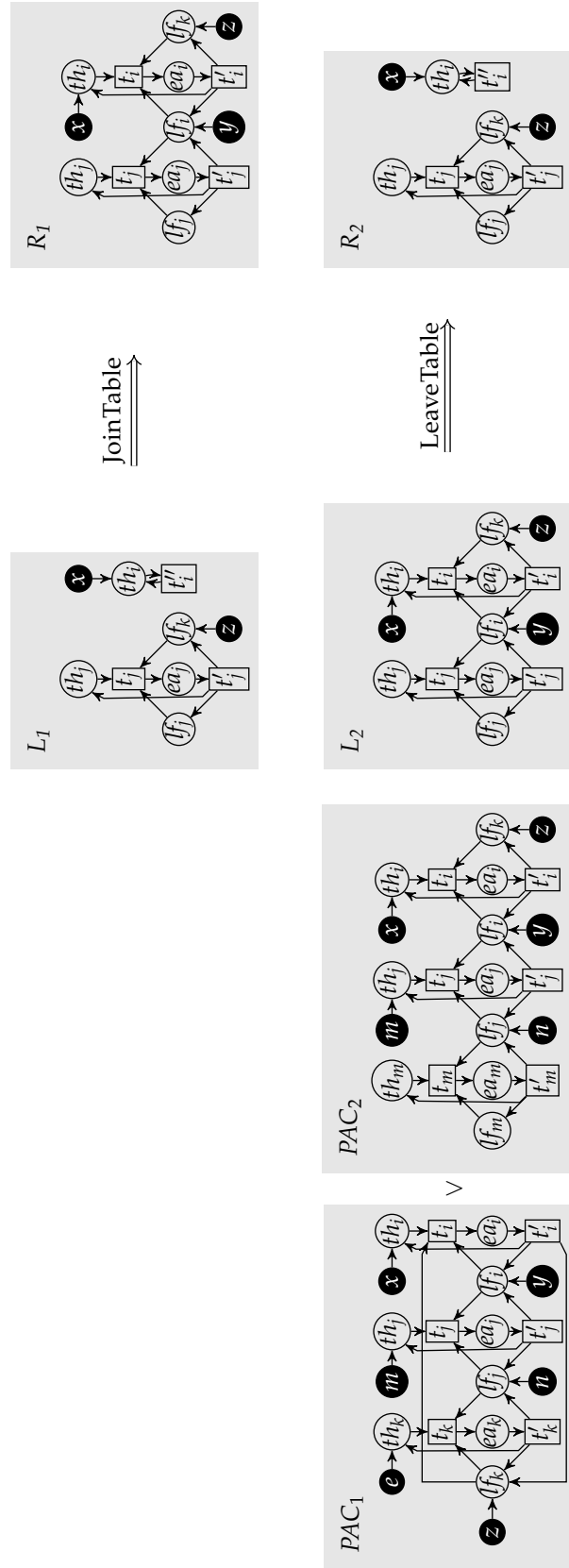


Figure 22: PTI net transformation rules with PACs modeling the behavior of the extended Mobile Dining Philosophers scenario

After our running example for a concrete PTI net transformation system is introduced, we give an additional example for the strict AC-confluence of critical pairs in the context of PTI net transformation systems.

Example 13 (Strict AC-Confluence of a Critical Pair in the Context of Mobile Dining Philosophers Scenario).

In this example, we construct a critical pair of the rule *LeaveTable* (see Figure 22) with itself and show that this critical pair is strictly AC-confluent according to Definition 29. First, we construct a weak critical pair, i.e., a pair of AC-disregarding transformations with jointly surjective PTI net morphisms α_1 and α_2 with $\alpha_1 = \alpha_2$ as given in Figure 23. The spans $K \leftarrow N_1 \rightarrow P_1$ and $K \leftarrow N_2 \rightarrow P_2$ in the lower part of the Figure 23 represent the two conflicting rule applications of the rule *LeaveTable* where one rule application deletes all transitions that are used by the other rule application and creates the corresponding new ones. Note that this weak critical pair does not satisfy the given nested application conditions, because there are no injective morphisms from PAC_1 to K or PAC_2 to K making the corresponding triangles between $L_2 = L'_2$, PAC_1 , and K or between $L_2 = L'_2$, PAC_2 , and K commute.

The weak critical pair in Figure 23 induces the following extension and conflict-inducing application conditions ac_K and ac_K^* on K according to Definition 25:

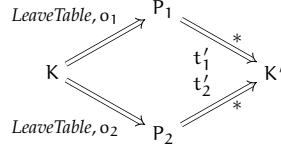
- $ac_K = \text{Shift}(\alpha_1, ac_{L_2}) \wedge \text{Shift}(\alpha_2, ac_{L'_2}) = ac_{K_1} \wedge ac_{K_2}$ and
- $ac_K^* = \neg(ac_{K,d_{21}}^* \wedge ac_{K,d_{12}}^*) = \neg(false \wedge false) = true$

The extension application condition ac_K consists of the application conditions ac_{K_1} and ac_{K_2} with $ac_{K_1} = ac_{K_2}$ marked by the outer gray rectangle in Figure 24, which we obtain by shifting of ac_{L_2} and $ac_{L'_2}$ (see the parts marked by dashed rectangles in Figure 23) along morphisms α_1 and α_2 to object K . The conflict-inducing application condition ac_K^* is true, because there are no morphisms $d_{12} : L_2 \rightarrow N_2$ and $d_{21} : L'_2 \rightarrow N_1$ (depicted by scratched dashed arrows in Figure 23) making the corresponding triangles commute.

The considered weak critical pair is a critical pair, because it can be embedded into a pair of AC-regarding transformations $H_1 \xleftarrow{\text{LeaveTable}, m_1} G \xrightarrow{\text{LeaveTable}, m_2} H_2$ (see Figure 25), so that the embedding morphism $m : K \rightarrow G$ satisfies $ac_K \wedge ac_K^*$ and the match morphisms $m_1 : L_2 \rightarrow G$, $m_2 : L'_2 \rightarrow G$ make the corresponding triangles between L_2 , K , and G resp. between L'_2 , K , and G commute.

Figure 26 shows that the embedding morphism $m : K \rightarrow G$ satisfies application conditions $ac_{K_1} = ac_{K_2}$ and hence also ac_K , because there is an injective PTI net morphism $q_1 : PAC'_1 \rightarrow G$ making the triangle (1) commute. Note that there is no injective PTI net morphism $q_2 : PAC'_2 \rightarrow G$ making the triangle (2) commute, but this does not arise a problem since for the satisfaction of an application condition consisting of two application conditions connected by the disjunction operator, it suffices that at least one of these conditions is satisfied (see Definition 16). Moreover, $m \models ac_K^*$ since every morphism satisfies true. Thus we get that $P_1 \xleftarrow{\text{LeaveTable}, \alpha_1} K \xrightarrow{\text{LeaveTable}, \alpha_2} P_2$ is a critical pair.

As the next step we have to check whether the critical pair constructed before is strictly AC-confluent, i.e., the merging transformation steps t'_1, t'_2 are strictly confluent for the case if we disregard application conditions as well as the extended AC-disregarding transformations $t_1 \triangleq K \xrightarrow{\text{LeaveTable}, \alpha_1} P_1 \xrightarrow{t'_1} K', t_2 \triangleq K \xrightarrow{\text{LeaveTable}, \alpha_2} P_2 \xrightarrow{t'_2} K'$ are AC-compatible.



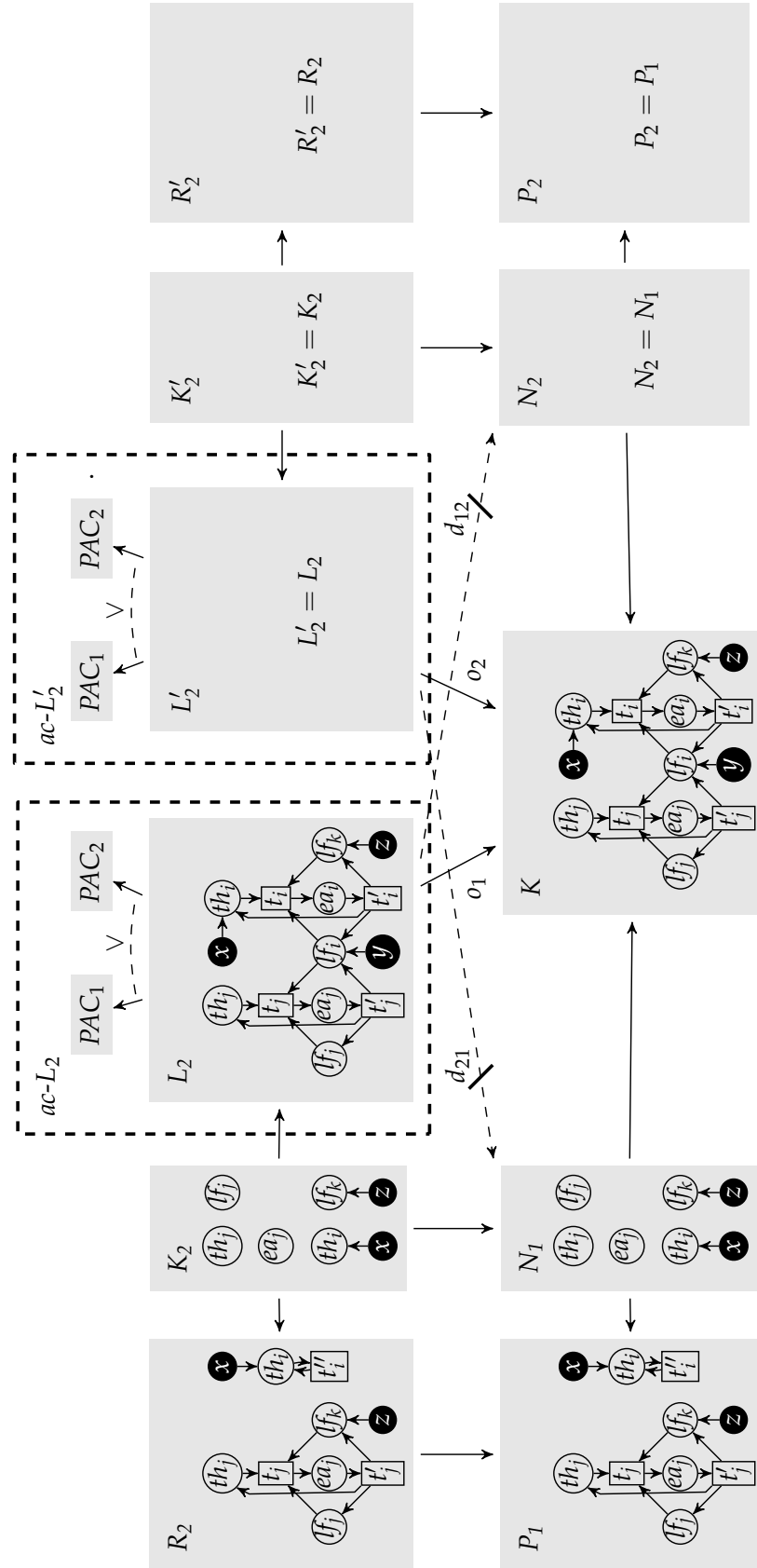
In [Figure 27](#)³⁵, we can see that the transformation steps $K \xrightarrow{\text{LeaveTable}, o_1} P_1$ and $K \xrightarrow{\text{LeaveTable}, o_2} P_2$ can be joined together to the PTI net K' , because after applying the rule `LeaveTable`, we can execute the identity transformation `Id` on the PTI nets P_1 and P_2 merging the both transformation steps together to the object K' that is isomorphic to the PTI nets P_1 and P_2 . Additionally, the maximal substructure of K (corresponding to the PTI net N_1 from [Figure 23](#)), that is preserved by the critical pair, is also preserved by the merging identity transformations $P_1 \xrightarrow{\text{Id}, o'_1} K'$ and $P_2 \xrightarrow{\text{Id}, o'_2} K'$. Therefore, the strictness condition holds as well.

Now it remains to show that the extended transformations t_1 and t_2 are AC-compatible, i.e., $(\text{ac}_K \wedge \text{ac}_K^*) \Rightarrow (\text{ac}(t_1) \wedge \text{ac}(t_2))$. For this reason we calculate the derived application conditions $\text{ac}(t_1)$ and $\text{ac}(t_2)$ obtaining the following:

$$\begin{aligned} \text{ac}(t_1) &= \text{ac}(K \xrightarrow{\text{LeaveTable}, o_1} P_1 \xrightarrow{\text{Id}, o'_1} K') = \text{ac}_{K_1} \wedge \text{true}, \\ \text{ac}(t_2) &= \text{ac}(K \xrightarrow{\text{LeaveTable}, o_2} P_2 \xrightarrow{\text{Id}, o'_2} K') = \text{ac}_{K_2} \wedge \text{true}, \\ &\Rightarrow \text{ac}(t_1) \wedge \text{ac}(t_2) = \text{ac}_{K_1} \wedge \text{ac}_{K_2} \end{aligned}$$

Since we already know that $\text{ac}_K \wedge \text{ac}_K^* = \text{ac}_{K_1} \wedge \text{ac}_{K_2}$, we obtain the AC-compatibility for t_1 and t_2 , because $(\text{ac}_{K_1} \wedge \text{ac}_{K_2}) \Rightarrow (\text{ac}_{K_1} \wedge \text{ac}_{K_2})$. Thus we have that the constructed critical pair $P_1 \xleftarrow{\text{LeaveTable}, o_1} K \xrightarrow{\text{LeaveTable}, o_2} P_2$ is strictly AC-confluent.

³⁵ Note that the inner square in [Figure 27](#) including the critical pair is AC-disregarding, but the outer square with G , H_1 , and H_2 is AC-regarding.

Figure 23: Weak critical pair of rules `LeaveTable` and `LeaveTable`

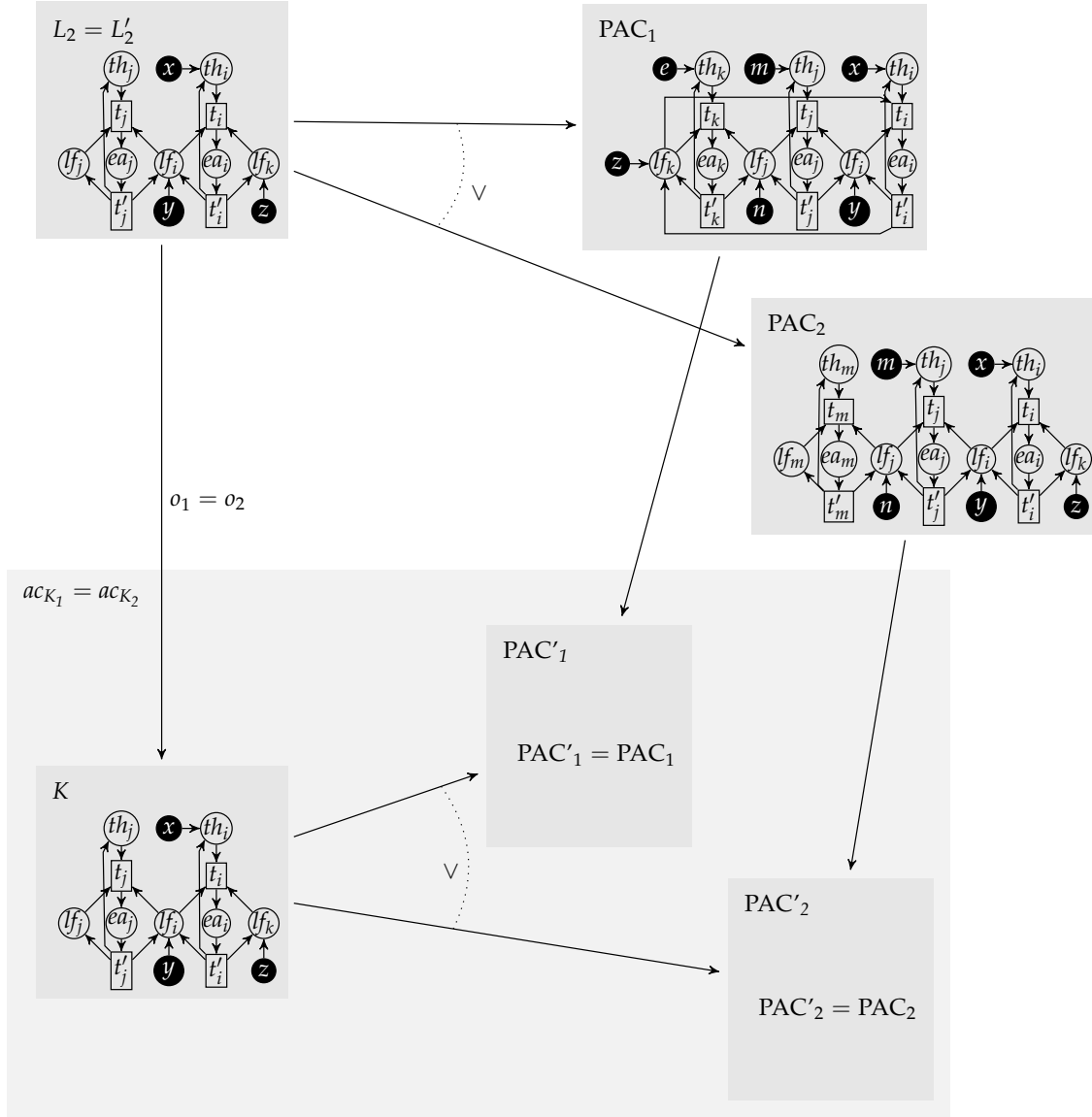
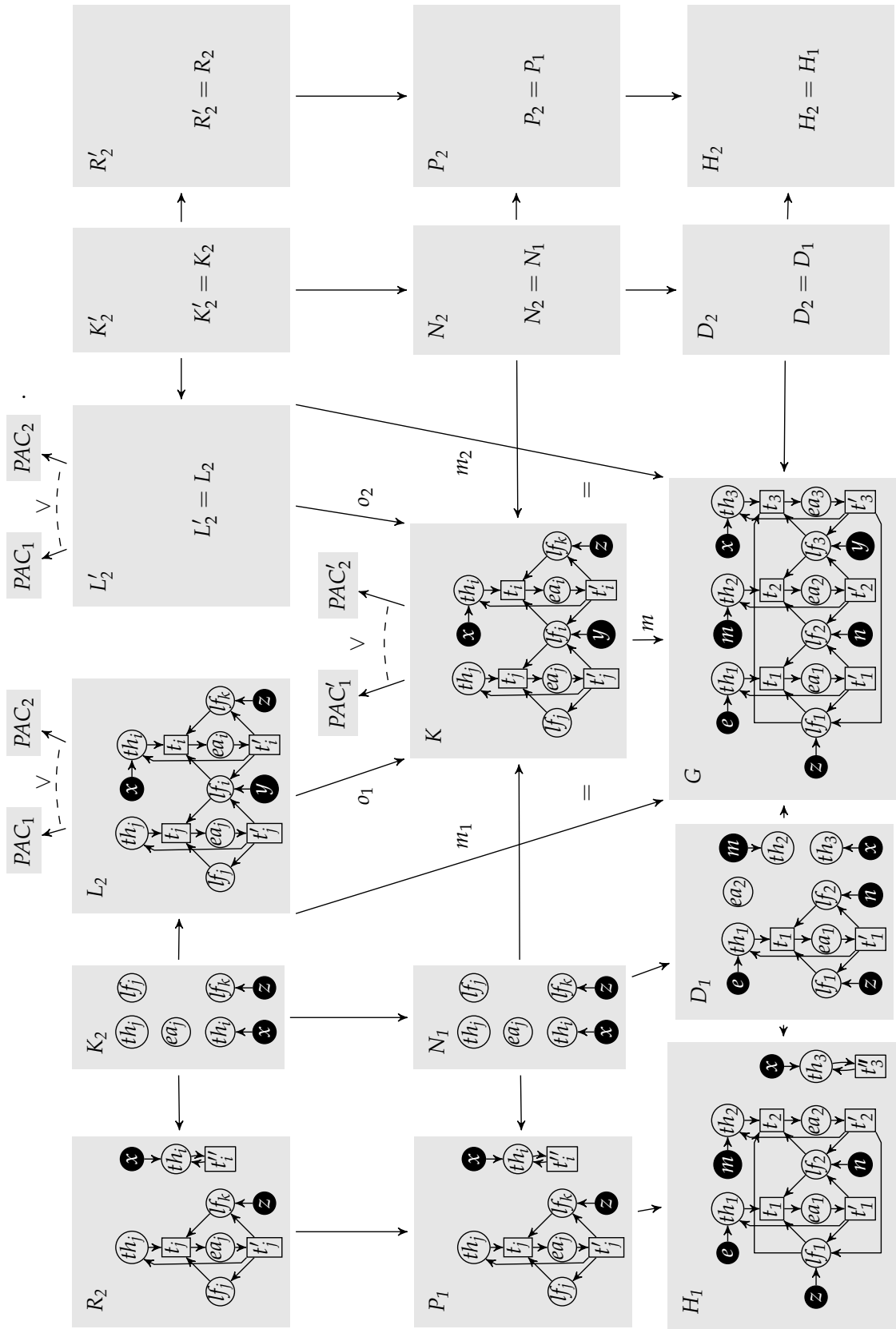


Figure 24: Extension application condition $ac_K = ac_{K_1} \wedge ac_{K_2}$ induced by the weak critical pair of rules `LeaveTable` and `LeaveTable`

Figure 25: Critical pair of rules *LeaveTable* and *LeaveTable*

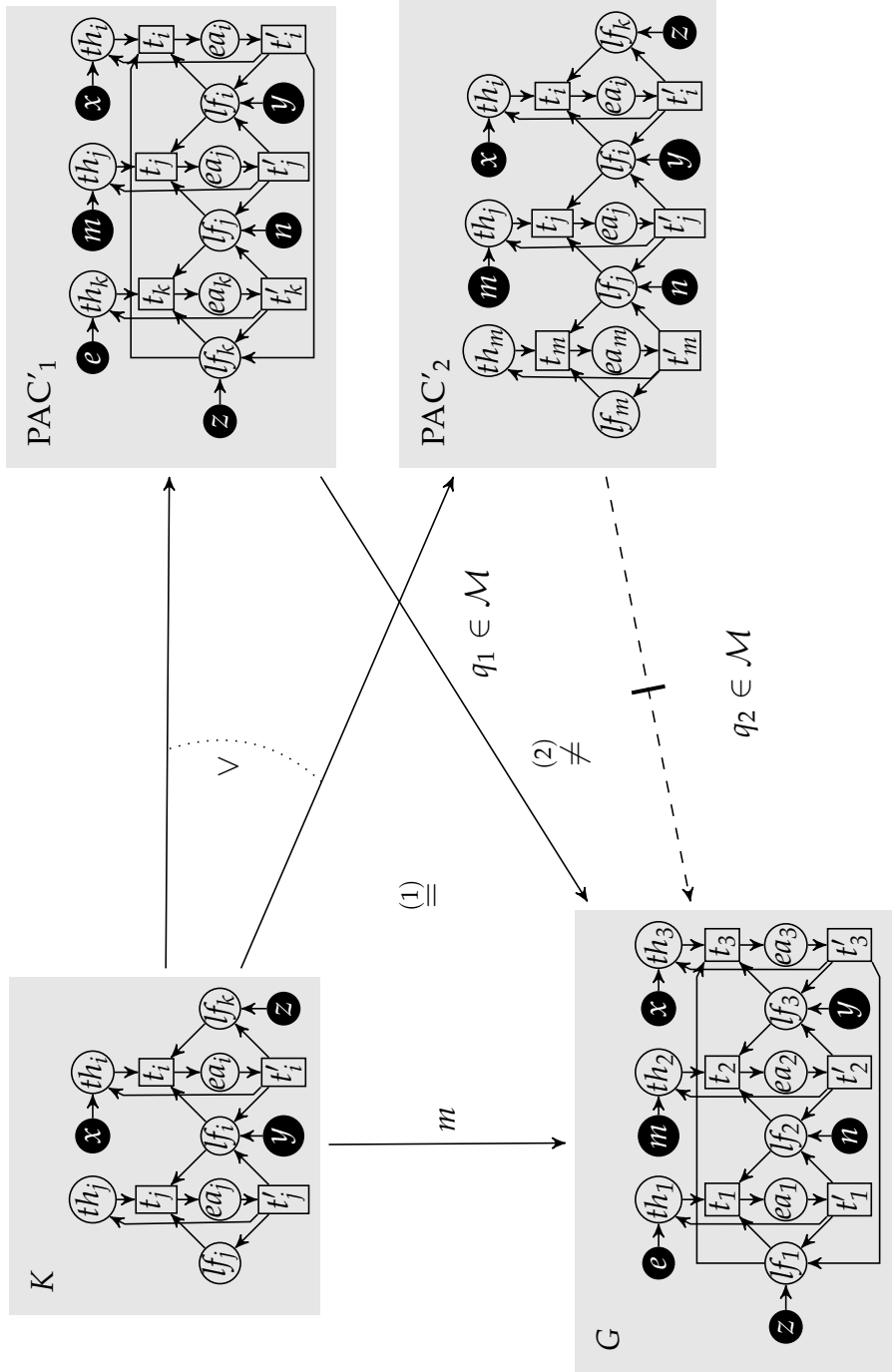
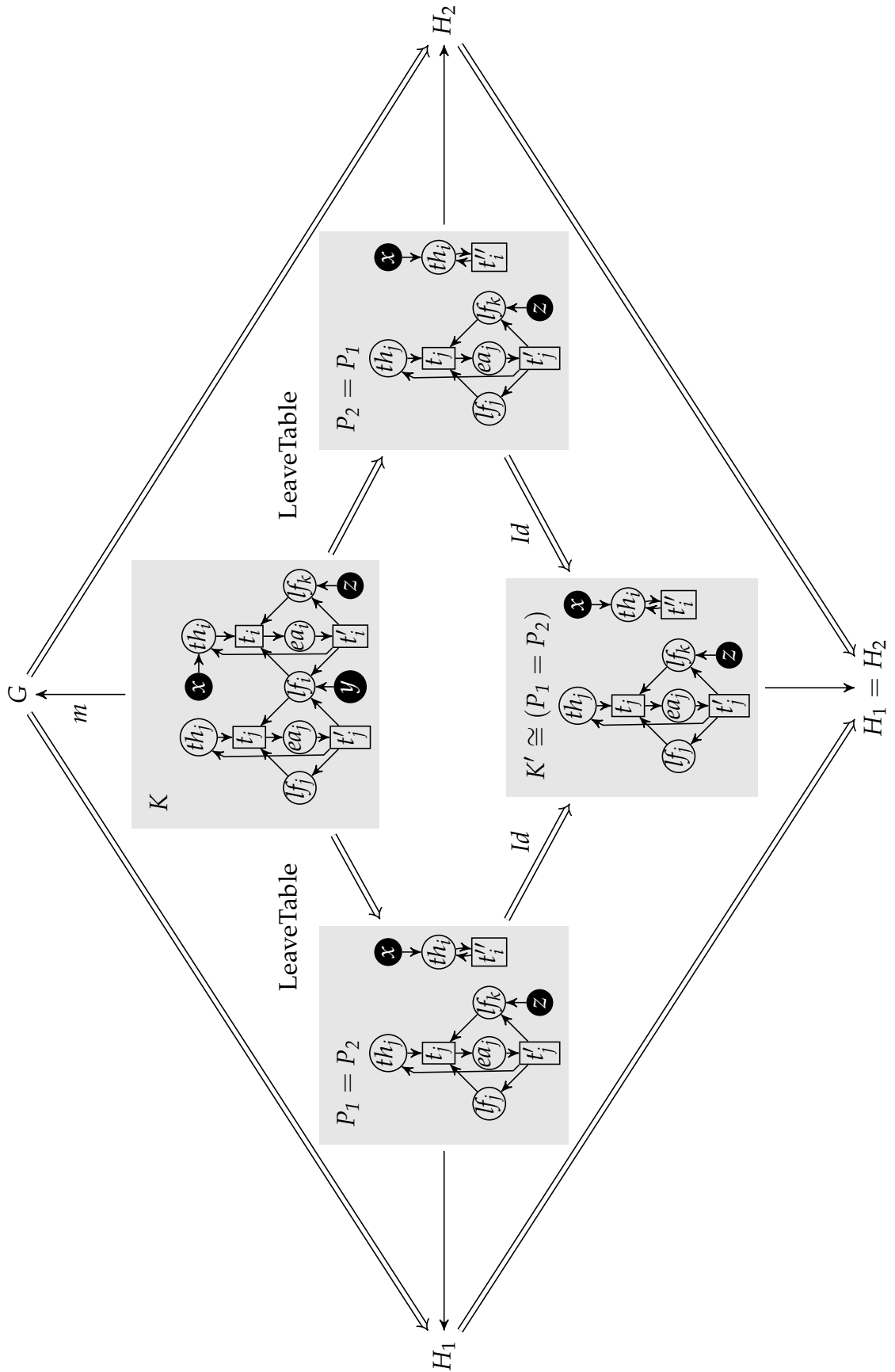


Figure 26: Embedding morphism $m : K \rightarrow G$ satisfies ac_K induced by the weak critical pair of rules LeaveTable and LeaveTable

Figure 27: Plain strict confluence of the critical pair of rules `LeaveTable` and `LeaveTable`

Part II

BEHAVIOR AND CONFLUENCE ANALYSIS OF \mathcal{M} -ADHESIVE TRANSFORMATION SYSTEMS

BEHAVIOR ANALYSIS OF \mathcal{M} -ADHESIVE TRANSFORMATION SYSTEMS USING \mathcal{M} -FUNCTORS

In this chapter, we introduce our framework of \mathcal{M} -functors that provides a foundation for the analysis of the system behavior and several relevant semantical properties of \mathcal{M} -adhesive transformation systems. In Section 3.1, we give a characterization of \mathcal{M} -functors by means of their sufficient technical properties ensuring that the objects related by an \mathcal{M} -functor behave equivalently with respect to the transformation steps defined according to the rules of the respective transformation systems. This semantical compatibility of \mathcal{M} -functors with the DPO-based transformation step relation is the key aspect for the behavior transfer. In Section 3.2, to capture the understanding of behavioral equivalence, we introduce two notions of bisimilarity and cover thereby the transfer of the multi-step behavior among the given transformation systems in a different way compared to before in Section 3.1. Using our first notion of bisimilarity, we verify that the objects of the source transformation system are indeed bisimilar to their functor-translated counterparts in the target transformation system, i.e., behavioral equivalent, as expected. Furthermore, using our second notion of bisimilarity, we show that \mathcal{M} -functors are also compatible with the multi-step behavior in the sense that bisimulations in the source transformation system of an \mathcal{M} -functor coincide with the corresponding translated bisimulations in the target transformation system. We consider bisimilarity in the context of these two notions because it is a reasonable choice for interactive systems where equivalent objects have equivalent enabled transformation steps that also preserve equivalence.

3.1 FUNCTORIAL BEHAVIOR TRANSFER BETWEEN \mathcal{M} -ADHESIVE TRANSFORMATION SYSTEMS

In this section, we introduce the notion of \mathcal{M} -functors between \mathcal{M} -adhesive categories that translate objects and morphisms from its source category into the corresponding objects and morphisms of its target category as well as consider how to use the framework of \mathcal{M} -functors to analyze the behavior of \mathcal{M} -adhesive transformation systems without or with nested application conditions. The behavior of an \mathcal{M} -adhesive transformation system is given by the possible transformation steps that can be executed following the DPO approach. As shown in the following, we can conclude under certain assumptions that the transformation steps, which are possible in the source transformation system of an \mathcal{M} -functor, are also possible in its target transformation system and vice versa, which altogether implies the behavioral equivalence of the functor-related parts of the source and the target transformation systems. Furthermore, for the case of \mathcal{M} -adhesive transformation systems in which the restriction of the morphisms of the category to only injective morphisms is required and where therefore the definition of general \mathcal{M} -functors is not possible, we adapt the introduced general theory to this special case defining an \mathcal{M} -functor on \mathcal{M} -morphisms only. We call this adapted kind of \mathcal{M} -functors the *restricted \mathcal{M} -functors*.

For the case of transformation systems containing rules with nested application conditions, a transformation step can be executed only if the application conditions of the

corresponding rule are satisfied. Considering this fact, we guarantee in the context of our framework that if a match morphism satisfies some nested application condition enabling a rule application then also the corresponding, by an \mathcal{M} -functor translated, match morphism satisfies the corresponding translated nested application condition and vice versa. Using this property, we can apply our framework also to the transformation systems containing rules with nested application conditions.

3.1.1 Translation and Creation of Transformations without Nested Application Conditions using \mathcal{M} -Functors

In order to obtain the desired semantical correspondence between two transformation systems, we need to ensure that the systems together with their operational behavior (or *behavior* for short) and relevant semantical properties are properly translated and reflected (the latter is also called *created* in the following). In this subsection, we establish a formal relationship between two corresponding \mathcal{M} -adhesive transformation systems. In particular, to ensure behavioral equivalence discussed before, we want to translate rule applicability and (direct) transformations of the first \mathcal{M} -adhesive transformation system into the corresponding rule applicability and (direct) transformations of the second \mathcal{M} -adhesive transformation system and, vice versa, we want to create rule applicability and (direct) transformations in the first \mathcal{M} -adhesive transformation system from those in the second \mathcal{M} -adhesive transformation system. Note that in this subsection, we first consider the case of transformations with arbitrary match morphisms, which we call *general match morphisms* from here on, and formulate all results only for this general case, while, afterwards, we discuss the differences as well as give some adapted definitions and results for the transformations with \mathcal{M} -match morphisms only.

We use the notion of an \mathcal{M} -functor [207] to define a formal relationship between two different \mathcal{M} -adhesive transformation systems.

Definition 41 (\mathcal{M} -Functor [207]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$ and $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$.¹ A functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ between two \mathcal{M} -adhesive categories is called \mathcal{M} -functor if $\mathcal{F}(\mathcal{M}_1) \subseteq \mathcal{M}_2$ and \mathcal{F} preserves pushouts along \mathcal{M} -morphisms².

\mathcal{M} -functors are assumed to satisfy the following basic technical properties, which are required among others to derive the main results of this thesis.

Definition 42 (Properties of \mathcal{M} -Functors [207]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$ and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$.

1. \mathcal{F} creates morphisms if for each general morphism $m' : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$ in $\text{Mor}_{\mathbf{C}_2}$ there is exactly one general morphism $m : L \rightarrow G$ in $\text{Mor}_{\mathbf{C}_1}$ such that $\mathcal{F}(m) = m'$.
2. \mathcal{F} preserves initial pushouts if for each initial pushout (1) in $(\mathbf{C}_1, \mathcal{M}_1)$ over a general match morphism $m : L \rightarrow G$ also (2) is an initial pushout in $(\mathbf{C}_2, \mathcal{M}_2)$ over the general match morphism $\mathcal{F}(m) : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$.

¹ By definition, each \mathcal{M} -functor \mathcal{F} translates each rule $\rho = (L \xleftarrow{l} K \xrightarrow{r} R)$ of the source \mathcal{M} -adhesive transformation system with $l, r \in \mathcal{M}_1$ into $\mathcal{F}(\rho) = (\mathcal{F}(L) \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R))$ in the target \mathcal{M} -adhesive transformation system with $\mathcal{F}(l), \mathcal{F}(r) \in \mathcal{M}_2$.

² Pushout along \mathcal{M} -morphisms means that at least one of the span morphisms of the pushout diagram is in \mathcal{M} .

$$\begin{array}{ccc}
 \begin{array}{ccc}
 B & \xrightarrow{b} & L \\
 \downarrow & (1) & \downarrow m \\
 C & \xrightarrow{c} & G
 \end{array} & \Rightarrow & \begin{array}{ccc}
 \mathcal{F}(B) & \xrightarrow{\mathcal{F}(b)} & \mathcal{F}(L) \\
 \downarrow & (2) & \downarrow \mathcal{F}(m) \\
 \mathcal{F}(C) & \xrightarrow{\mathcal{F}(c)} & \mathcal{F}(G)
 \end{array}
 \end{array}
 \quad \begin{array}{l} \text{IPO in } (\mathbf{C}_1, \mathcal{M}_1) \\ \text{IPO in } (\mathbf{C}_2, \mathcal{M}_2) \end{array}$$

The technical property of morphism creation guarantees further beneficial technical properties used throughout the thesis, namely, that \mathcal{F} creates identities and isomorphisms as well as that \mathcal{F} is injective on objects and morphisms as stated in the following three lemmas.

Lemma 8 (\mathcal{F} Creates Identities and Isomorphisms).

Consider \mathcal{M} -adhesive categories $(\mathbf{C}_1, \mathcal{M}_1)$ and $(\mathbf{C}_2, \mathcal{M}_2)$. Then an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ creates identities and isomorphisms if \mathcal{F} creates morphisms.

Proof.

The detailed proof for this lemma is given in [Appendix B on page 275](#). □

Lemma 9 (\mathcal{F} is Injective on Objects).

Consider \mathcal{M} -adhesive categories $(\mathbf{C}_1, \mathcal{M}_1)$ and $(\mathbf{C}_2, \mathcal{M}_2)$. Then an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ is injective on objects, i.e.,

$$\forall H_1, H_2 \in \text{Ob}_{\mathbf{C}_1}. (\mathcal{F}(H_1) = \mathcal{F}(H_2)) \Rightarrow (H_1 = H_2)$$

if \mathcal{F} creates morphisms.

Proof.

The detailed proof for this lemma is given in [Appendix B on page 276](#). □

Lemma 10 (\mathcal{F} is Injective on Morphisms).

Consider \mathcal{M} -adhesive categories $(\mathbf{C}_1, \mathcal{M}_1)$ and $(\mathbf{C}_2, \mathcal{M}_2)$. Then an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ is injective on morphisms, i.e.,

$$\forall (\mathcal{F}(m), \mathcal{F}(n) : \mathcal{F}(G) \rightarrow \mathcal{F}(H)) \in \text{Mor}_{\mathbf{C}_2}. (\mathcal{F}(m) = \mathcal{F}(n)) \Rightarrow (m = n)$$

if \mathcal{F} creates morphisms.

Proof.

The detailed proof for this lemma is given in [Appendix B on page 276](#). □

Subsequently, we define when an \mathcal{M} -functor translates and creates the operational behavior between the involved \mathcal{M} -adhesive transformation systems. This operational behavior is given as usual by direct transformations and transformations, i.e., sequences of direct transformations written $G \xrightarrow{*} H$.

Definition 43 (Translation and Creation of Rule Applicability and Direct Transformations with General Match Morphisms [206]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$.

- \mathcal{F} translates applicability of a rule $\rho = (L \xleftarrow{l} K \xrightarrow{r} R)$ in P to some object G in $\text{Ob}_{\mathbf{C}_1}$ if the applicability of ρ to G at some general match morphism $m : L \rightarrow G$ implies the applicability of $\mathcal{F}(\rho) = (\mathcal{F}(L) \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R))$ to $\mathcal{F}(G)$ at the general match morphism $\mathcal{F}(m) : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$.

- \mathcal{F} translates direct transformations if for each direct transformation $G \xrightarrow{\rho, m} H$ in AS_1 given by DPO (1) + (2) with a general match morphism $m : L \rightarrow G$ there is a direct transformation $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(m)} \mathcal{F}(H)$ in AS_2 given by DPO (3) + (4) with the general match morphism $\mathcal{F}(m) : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$.

$$\begin{array}{ccc}
 L & \xleftarrow{l} K & \xrightarrow{r} R \\
 m \downarrow & (1) \downarrow & (2) \downarrow \\
 G & \xleftarrow{\quad} D & \xrightarrow{\quad} H
 \end{array}
 \Rightarrow
 \begin{array}{ccc}
 \mathcal{F}(L) & \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) & \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R) \\
 \mathcal{F}(m) \downarrow & (3) \downarrow & (4) \downarrow \\
 \mathcal{F}(G) & \xleftarrow{\quad} \mathcal{F}(D) & \xrightarrow{\quad} \mathcal{F}(H)
 \end{array}$$

- \mathcal{F} creates applicability of a rule $\rho = (L \xleftarrow{l} K \xrightarrow{r} R)$ in P to some object G in Ob_{C_1} if the applicability of $\mathcal{F}(\rho) = (\mathcal{F}(L) \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R))$ to $\mathcal{F}(G)$ at some general match morphism $m' : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$ implies the applicability of ρ to G at the general match morphism $m : L \rightarrow G$ and $\mathcal{F}(m) = m'$.
- \mathcal{F} creates direct transformations if for each direct transformation $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), m'} H'$ in AS_2 given by DPO (1') + (2') with a general match morphism $m' : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$ there is a direct transformation $G \xrightarrow{\rho, m} H$ in AS_1 given by DPO (3') + (4') with the general match morphism $m : L \rightarrow G$ such that $\mathcal{F}(m) = m'$ and $\mathcal{F}(H) \cong H'^3$ leading to the direct transformation $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(m)} \mathcal{F}(H)$ in AS_2 .

$$\begin{array}{ccc}
 \mathcal{F}(L) & \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) & \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R) \\
 m' = \mathcal{F}(m) \downarrow & (1') \downarrow & (2') \downarrow \\
 \mathcal{F}(G) & \xleftarrow{\quad} \mathcal{F}(D) & \xrightarrow{\quad} H' \cong \mathcal{F}(H)
 \end{array}
 \Rightarrow
 \begin{array}{ccc}
 L & \xleftarrow{l} K & \xrightarrow{r} R \\
 m \downarrow & (3') \downarrow & (4') \downarrow \\
 G & \xleftarrow{\quad} D & \xrightarrow{\quad} H
 \end{array}$$

Consider an \mathcal{M} -adhesive transformation system $AS_1 = (C_1, \mathcal{M}_1, P)$. As already mentioned before, we want to translate rule applicability and (direct) transformations from AS_1 to $AS_2 = (C_2, \mathcal{M}_2, \mathcal{F}(P))$ with translated rules $\mathcal{F}(P)$ and, vice versa, to create rule applicability and (direct) transformations in AS_1 from those in AS_2 . This can be handled by the following theorem already introduced and shown in [207]. Note that for the source and the target transformation systems of an \mathcal{M} -functor \mathcal{F} , Theorem 1 guarantees the behavioral equivalence of their functor-related parts.

Theorem 1 (Translation and Creation of Rule Applicability and (Direct) Transformations for Rules without Nested Application Conditions [207]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (C_1, \mathcal{M}_1, P)$ and $AS_2 = (C_2, \mathcal{M}_2, \mathcal{F}(P))$. Each \mathcal{M} -functor $\mathcal{F} : (C_1, \mathcal{M}_1) \rightarrow (C_2, \mathcal{M}_2)$ translates rule applicability and (direct) transformations. Vice versa, \mathcal{F} creates rule applicability and (direct) transformations if (C_1, \mathcal{M}_1) has initial pushouts and \mathcal{F} creates morphisms as well as preserves initial pushouts.

Proof.

- **Translation:**

- \mathcal{F} translates rule applicability:

Consider additionally a rule $\rho = (L \xleftarrow{l} K \xrightarrow{r} R) \in P_1$, which is applicable to some object G in AS_1 at a general match morphism $m : L \rightarrow G$. Since ρ is applicable to G in AS_1 by assumption, there is a pushout complement D in a pushout (1). This implies that there is the pushout complement $\mathcal{F}(D)$ in pushout (3), because \mathcal{F} preserves pushouts along \mathcal{M} -morphisms. Thus, we obtain that the rule $\mathcal{F}(\rho) \in P_2$

³ $A \cong B$ means that there is an isomorphism $i : A \rightarrow B$.

with $\mathcal{F}(\rho) = (\mathcal{F}(L) \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R))$ is applicable to $\mathcal{F}(G)$ in AS_2 at the general match morphism $\mathcal{F}(m) : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$.

– **\mathcal{F} translates direct transformations:**

The fact that (2) is a pushout in \mathbf{C}_1 implies directly that (4) is a pushout in \mathbf{C}_2 , because \mathcal{F} preserves pushouts along \mathcal{M} -morphisms. Furthermore, the direct transformation $G \xrightarrow{\rho, m} H$ with a general match morphism m in AS_1 implies the existence of the corresponding direct transformation $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(m)} \mathcal{F}(H)$ with a general match morphism $\mathcal{F}(m) : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$ in AS_2 . Thus, we have by Definition 43 that \mathcal{F} translates AS_1 -transformations into the corresponding AS_2 -transformations.

$$\begin{array}{ccc}
 L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
 m \downarrow & (1) & \downarrow & (2) & \downarrow \\
 G & \longleftarrow & D & \longrightarrow & H
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{ccc}
 \mathcal{F}(L) & \xleftarrow{\mathcal{F}(l)} & \mathcal{F}(K) & \xrightarrow{\mathcal{F}(r)} & \mathcal{F}(R) \\
 \mathcal{F}(m) \downarrow & (3) & \downarrow & (4) & \downarrow \\
 \mathcal{F}(G) & \longleftarrow & \mathcal{F}(D) & \longrightarrow & \mathcal{F}(H)
 \end{array}$$

• **Creation:**

– **\mathcal{F} creates rule applicability:**

Consider additionally a rule $\mathcal{F}(\rho) = (\mathcal{F}(L) \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R)) \in P_2$, which is applicable to some object $\mathcal{F}(G)$ in AS_2 at a general match morphism $m' : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$. Since \mathcal{F} creates morphisms by assumption, we have a unique general match morphism $m : L \rightarrow G$ with $\mathcal{F}(m) = m'$. Let (1) be an initial pushout over m in $(\mathbf{C}_1, \mathcal{M}_1)$. By assumption on \mathcal{F} , (2) is an initial pushout over $\mathcal{F}(m)$ and (4), (5) are pushouts in $(\mathbf{C}_2, \mathcal{M}_2)$. This means that all requirements for the applicability of the rule $\mathcal{F}(\rho)$ to the object $\mathcal{F}(G)$ at the match morphism $m' = \mathcal{F}(m)$ are satisfied. According to Definition 5, this implies the existence of a morphism $b'' : \mathcal{F}(B) \rightarrow \mathcal{F}(K)$ in \mathcal{M}_2 with $\mathcal{F}(l) \circ b'' = \mathcal{F}(b)$. Since \mathcal{F} creates morphisms by assumption, there is a unique morphism $b' : B \rightarrow K$ with $\mathcal{F}(b') = b''$. Moreover, the uniqueness of morphism creation implies that $l \circ b' = b$ and hence $b' \in \mathcal{M}_1$ by decomposition property of \mathcal{M}_1 . Therefore, the gluing condition is satisfied and we have applicability of the rule ρ to the object G at the general match morphism $m : L \rightarrow G$ and a pushout complement D in pushout (3).

– **\mathcal{F} creates direct transformations:**

Consider the direct transformation $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), m'} H'$ given by pushouts (4) and (5) with a general match morphism m' in AS_2 . We have already constructed the pushout (3) in \mathbf{C}_1 and can construct the pushout (6) along the morphism $r \in \mathcal{M}_1$ leading to a direct transformation $G \xrightarrow{\rho, m} H$ with a general match morphism m in AS_1 . Since \mathcal{F} preserves pushouts along \mathcal{M} -morphisms and pushout complements in \mathbf{C}_2 are unique up to isomorphism, we have that $\mathcal{F}(D) \cong D'$, $\mathcal{F}(H) \cong H'$ and hence also $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(m)} \mathcal{F}(H)$ with the general match morphism $\mathcal{F}(m)$ is a direct transformation in AS_2 . Thus, we have by Definition 43 that \mathcal{F} creates AS_1 -transformations from the corresponding AS_2 -transformations.

$$\begin{array}{ccc}
\begin{array}{ccccc}
& & b' & & \\
& \curvearrowright & & \curvearrowleft & \\
B & \xrightarrow{b} & L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
\downarrow g & (1) & \downarrow m & (3) & \downarrow & (6) & \downarrow \\
C & \longrightarrow & G & \longleftarrow & D & \longrightarrow & H
\end{array} & \Longleftarrow &
\begin{array}{ccccccc}
& & b'' & & & & \\
& \curvearrowright & & \curvearrowleft & & & \\
\mathcal{F}(B) & \xrightarrow{\mathcal{F}(b)} & \mathcal{F}(L) & \xleftarrow{\mathcal{F}(l)} & \mathcal{F}(K) & \xrightarrow{\mathcal{F}(r)} & \mathcal{F}(R) \\
\downarrow \mathcal{F}(g) & (2) & \downarrow m' & (4) & \downarrow & (5) & \downarrow \\
\mathcal{F}(C) & \longrightarrow & \mathcal{F}(G) & \longleftarrow & D' & \longrightarrow & H'
\end{array}
\end{array}$$

□

The result from Theorem 1 holds also for sequences of direct transformations. This can be easily shown by induction over the length of a transformation sequence where Theorem 1 is used to show the induction step.

If we want to consider only (direct) transformations with injective match morphisms, as it is reasonable in the case of PTI net transformations in our application, then it suffices to define the \mathcal{M} -functor \mathcal{F} on injective morphisms only. We call this special kind of an \mathcal{M} -functor a *restricted \mathcal{M} -functor* [207] and denote it in the following by \mathcal{F}_R . Moreover, the mentioned restriction is necessary if the non-restricted functor is not well-defined for non-injective morphisms. Subsequently, we discuss the adaptation of all notions, constructions, and results already considered in this subsection to the case of transformations with \mathcal{M} -match morphisms. We apply this adapted theory in Part iv to the functor between the categories of PTI nets and typed attributed graphs.

According to [207], the formal definition of a restricted \mathcal{M} -functor is as follows.

Definition 44 (Restricted \mathcal{M} -Functor [207]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$ and $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}_R(P))$. A functor $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$ between two \mathcal{M} -adhesive categories $(\mathbf{C}_1, \mathcal{M}_1)$ and $(\mathbf{C}_2, \mathcal{M}_2)$ with $\mathbf{C}_i|_{\mathcal{M}_i}$ the restriction of \mathbf{C}_i to \mathcal{M}_i -morphisms for $i \in \{1, 2\}$ is called a *restricted \mathcal{M} -functor* if $\mathcal{F}_R(\mathcal{M}_1) \subseteq \mathcal{M}_2$ and \mathcal{F}_R translates pushouts of \mathcal{M}_1 -morphisms⁴ in $(\mathbf{C}_1, \mathcal{M}_1)$ into pushouts of \mathcal{M}_2 -morphisms in $(\mathbf{C}_2, \mathcal{M}_2)$.

The following two definitions are very similar to the Definitions 42 and 43 already introduced in this subsection. Nevertheless, we include these definitions to clarify the arising differences compared to the general case.

Definition 45 (Properties of Restricted \mathcal{M} -Functors [206]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}_R(P))$, and a restricted \mathcal{M} -functor $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$.

1. \mathcal{F}_R creates \mathcal{M} -morphisms if for each \mathcal{M}_2 -morphism $m' : \mathcal{F}_R(L) \rightarrow \mathcal{F}_R(G)$ in $\text{Mor}_{\mathbf{C}_2}$ there exists exactly one \mathcal{M}_1 -morphism $m : L \rightarrow G$ in $\text{Mor}_{\mathbf{C}_1}$ such that $\mathcal{F}_R(m) = m'$.
2. \mathcal{F}_R preserves initial pushouts over \mathcal{M} -morphisms if for each initial pushout (1) in $(\mathbf{C}_1, \mathcal{M}_1)$ over an \mathcal{M}_1 -morphism $m : L \rightarrow G$ also (2) is an initial pushout in $(\mathbf{C}_2, \mathcal{M}_2)$ over an \mathcal{M}_2 -morphism $\mathcal{F}_R(m) : \mathcal{F}_R(L) \rightarrow \mathcal{F}_R(G)$.

$$\begin{array}{ccc}
\begin{array}{ccc}
B & \xrightarrow{b} & L \\
\downarrow & (1) & \downarrow m \\
C & \xrightarrow{c} & G
\end{array} & \Rightarrow &
\begin{array}{ccc}
\mathcal{F}_R(B) & \xrightarrow{\mathcal{F}_R(b)} & \mathcal{F}_R(L) \\
\downarrow & (2) & \downarrow \mathcal{F}_R(m) \\
\mathcal{F}_R(C) & \xrightarrow{\mathcal{F}_R(c)} & \mathcal{F}_R(G)
\end{array} \\
\text{IPO in } (\mathbf{C}_1, \mathcal{M}_1) & & \text{IPO in } (\mathbf{C}_2, \mathcal{M}_2)
\end{array}$$

⁴ Pushout of \mathcal{M} -morphisms means that both of the span morphisms of the pushout diagram are in \mathcal{M} .

Definition 46 (Translation and Creation of Rule Applicability and Direct Transformations with \mathcal{M} -Match Morphisms [206]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}_R(P))$, and a restricted \mathcal{M} -functor $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$.

- \mathcal{F}_R translates applicability of a rule $\rho = (L \xleftarrow{l} K \xrightarrow{r} R)$ in P to some object G in $\text{Ob}_{\mathbf{C}_1}$ if the applicability of ρ to G at some \mathcal{M} -match morphism $m : L \rightarrow G$ implies the applicability of $\mathcal{F}_R(\rho) = (\mathcal{F}_R(L) \xleftarrow{\mathcal{F}_R(l)} \mathcal{F}_R(K) \xrightarrow{\mathcal{F}_R(r)} \mathcal{F}_R(R))$ to $\mathcal{F}_R(G)$ at the \mathcal{M} -match morphism $\mathcal{F}_R(m) : \mathcal{F}_R(L) \rightarrow \mathcal{F}_R(G)$.
- \mathcal{F}_R translates direct transformations if for each direct transformation $G \xrightarrow{\rho, m} H$ in AS_1 given by DPO (1) + (2) with an \mathcal{M} -match morphism $m : L \rightarrow G$ there is a direct transformation $\mathcal{F}_R(G) \xrightarrow{\mathcal{F}_R(\rho), \mathcal{F}_R(m)} \mathcal{F}_R(H)$ in AS_2 given by DPO (3) + (4) with the \mathcal{M} -match morphism $\mathcal{F}_R(m) : \mathcal{F}_R(L) \rightarrow \mathcal{F}_R(G)$.

$$\begin{array}{ccc}
 L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
 m \downarrow & (1) & \downarrow & (2) & \downarrow \\
 G & \xleftarrow{\quad} & D & \xrightarrow{\quad} & H
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{ccc}
 \mathcal{F}_R(L) & \xleftarrow{\mathcal{F}_R(l)} & \mathcal{F}_R(K) & \xrightarrow{\mathcal{F}_R(r)} & \mathcal{F}_R(R) \\
 \mathcal{F}_R(m) \downarrow & (3) & \downarrow & (4) & \downarrow \\
 \mathcal{F}_R(G) & \xleftarrow{\quad} & \mathcal{F}_R(D) & \xrightarrow{\quad} & \mathcal{F}_R(H)
 \end{array}$$

- \mathcal{F}_R creates applicability of a rule $\rho = (L \xleftarrow{l} K \xrightarrow{r} R)$ in P to some object G in $\text{Ob}_{\mathbf{C}_1}$ if the applicability of $\mathcal{F}_R(\rho) = (\mathcal{F}_R(L) \xleftarrow{\mathcal{F}_R(l)} \mathcal{F}_R(K) \xrightarrow{\mathcal{F}_R(r)} \mathcal{F}_R(R))$ to $\mathcal{F}_R(G)$ at some \mathcal{M} -match morphism $m' : \mathcal{F}_R(L) \rightarrow \mathcal{F}_R(G)$ implies the applicability of ρ to G at the \mathcal{M} -match morphism $m : L \rightarrow G$ and $\mathcal{F}_R(m) = m'$.
- \mathcal{F}_R creates direct transformations if for each direct transformation $\mathcal{F}_R(G) \xrightarrow{\mathcal{F}_R(\rho), m'} H'$ in AS_2 given by DPO (1') + (2') with an \mathcal{M} -match morphism $m' : \mathcal{F}_R(L) \rightarrow \mathcal{F}_R(G)$ there is a direct transformation $G \xrightarrow{\rho, m} H$ in AS_1 given by DPO (3') + (4') with the \mathcal{M} -match morphism $m : L \rightarrow G$ such that $\mathcal{F}_R(m) = m'$ and $\mathcal{F}_R(H) \cong H'$ leading to the direct transformation $\mathcal{F}_R(G) \xrightarrow{\mathcal{F}_R(\rho), \mathcal{F}_R(m)} \mathcal{F}_R(H)$ in AS_2 .

$$\begin{array}{ccc}
 \mathcal{F}_R(L) & \xleftarrow{\mathcal{F}_R(l)} & \mathcal{F}_R(K) & \xrightarrow{\mathcal{F}_R(r)} & \mathcal{F}_R(R) \\
 m' = \mathcal{F}_R(m) \downarrow & (1') & \downarrow & (2') & \downarrow \\
 \mathcal{F}_R(G) & \xleftarrow{\quad} & \mathcal{F}_R(D) & \xrightarrow{\quad} & H' \cong \mathcal{F}_R(H)
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{ccc}
 L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
 m \downarrow & (3') & \downarrow & (4') & \downarrow \\
 G & \xleftarrow{\quad} & D & \xrightarrow{\quad} & H
 \end{array}$$

The following theorem adapts Theorem 1 to the case of transformations with \mathcal{M} -match morphisms ensuring the behavioral equivalence of the functor-related parts of the source and the target transformation systems of a restricted \mathcal{M} -functor. Each restricted \mathcal{M} -functor must satisfy the adapted requirements to be able to translate and create rule applicability and (direct) transformations.

Theorem 2 (Translation and Creation of Rule Applicability and (Direct) Transformations with \mathcal{M} -Match Morphisms for Rules without Nested Application Conditions [207]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$ and $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}_R(P))$. Each restricted \mathcal{M} -functor $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$ translates rule applicability and (direct) transformations. Vice versa, \mathcal{F}_R creates rule applicability and (direct) transformations if $(\mathbf{C}_1, \mathcal{M}_1)$ has initial pushouts and \mathcal{F}_R creates \mathcal{M} -morphisms as well as preserves initial pushouts over \mathcal{M} -morphisms.

Proof.

The proof for this theorem works analogously to the proof of Theorem 1. For more details see [206].

□

To summarize the additional properties of restricted \mathcal{M} -functors, we introduce the following remark.

Remark 10 (Additional Properties Holding for Restricted \mathcal{M} -Functors).

The properties given in Lemmas 8 - 10 hold also for restricted \mathcal{M} -functors. The proofs for these adapted properties work analogously using as assumptions the corresponding restricted \mathcal{M} -functor properties given in Definition 45. Note that we cannot replace the \mathcal{M} -adhesive categories $(\mathbf{C}_i, \mathcal{M}_i)$ for $i \in \{1, 2\}$ by $(\mathbf{C}_i|_{\mathcal{M}_i}, \mathcal{M}_i)$, because $(\mathbf{C}_i|_{\mathcal{M}_i}, \mathcal{M}_i)$ are in general not \mathcal{M} -adhesive.

In the following chapters, we are introducing various definitions and results for the general setting of \mathcal{M} -functors, which could be often formulated identically for the setting of restricted \mathcal{M} -functors. For brevity and readability, we omit the explicit inclusion of all these similar definitions and results. Where necessary, we point out how the corresponding definitions and results for the setting of restricted \mathcal{M} -functors deviate.

3.1.2 Translation and Creation of Transformations with Nested Application Conditions using \mathcal{M} -Functors

In this subsection, we extend our theoretical results on the translation and creation of rule applicability and (direct) transformations introduced in the previous subsection to the case of transformation systems with nested application conditions enabling their functorial behavior transfer. In particular, we first define how nested application conditions can be translated using \mathcal{M} -functors as well as recall how the satisfaction relation is defined for translated nested application conditions. Afterwards, we show that a nested application condition is satisfied by some morphism in the source transformation system of the considered \mathcal{M} -functor if and only if the corresponding translated nested application condition is satisfied by the corresponding translated morphism in the target transformation system. This result is important for the extension of Theorem 1 to transformations with nested application conditions, which we present in the end of this subsection in Theorem 3.

As the first step, we define how nested application conditions can be translated by an \mathcal{M} -functor as well as adapt the notion of satisfaction to the case of translated nested application conditions.

Definition 47 (Translated Nested Application Condition [213]).

Consider an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$. A nested application condition ac_P over an object P can be translated by \mathcal{F} into a nested application condition $\mathcal{F}(\text{ac}_P)$ over an object $\mathcal{F}(P)$ as follows:

- $\mathcal{F}(\text{true}) = \text{true}$ for $\text{ac}_P = \text{true}$,
- $\mathcal{F}(\exists(a, \text{ac}_C)) = \exists(\mathcal{F}(a), \mathcal{F}(\text{ac}_C))$ for $\text{ac}_P = \exists(a, \text{ac}_C)$,
- $\mathcal{F}(\neg \text{ac}_P) = \neg(\mathcal{F}(\text{ac}_P))$,
- $\mathcal{F}(\bigwedge_{i \in \mathcal{I}} \text{ac}_{P,i}) = \bigwedge_{i \in \mathcal{I}} \mathcal{F}(\text{ac}_{P,i})$ where $\text{ac}_{P,i}$ is a nested application condition with $i \in \mathcal{I}$ for some index set \mathcal{I} ,
- $\mathcal{F}(\bigvee_{i \in \mathcal{I}} \text{ac}_{P,i}) = \bigvee_{i \in \mathcal{I}} \mathcal{F}(\text{ac}_{P,i})$ where $\text{ac}_{P,i}$ is a nested application condition with $i \in \mathcal{I}$ for some index set \mathcal{I} .

Remark 11 (Satisfaction of Translated Nested Application Conditions [213]).

Consider an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$. A translated nested application condition $\mathcal{F}(\text{ac}_P)$ is a nested application condition over an object $\mathcal{F}(P)$. Hence the satisfaction relation is defined as follows:

- Every morphism in \mathbf{C}_2 satisfies $\mathcal{F}(\text{true})$.
- A morphism $p' : \mathcal{F}(P) \rightarrow G'$ satisfies a nested application condition $\exists(\mathcal{F}(a), \mathcal{F}(\text{ac}_C))$ where $\mathcal{F}(\text{ac}_C)$ is a nested application condition over an object $\mathcal{F}(C)$ if there is a morphism $q' : \mathcal{F}(C) \rightarrow G'$ in \mathcal{M}_2 such that $q' \circ \mathcal{F}(a) = p'$ and $q' \models \mathcal{F}(\text{ac}_C)$ (see Figure 28).
- $p' \models \neg \mathcal{F}(\text{ac}_P)$ means that $p' \not\models \mathcal{F}(\text{ac}_P)$.
- $p' \models \bigwedge_{i \in \mathcal{I}} \mathcal{F}(\text{ac}_{P,i})$ means that for all $i \in \mathcal{I}$ it holds that $p' \models \mathcal{F}(\text{ac}_{P,i})$.
- $p' \models \bigvee_{i \in \mathcal{I}} \mathcal{F}(\text{ac}_{P,i})$ means that for some $i \in \mathcal{I}$ it holds that $p' \models \mathcal{F}(\text{ac}_{P,i})$.

An example for a concrete translated nested application condition and its satisfaction is given in Example 14 in Figure 36 (see Section 6.1) where the NAC of the hypergraph transformation rule `enterServer` (see Figure 13) is translated by the \mathcal{M} -functor \mathcal{F}_{HG} introduced in Definition 63 leading to the NAC of the translated rule $\mathcal{F}_{\text{HG}}(\text{enterServer})$ depicted in Figure 39.

$$\begin{array}{ccc}
 \mathcal{F}(\text{ac}_P) \triangleright \mathcal{F}(P) & \xrightarrow{\mathcal{F}(a)} & \mathcal{F}(C) \triangleleft \mathcal{F}(\text{ac}_C) \\
 & \searrow \quad \swarrow & \\
 & \text{=} & \\
 p' \searrow & & \swarrow q' \in \mathcal{M}_2 \\
 & G' &
 \end{array}$$

Figure 28: Diagram for satisfaction of a nested application condition $\mathcal{F}(\text{ac}_P) = \exists(\mathcal{F}(a), \mathcal{F}(\text{ac}_C))$.

The following lemma states that we can translate and create the satisfaction of nested application conditions using \mathcal{M} -functors. This is important for the translation and creation of direct transformations since the nested application conditions related to the rules of the source transformation system must restrict the rule applications equivalently to the corresponding translated rules in the target transformation system.

Lemma 11 (Satisfaction of Nested Application Conditions [213]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (\mathcal{M}) -morphisms⁵. Then a morphism $p : P \rightarrow G$ satisfies a nested application condition ac_P in AS_1 iff the corresponding morphism $\mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G)$ satisfies the nested application condition $\mathcal{F}(\text{ac}_P)$ in AS_2 .

Proof.

This lemma can be proved by induction over the depth of a nested application condition. For detailed proof see [Appendix B](#) on page 276. \square

Now we can extend Theorem 1 to the translation and creation of transformations for rules with nested application conditions according to the following theorem. This extension allows for the functorial behavior transfer in this generalized setting.

Theorem 3 (Translation and Creation of Rule Applicability and (Direct) Transformations for Rules with Nested Application Conditions).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$ and $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$ containing rules with nested application conditions. Each \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ translates rule applicability and (direct) transformations. Vice versa, \mathcal{F} creates rule applicability and (direct) transformations if $(\mathbf{C}_1, \mathcal{M}_1)$ has initial pushouts and \mathcal{F} creates (\mathcal{M}) -morphisms as well as preserves initial pushouts.

Proof.

- **Translation:**

- \mathcal{F} translates rule applicability:

Assume that a rule $\rho = (p = (L \xleftarrow{l} K \xrightarrow{r} R), \text{ac}_L) \in P$ is applicable to some object G in AS_1 at some general match morphism $m : L \rightarrow G$. This implies that there is a pushout complement D such that (1) is a pushout and m satisfies the nested application condition ac_L according to Definition 18. Since \mathcal{F} preserves pushouts along \mathcal{M} -morphisms, there is the pushout complement $\mathcal{F}(D)$ such that (2) is a pushout. Furthermore, according to Lemma 11, we have that the translated general match morphism $\mathcal{F}(m) : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$ satisfies the corresponding translated nested application condition $\mathcal{F}(\text{ac}_L)$. Hence, the rule $\mathcal{F}(\rho) = (\mathcal{F}(p) = (\mathcal{F}(L) \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R)), \mathcal{F}(\text{ac}_L))$ is applicable to $\mathcal{F}(G)$ at $\mathcal{F}(m)$.

⁵ \mathcal{F} creates (\mathcal{M}) -morphisms means that \mathcal{F} creates both morphisms and \mathcal{M} -morphisms. Note that if we formulate this lemma for a restricted \mathcal{M} -functor, it suffices to require that \mathcal{F}_R creates \mathcal{M} -morphisms only.

$\mathcal{F}(R)), \mathcal{F}(\text{ac}_L)) \in \mathcal{F}(P)$ with translated nested application condition $\mathcal{F}(\text{ac}_L)$ is applicable at the translated general match morphism $\mathcal{F}(m)$.

$$\begin{array}{ccc}
 \text{ac}_L \triangleright L & \xleftarrow{l} K \xrightarrow{r} R & \\
 m \downarrow & (1) \downarrow & \\
 G & \longleftarrow D &
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{ccc}
 \mathcal{F}(\text{ac}_L) \triangleright \mathcal{F}(L) & \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R) & \\
 \mathcal{F}(m) \downarrow & (2) \downarrow & \\
 \mathcal{F}(G) & \longleftarrow \mathcal{F}(D) &
 \end{array}$$

– \mathcal{F} translates direct transformations:

As the next step, we construct pushout (3) over the morphisms $k : K \rightarrow D$ and $r : K \rightarrow R$ leading together with pushout (1) to the direct transformation $G \xrightarrow{\rho, m} H$ in AS_1 . Since \mathcal{F} preserves pushouts along \mathcal{M} -morphisms, we have that (4) is a pushout as well. Hence, we get by DPO (2) + (4) the direct transformation $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(m)} \mathcal{F}(H)$ in AS_2 . Thus, we have by Definition 43 that \mathcal{F} translates direct transformations.

$$\begin{array}{ccc}
 \text{ac}_L \triangleright L & \xleftarrow{l} K \xrightarrow{r} R & \\
 m \downarrow & (1) \downarrow k \downarrow & (3) \downarrow \\
 G & \longleftarrow D \longrightarrow H &
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{ccc}
 \mathcal{F}(\text{ac}_L) \triangleright \mathcal{F}(L) & \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R) & \\
 \mathcal{F}(m) \downarrow & (2) \downarrow \mathcal{F}(k) \downarrow & (4) \downarrow \\
 \mathcal{F}(G) & \longleftarrow \mathcal{F}(D) \longrightarrow \mathcal{F}(H) &
 \end{array}$$

• **Creation:**

– \mathcal{F} creates rule applicability:

Assume that a translated rule $\mathcal{F}(\rho) = (\mathcal{F}(p) = (\mathcal{F}(L) \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R)), \mathcal{F}(\text{ac}_L)) \in \mathcal{F}(P)$ is applicable to some object $\mathcal{F}(G)$ in AS_2 at some general match morphism $m' : \mathcal{F}(L) \rightarrow \mathcal{F}(G)$. Since \mathcal{F} creates morphisms by assumption, we have a unique general match morphism $m : L \rightarrow G$ such that $\mathcal{F}(m) = m'$. Let (1) be an initial pushout over m in $(\mathcal{C}_1, \mathcal{M}_1)$. By assumption on \mathcal{F} , (2) is an initial pushout over $\mathcal{F}(m)$ and (4), (5) are pushouts in $(\mathcal{C}_2, \mathcal{M}_2)$. Then we have according to Definition 5 that there is a morphism $b'' : \mathcal{F}(B) \rightarrow \mathcal{F}(K)$ in \mathcal{M}_2 with $\mathcal{F}(l) \circ b'' = \mathcal{F}(b)$. Moreover, since $\mathcal{F}(\rho)$ is applicable to $\mathcal{F}(G)$ at the general match morphism m' , we know that m' satisfies the translated nested application condition $\mathcal{F}(\text{ac}_L)$ and D' is a pushout complement in pushout (4) (see Definition 18). Since \mathcal{F} creates morphisms by assumption, there is a unique morphism $b' : B \rightarrow K$ with $\mathcal{F}(b') = b''$. Moreover, the uniqueness of morphism creation implies that $l \circ b' = b$ and hence $b' \in \mathcal{M}_1$ by decomposition property of \mathcal{M}_1 . Therefore, the gluing condition for the plain case is satisfied and we have a pushout complement D in pushout (3). Now, applying Lemma 11, we obtain additionally that the general match morphism m satisfies the nested application condition ac_L , because \mathcal{F} creates (\mathcal{M}) -morphisms by assumption. Hence, using Definition 18, we have that the rule ρ is applicable to the object G at the general match morphism m , which implies that \mathcal{F} creates rule applicability.

$$\begin{array}{ccc}
 \begin{array}{ccccc}
 & b' & & & \\
 & \text{ac}_L & & & \\
 B & \xrightarrow{b} L & \xleftarrow{l} K & \xrightarrow{r} R & \\
 \downarrow & (1) \downarrow m \downarrow & \downarrow & & \\
 C & \longrightarrow G & \longleftarrow D & &
 \end{array} & \Leftarrow &
 \begin{array}{ccccc}
 & b'' & & & \\
 & \mathcal{F}(\text{ac}_L) & & & \\
 \mathcal{F}(B) & \xrightarrow{\mathcal{F}(b)} \mathcal{F}(L) & \xleftarrow{\mathcal{F}(l)} \mathcal{F}(K) & \xrightarrow{\mathcal{F}(r)} \mathcal{F}(R) & \\
 \downarrow & (2) \downarrow m' \downarrow & \downarrow & (5) \downarrow & \\
 \mathcal{F}(C) & \longrightarrow \mathcal{F}(G) & \longleftarrow D' & \longrightarrow H' &
 \end{array}
 \end{array}$$

– \mathcal{F} creates direct transformations:

Consider now the direct transformation $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(\mathfrak{m})} \mathcal{F}(H)$ in AS_2 given by DPO (4) + (5) with $\mathcal{F}(\mathfrak{m})$ and $\mathcal{F}(\rho)$ as considered before. We have already constructed pushout (3) in $(\mathbf{C}_1, \mathcal{M}_1)$ and can construct pushout (6) along the morphisms $k : K \rightarrow D$ and $r : K \rightarrow R$ leading to a direct transformation $G \xrightarrow{\rho, \mathfrak{m}} H$ with the general match morphism \mathfrak{m} . Since \mathcal{F} preserves pushouts along \mathcal{M} -morphisms and pushouts as well as pushout complements in $(\mathbf{C}_2, \mathcal{M}_2)$ are unique up to isomorphism, we have that $\mathcal{F}(D) \cong D'$, $\mathcal{F}(H) \cong H'$ leading to the direct transformation $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(\mathfrak{m})} \mathcal{F}(H)$ in AS_2 . Thus, we obtain that \mathcal{F} creates direct transformations using Definition 43 and the fact that the general match morphism \mathfrak{m} satisfies the nested application condition ac_L as shown in the previous part of the proof.

$$\begin{array}{ccc}
 \begin{array}{ccccc}
 & & b' & & \\
 & & \text{ac}_L & & \\
 & \swarrow & \nabla & \searrow & \\
 B & \xrightarrow{b} & L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
 \downarrow & (1) & \downarrow m & (3) & \downarrow k & (6) & \downarrow \\
 C & \longrightarrow & G & \longleftarrow & D & \longrightarrow & H
 \end{array}
 & \Longleftarrow &
 \begin{array}{ccccc}
 & & b'' & & \\
 & & \mathcal{F}(\text{ac}_L) & & \\
 & \swarrow & \nabla & \searrow & \\
 \mathcal{F}(B) & \xrightarrow{\mathcal{F}(b)} & \mathcal{F}(L) & \xleftarrow{\mathcal{F}(l)} & \mathcal{F}(K) & \xrightarrow{\mathcal{F}(r)} & \mathcal{F}(R) \\
 \downarrow & (2) & \downarrow m' & (4) & \downarrow \mathcal{F}(k) & (5) & \downarrow \\
 \mathcal{F}(C) & \longrightarrow & \mathcal{F}(G) & \longleftarrow & D' & \longrightarrow & H' \\
 & & & & \wr \parallel & & \wr \parallel \\
 & & & & \mathcal{F}(D) & & \mathcal{F}(H)
 \end{array}
 \end{array}$$

□

The result from Theorem 3 holds also for sequences of direct transformations. This can be easily shown by induction over the length of a transformation sequence where Theorem 3 is used to show the induction step.

The following remark summarizes how Theorem 3 can be applied to transformations with \mathcal{M} -match morphisms.

Remark 12 (Translation and Creation of Rule Applicability and (Direct) Transformations with \mathcal{M} -Match Morphisms for Rules with Nested Application Conditions).

Note that Theorem 3 is formulated and shown for the case of transformations with general match morphisms. Nevertheless, this theorem holds also for transformations with \mathcal{M} -match morphisms requiring the existence of initial pushouts in $(\mathbf{C}_1, \mathcal{M}_1)$ as well as the creation of \mathcal{M} -morphisms and preservation of initial pushouts over \mathcal{M} -morphisms by a restricted \mathcal{M} -functor instead of the assumptions listed in Theorem 3.

3.2 BISIMULATION-BASED BEHAVIOR ANALYSIS OF \mathcal{M} -ADHESIVE TRANSFORMATION SYSTEMS

In this section, we relate the behavioral equivalence in terms of Theorems 1, 2, and 3 from the previous section to the notion of bisimilarity [217]. In contrast to the approach introduced in the previous section, where we compared the multi-step behaviors of the two systems by comparing their possible sequences of direct transformations, the notion of bisimilarity compares additionally the interactive behavior of the involved systems by relating their enabled steps throughout the execution. We fill the gap between the approach introduced in the previous section and the notion of bisimilarity to enable further development of our theory and to emphasize the general applicability of our theory developed so far. We achieve this by, firstly, introducing the definition of bisimilarity in our framework of \mathcal{M} -adhesive transformation systems and by, secondly, relating the results from the previous section to the introduced notion of bisimilarity focusing on two application scenarios. We are interested in two different kinds of bisimilarity: inter-transformation-system bisimilarity and intra-transformation-system bisimilarity. For a given \mathcal{M} -functor \mathcal{F} , we speak of an inter-transformation-system bisimilarity if an object X of the source transformation system and its functor-translated counterpart, the object $\mathcal{F}(X)$, are behavioral equivalent (i.e., the \mathcal{M} -Functor \mathcal{F} induces a unique bisimulation between objects of the source transformation system and the corresponding \mathcal{F} -images contained in the target transformation system where transformation rules are related via \mathcal{F}). Moreover, considering a given \mathcal{M} -functor \mathcal{F} , we have an intra-transformation-system bisimilarity if it holds that two objects X and Y behave in the source transformation system equivalently if and only if their functor-translated counterparts $\mathcal{F}(X)$ and $\mathcal{F}(Y)$ behave equivalently also in the target transformation system.

In the following two subsections, we introduce the notion of bisimilarity in the context of \mathcal{M} -adhesive transformation systems without or with nested application conditions as well as characterize how bisimilarity is induced by means of an \mathcal{M} -functor (see Subsection 3.2.1) and how an \mathcal{M} -functor translates and creates bisimulations among the involved \mathcal{M} -adhesive transformation systems (see Subsection 3.2.2).

3.2.1 \mathcal{F} -Bisimilarity

In this subsection, we introduce the notions of R-simulations and R-bisimulations along with some basic properties, before formally verifying that \mathcal{M} -functors transfer the behavior of \mathcal{M} -adhesive transformation systems with respect to our notion of bisimulation.

We begin this subsection with the technical notion of isomorphism closure. We need this notion in the context of category theory where objects are related to each other up to isomorphism. Formally, this means that if two objects are in the relation A and additionally there are two objects isomorphic to the given objects, then the isomorphic objects are in the isomorphism closure of A .

Definition 48 (Isomorphism Closure).

Consider categories \mathbf{C}_1 and \mathbf{C}_2 and a binary relation $A \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$. Then the isomorphism closure of A is given by $\text{IC}(A) = \{(\overline{G}, \overline{G}') \mid \exists (G, G') \in A. G \cong \overline{G} \wedge G' \cong \overline{G}'\}$.

$$\begin{array}{ccc} \overline{G} & \xleftarrow{\text{IC}(A)} & \overline{G}' \\ \parallel & & \parallel \\ G & \xleftarrow{A} & G' \end{array}$$

We build on the well-known definitions of simulation and bisimulation from [217] to be able to compare the behavior of different \mathcal{M} -adhesive transformation systems without or with nested application conditions.

Definition 49 (R-Simulation, R-Similarity [217]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, P_2)$, a rule relation $R \subseteq P_1 \times P_2$, and objects $G_1 \in \text{Ob}_{\mathbf{C}_1}$, $G_2 \in \text{Ob}_{\mathbf{C}_2}$.

- A binary relation $S_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ is an *R-simulation* iff for each pair of objects $(G_1, G_2) \in S_R$ whenever a transformation step $G_1 \xrightarrow{\rho_1, m_1} H_1$ is possible in AS_1 with a rule $\rho_1 \in P_1$ and a match morphism m_1 , there is an object $H_2 \in \text{Ob}_{\mathbf{C}_2}$ such that a transformation step $G_2 \xrightarrow{\rho_2, m_2} H_2$ is possible in AS_2 and $(H_1, H_2) \in S_R$ with $\rho_2 \in P_2$, $(\rho_1, \rho_2) \in R$, and a match morphism m_2 .
- G_1 *R-simulates* G_2 , written $G_1 \preceq_R G_2$, iff there is an *R-simulation* $S_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ such that $(G_1, G_2) \in S_R$.
- The binary relation $\preceq_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ is also called *R-similarity*.

$$\begin{array}{ccc}
 & \rho_1 \xleftrightarrow{R} \rho_2 & \\
 & \xleftrightarrow{S_R} & \\
 G_1 & & G_2 \\
 \rho_1, m_1 \downarrow & & \downarrow \rho_2, m_2 \\
 H_1 & \xleftrightarrow{S_R} & H_2 \\
 & G_1 \preceq_R G_2 &
 \end{array}$$

Definition 50 (R-Bisimulation, R-Bisimilarity [217]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, P_2)$, and objects $G_1 \in \text{Ob}_{\mathbf{C}_1}$, $G_2 \in \text{Ob}_{\mathbf{C}_2}$.

- A binary relation $B_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ is called *R-bisimulation* iff B_R is an *R-simulation* and B_R^{-1} is an R^{-1} -simulation according to Definition 49.
- G_1 and G_2 are *R-bisimilar*, written $G_1 \sim_R G_2$, iff there is an *R-bisimulation* $B_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ such that $(G_1, G_2) \in B_R$.
- The binary relation $\sim_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ is also called *R-bisimilarity*.

As usual for category theory, we consider objects occurring in *R-simulations* resp. *R-bisimulations* up to isomorphism. This corresponds in general to the notion of *simulation/bisimulation up to isomorphism*.

Lemma 12 (R-Simulations are Closed under Isomorphism Closure).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, P_2)$, a rule relation $R \subseteq P_1 \times P_2$, and an *R-simulation* $S_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$. Then the isomorphism closure $\text{IC}(S_R) \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ of S_R is an *R-simulation*.

Proof.

The detailed proof for this lemma is given in [Appendix B](#) on [page 279](#). □

Lemma 13 (R-Bisimulations are Closed under Isomorphism Closure).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, P_2)$, a rule relation $R \subseteq P_1 \times P_2$, and an R -bisimulation $B_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$. Then the isomorphism closure $IC(B_R) \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ of B_R is an R -bisimulation.

Proof.

The detailed proof for this lemma is given in [Appendix B](#) on [page 280](#). \square

When considering \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ between the underlying \mathcal{M} -adhesive categories in this thesis, the rule relation R from [Definition 49](#) identifies the rules according to the functorial translation, i.e., $R = \{(\rho, \mathcal{F}(\rho)) \mid \rho \in P\}$. Thus, we use in this situation the terms \mathcal{F} -simulation (\mathcal{F} -similarity) resp. \mathcal{F} -bisimulation (\mathcal{F} -bisimilarity) instead of R -simulation (R -similarity) resp. R -bisimulation (R -bisimilarity).

The following theorem states for transformation systems without or with nested application conditions that every transformation step in the source transformation system of an \mathcal{M} -functor can be \mathcal{F} -simulated by the corresponding translated transformation step in the target transformation system and vice versa using an \mathcal{F} -bisimulation. This \mathcal{F} -bisimulation corresponds to the notion of the inter-transformation-system bisimulation discussed before relating objects from two different \mathcal{M} -adhesive transformation systems and ensures the behavioral equivalence of the functor-related parts of the involved transformation systems.

Theorem 4 (\mathcal{F} -Bisimilarity of \mathcal{M} -Adhesive Transformation Systems).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (direct) transformations according to [Theorem 3](#) and (\mathcal{M}) -morphisms⁶. Then for every object $G \in \text{Ob}_{\mathbf{C}_1}$ in AS_1 it holds that G is \mathcal{F} -bisimilar to the corresponding object $\mathcal{F}(G) \in \text{Ob}_{\mathbf{C}_2}$ in AS_2 , written $G \sim_{\mathcal{F}} \mathcal{F}(G)$.

Proof.

We have to show that $G \sim_{\mathcal{F}} \mathcal{F}(G)$, which means according to [Definition 50](#) that there is a relation $B_{\mathcal{F}} \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ such that $B_{\mathcal{F}}$ is an \mathcal{F} -simulation, $B_{\mathcal{F}}^{-1} \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_1}$ is an \mathcal{F}^{-1} -simulation, and $(G, \mathcal{F}(G)) \in B_{\mathcal{F}}$.

Define $B_{\mathcal{F}} = \{(G, \mathcal{F}(G)) \mid G \in \text{Ob}_{\mathbf{C}_1}\}$. Furthermore, according to the definition of $B_{\mathcal{F}}$, we obviously have that $(G, \mathcal{F}(G)) \in B_{\mathcal{F}}$. It remains to show that $B_{\mathcal{F}}$ and $B_{\mathcal{F}}^{-1}$ are \mathcal{F} - resp. \mathcal{F}^{-1} -simulations.

- **Part 1 ($B_{\mathcal{F}}$ is an \mathcal{F} -simulation):**

Assume that $(G, \mathcal{F}(G)) \in B_{\mathcal{F}}$. According to [Definition 49](#), $(G, \mathcal{F}(G)) \in B_{\mathcal{F}}$ must imply that

$$\forall H \in \text{Ob}_{\mathbf{C}_1}, \rho \in P, m : L \rightarrow G. ((G \xrightarrow{\rho, m} H)$$

$$\Rightarrow (\exists H' \in \text{Ob}_{\mathbf{C}_2}, m' : \mathcal{F}(L) \rightarrow \mathcal{F}(G). (\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), m'} H' \wedge (H, H') \in B_{\mathcal{F}}))).$$

Assume the premise of the statement and fix $H \in \text{Ob}_{\mathbf{C}_1}$, $\rho \in P$, $m \in \text{Mor}_{\mathbf{C}_1}$. Since \mathcal{F} preserves commuting diagrams by general functor property and pushouts along \mathcal{M} -morphisms by the \mathcal{M} -functor property, we have the transformation step $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(m)} \mathcal{F}(H)$ in AS_2 (for the case if the considered transformation step in AS_1 contains a nested application

⁶ Note that if we formulate this theorem for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R creates (direct) transformations according to [Remark 12](#) and \mathcal{M} -morphisms according to [Definition 45](#). The proof for the adapted theorem works analogously to that of the current theorem.

condition ac_L satisfied by the match morphism \mathfrak{m} by assumption, we additionally obtain by Lemma 11 that also the translated match morphism $\mathcal{F}(\mathfrak{m})$ satisfies the corresponding translated nested application condition $\mathcal{F}(\text{ac}_L)$. Thus, there are an object $H' = \mathcal{F}(H)$ with the transformation step $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(\mathfrak{m})} \mathcal{F}(H)$ and the match morphism $\mathfrak{m}' = \mathcal{F}(\mathfrak{m})$. Furthermore, $(H, \mathcal{F}(H)) \in B_{\mathcal{F}}$ according to the definition of $B_{\mathcal{F}}$ given above. This implies that $B_{\mathcal{F}}$ is an \mathcal{F} -simulation.

$$\begin{array}{ccc} G & \xleftarrow{B_{\mathcal{F}}} & \mathcal{F}(G) \\ \rho, \mathfrak{m} \downarrow & & \downarrow \mathcal{F}(\rho), \mathfrak{m}' = \mathcal{F}(\mathfrak{m}) \\ H & \xleftarrow{B_{\mathcal{F}}} & H' = \mathcal{F}(H) \end{array}$$

• **Part 2 ($B_{\mathcal{F}}^{-1}$ is an \mathcal{F}^{-1} -simulation):**

Assume that $(\mathcal{F}(G), G) \in B_{\mathcal{F}}^{-1}$. According to Definition 49, $(\mathcal{F}(G), G) \in B_{\mathcal{F}}^{-1}$ must imply that

$$\begin{aligned} & \forall H' \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho) \in \mathcal{F}(P), \mathfrak{m}' : \mathcal{F}(L) \rightarrow \mathcal{F}(G). ((\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathfrak{m}'} H')) \\ & \Rightarrow (\exists H \in \text{Ob}_{\mathbf{C}_1}, \mathfrak{m} : L \rightarrow G. (G \xrightarrow{\rho, \mathfrak{m}} H \wedge (H', H) \in B_{\mathcal{F}}^{-1}))). \end{aligned}$$

Assume the premise of the statement and fix $H' \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho) \in \mathcal{F}(P), \mathfrak{m}' \in \text{Mor}_{\mathbf{C}_2}$. Since \mathcal{F} creates morphisms by assumption, we have the unique match morphism $\mathfrak{m} : L \rightarrow G$ such that $\mathcal{F}(\mathfrak{m}) = \mathfrak{m}'$. Since \mathcal{F} creates (direct) transformations by assumption, the corresponding rule $\rho \in P$ is applicable to the match morphism \mathfrak{m} leading to the transformation step $G \xrightarrow{\rho, \mathfrak{m}} H$ in AS_1 such that it additionally holds that $\mathcal{F}(H) \cong H'$. This fact implies that there exists the transformation step $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(\mathfrak{m})} \mathcal{F}(H)$ in AS_2 , because transformation steps are unique up to isomorphism (for the case if the considered transformation step in AS_2 contains a nested application condition $\mathcal{F}(\text{ac}_L)$ satisfied by the match morphism $\mathcal{F}(\mathfrak{m}) = \mathfrak{m}'$ by assumption, we additionally obtain by Lemma 11 that also the match morphism \mathfrak{m} satisfies the corresponding nested application condition ac_L). Furthermore, $(\mathcal{F}(H), H) \in B_{\mathcal{F}}^{-1}$ according to the definition of $B_{\mathcal{F}}$ given above. This implies that $B_{\mathcal{F}}^{-1}$ is an \mathcal{F}^{-1} -simulation.

$$\begin{array}{ccc} \mathcal{F}(G) & \xleftarrow{B_{\mathcal{F}}^{-1}} & G \\ \mathcal{F}(\rho), \mathfrak{m}' = \mathcal{F}(\mathfrak{m}) \downarrow & & \downarrow \rho, \mathfrak{m} \\ H' \cong \mathcal{F}(H) & \xleftarrow{B_{\mathcal{F}}^{-1}} & H \end{array}$$

□

3.2.2 \mathcal{F} -Transfer of Bisimilarity

In this subsection, we consider how we can relate intra-transformation-system bisimulations, i.e., bisimulations on $\text{Ob}_{\mathbf{C}_1}$ -objects with bisimulations on $\text{Ob}_{\mathbf{C}_2}$ -objects using \mathcal{M} -functors. In the previous subsection, we considered bisimulations between objects of different categories without transferring them, whereas we focus on translation and creation of bisimulations in this subsection. The translation and creation of a bisimulation using an \mathcal{M} -functor is only possible if the objects related in the bisimulation belong to the same category.

As the first step, we show that we can translate R-bisimilarity of $\text{Ob}_{\mathbf{C}_1}$ -objects using an \mathcal{M} -functor \mathcal{F} into $\mathcal{F}(\mathbf{R})$ -bisimilarity of the corresponding translated $\text{Ob}_{\mathbf{C}_2}$ -objects, and vice versa, we can create R-bisimilarity of $\text{Ob}_{\mathbf{C}_1}$ -objects from the corresponding $\mathcal{F}(\mathbf{R})$ -bisimilarity of the translated $\text{Ob}_{\mathbf{C}_2}$ -objects.

Lemma 14 (\mathcal{F} -Image Restricted \mathcal{F} -Transfer of R-Bisimilarity).

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $\text{AS}_2 = (\mathbf{C}_1, \mathcal{M}_1, P_2)$, $\text{AS}_3 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P_1))$, $\text{AS}_4 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P_2))$, rule relation $\mathbf{R} \subseteq P_1 \times P_2$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (direct) transformations according to Theorem 3 and (\mathcal{M}) -morphisms⁷. Then \mathcal{F} translates and creates \mathcal{F} -image restricted R-bisimilarity of objects, i.e., $B_{\mathbf{R}} \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_1}$ is an R-bisimulation iff $\mathcal{F}(B_{\mathbf{R}}) \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ is an $\mathcal{F}(\mathbf{R})$ -bisimulation.

Proof.

The detailed proof for this lemma is given in [Appendix B](#) on page 280. \square

As the second step, we state in the following lemma that $\mathcal{F}(\mathbf{R})$ -bisimulations may be reduced to smaller $\mathcal{F}(\mathbf{R})$ -bisimulations by removing all non- \mathcal{F} -images. This reduction is possible, because non- \mathcal{F} -images cannot be reached from \mathcal{F} -images applying translated transformation rules.

Lemma 15 (R-Bisimulation is Preserved under \mathcal{F} -Image Restriction).

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $\text{AS}_2 = (\mathbf{C}_1, \mathcal{M}_1, P_2)$, $\text{AS}_3 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P_1))$, $\text{AS}_4 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P_2))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (direct) transformations according to Theorem 3 and (\mathcal{M}) -morphisms⁷. Consider furthermore a rule relation $\mathbf{R} \subseteq P_1 \times P_2$ as well as an $\mathcal{F}(\mathbf{R})$ -bisimulation $B'_{\mathcal{F}(\mathbf{R})} \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ that is closed under isomorphisms, i.e., $\text{IC}(B'_{\mathcal{F}(\mathbf{R})}) = B'_{\mathcal{F}(\mathbf{R})}$. Then $\bar{B}_{\mathcal{F}(\mathbf{R})} \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ with $\bar{B}_{\mathcal{F}(\mathbf{R})} = B'_{\mathcal{F}(\mathbf{R})} \cap (\mathcal{F}(\text{Ob}_{\mathbf{C}_1}) \times \mathcal{F}(\text{Ob}_{\mathbf{C}_1}))$ is also an $\mathcal{F}(\mathbf{R})$ -bisimulation.

Proof.

The detailed proof for this lemma is given in [Appendix B](#) on page 283. \square

Based on the two previous lemmas, we state that bisimilarity of $\text{Ob}_{\mathbf{C}_1}$ -objects, on the one hand, and $\text{Ob}_{\mathbf{C}_2}$ -objects, on the other hand, can be translated and created using \mathcal{M} -functors.

Theorem 5 (\mathcal{F} -Transfer of R-Bisimilarity).

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $\text{AS}_2 = (\mathbf{C}_1, \mathcal{M}_1, P_2)$, $\text{AS}_3 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P_1))$, $\text{AS}_4 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P_2))$, a rule relation $\mathbf{R} \subseteq P_1 \times P_2$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (direct) transformations according to Theorem 3 and (\mathcal{M}) -morphisms⁷. Then for arbitrary objects $G, G' \in \text{Ob}_{\mathbf{C}_1}$ it holds that G is R-bisimilar to G' iff $\mathcal{F}(G)$ is $\mathcal{F}(\mathbf{R})$ -bisimilar to $\mathcal{F}(G')$, written $(G \sim_{\mathbf{R}} G') \Leftrightarrow (\mathcal{F}(G) \sim_{\mathcal{F}(\mathbf{R})} \mathcal{F}(G'))$.

Proof.

- (\Rightarrow) :

Fix $G, G' \in \text{Ob}_{\mathbf{C}_1}$ and assume that $G \sim_{\mathbf{R}} G'$. This means according to Definition 50 that there is an R-bisimulation $B_{\mathbf{R}} \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_1}$ such that $(G, G') \in B_{\mathbf{R}}$. This implies by application of \mathcal{F} that $(\mathcal{F}(G), \mathcal{F}(G')) \in \mathcal{F}(B_{\mathbf{R}})$ and $\mathcal{F}(B_{\mathbf{R}}) \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ is an $\mathcal{F}(\mathbf{R})$ -

⁷ Note that if we formulate this lemma (theorem) for a restricted \mathcal{M} -functor, we require that $\mathcal{F}_{\mathbf{R}}$ creates (direct) transformations according to Remark 12 and \mathcal{M} -morphisms according to Definition 45. The proof for the adapted lemma (theorem) works analogously to that of the current lemma (theorem).

bisimulation according to Lemma 14. Now applying Definition 50 again, we obtain that $\mathcal{F}(G) \sim_{\mathcal{F}(R)} \mathcal{F}(G')$.

• (\Leftarrow) :

Fix $G, G' \in \text{Ob}_{\mathbf{C}_1}$ and assume that $\mathcal{F}(G) \sim_{\mathcal{F}(R)} \mathcal{F}(G')$. According to Definition 50, we have that there is an $\mathcal{F}(R)$ -bisimulation $B'_{\mathcal{F}(R)} \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ such that $(\mathcal{F}(G), \mathcal{F}(G')) \in B'_{\mathcal{F}(R)}$. But it holds as well that $(\mathcal{F}(G), \mathcal{F}(G')) \in \bar{B}_{\mathcal{F}(R)}$ with $\bar{B}_{\mathcal{F}(R)} = \text{IC}(B'_{\mathcal{F}(R)}) \cap (\mathcal{F}(\text{Ob}_{\mathbf{C}_1}) \times \mathcal{F}(\text{Ob}_{\mathbf{C}_1}))$ where $\text{IC}(B'_{\mathcal{F}(R)})^8$ is the isomorphism closure of $B'_{\mathcal{F}(R)}$ according to Definition 48 and we have that $\bar{B}_{\mathcal{F}(R)}$ is an $\mathcal{F}(R)$ -bisimulation according to Lemmas 13 and 15. Since the pairs in $\bar{B}_{\mathcal{F}(R)}$ contain only \mathcal{F} -images, we have that there is the R -bisimulation $B_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_1}$ with $B_R = \{(G, G') \mid (\mathcal{F}(G), \mathcal{F}(G')) \in \bar{B}_{\mathcal{F}(R)}\}$ such that $\mathcal{F}(B_R) = \bar{B}_{\mathcal{F}(R)}$ by application of Lemma 14. Since $(\mathcal{F}(G), \mathcal{F}(G')) \in \mathcal{F}(B_R)$, we know that there are $\underline{G}, \underline{G}' \in \text{Ob}_{\mathbf{C}_1}$ such that $(\mathcal{F}(G), \mathcal{F}(G')) = (\mathcal{F}(\underline{G}), \mathcal{F}(\underline{G}'))$ and $(\underline{G}, \underline{G}') \in B_R$. Thus, it obviously holds that $\mathcal{F}(G) = \mathcal{F}(\underline{G})$ and $\mathcal{F}(G') = \mathcal{F}(\underline{G}')$. Using now the fact that \mathcal{F} creates morphisms (and thus also identities implying the injectivity of \mathcal{F} on objects according to Lemmas 8 and 9) by assumption, we obtain that also $G = \underline{G}$ and $G' = \underline{G}'$ implying that $(G, G') \in B_R$. Hence, by application of Definition 50 we have that $G \sim_R G'$. \square

We refer the readers interested in an example for the introduced concepts to Section 6.3 where we give an example demonstrating \mathcal{F}_{HG} -bisimilarity and \mathcal{F}_{HG} -transfer of R -bisimilarity for concrete hypergraph transformation systems.

8 Isomorphism closure $\text{IC}(B'_{\mathcal{F}(R)})$ of $B'_{\mathcal{F}(R)}$ guarantees here that $\bar{B}_{\mathcal{F}(R)}$ is an $\mathcal{F}(R)$ -bisimulation by Lemma 15.

CONFLUENCE ANALYSIS OF \mathcal{M} -ADHESIVE TRANSFORMATION SYSTEMS USING \mathcal{M} -FUNCTORS

In this chapter, we address the theoretical framework that allows us to analyze local confluence and termination of \mathcal{M} -adhesive transformation systems and hence also their confluence. In particular, we want to discuss under which requirements local confluence and termination can be translated by an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ from a transformation system $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$ to another one $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$ with translated productions $\mathcal{F}(P)$ and, vice versa, under which requirements local confluence and termination of AS_1 can be created from the local confluence and termination of AS_2 using \mathcal{F} . Considering both directions, we speak of the \mathcal{F} -transfer of local confluence and termination leading altogether to the \mathcal{F} -transfer of confluence.

We discuss in Section 4.1 the \mathcal{F} -transfer of local confluence for the case of transformations without nested application conditions, called in the following the *plain case*, while the translation and creation of local confluence with nested application conditions is then presented in Section 4.2. Finally, in Section 4.3, we introduce the analysis approach for termination and confluence of \mathcal{M} -adhesive transformation systems (implying also their functional behavior) using the functorial transfer of these semantical properties.

4.1 \mathcal{F} -TRANSFER OF LOCAL CONFLUENCE FOR TRANSFORMATIONS WITHOUT NESTED APPLICATION CONDITIONS

As already described in Subsubsection 2.2.2.4, a transformation system is locally confluent if starting with some object G whenever we can transform G into the objects H_1 and H_2 applying some rules ρ_1 and ρ_2 , respectively, we can subsequently transform these two objects in one or more steps into the same object X .

In the following theorem, we show that the local confluence of a target transformation system can be deduced from the local confluence of the source transformation system and, vice versa, using \mathcal{M} -functors.

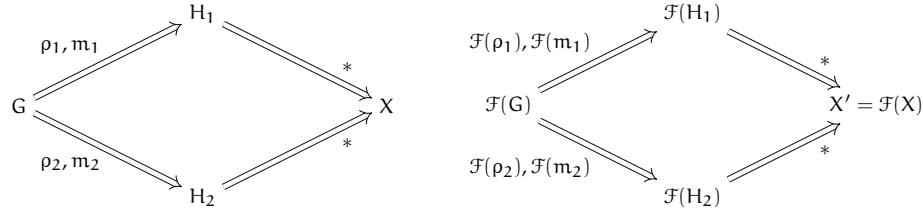
Theorem 6 (Translation and Creation of Local Confluence for Transformations without Nested Application Conditions [209]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that translates and creates (direct) transformations according to Theorem 1 as well as creates morphisms¹. Then AS_1 is locally confluent for all transformation spans $H_1 \xrightarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ iff AS_2 is locally confluent for all translated transformation spans $\mathcal{F}(H_1) \xrightarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$.

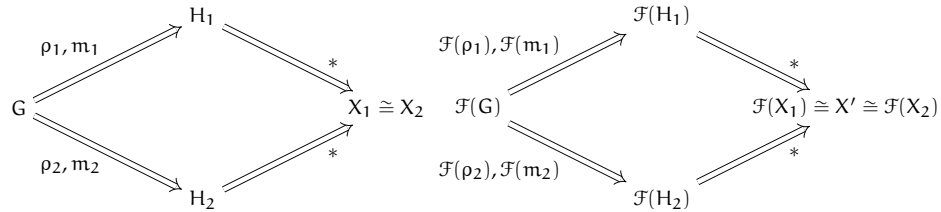
¹ Note that if we formulate this theorem for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R translates and creates (direct) transformations according to Theorem 2 and creates \mathcal{M} -morphisms according to Definition 45. The proof for the adapted theorem works analogously to that of the current theorem.

Proof.• **Translation:**

Assume local confluence of $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ in AS_1 . Then there exist an object X and transformations $H_1 \xrightarrow{*} X$, $H_2 \xrightarrow{*} X$ via P . Due to the assumption that \mathcal{F} translates (direct) transformations, there exist the object $X' = \mathcal{F}(X)$ and transformations $\mathcal{F}(H_1) \xrightarrow{*} \mathcal{F}(X)$, $\mathcal{F}(H_2) \xrightarrow{*} \mathcal{F}(X)$ via $\mathcal{F}(P)$. Hence, the translated transformation span $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$ is locally confluent in AS_2 .

• **Creation:**

Assume local confluence of $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$ in AS_2 . Then there exist an object X' and transformations $\mathcal{F}(H_1) \xrightarrow{*} X'$, $\mathcal{F}(H_2) \xrightarrow{*} X'$ via $\mathcal{F}(P)$. Due to the assumption that \mathcal{F} creates (direct) transformations, there exist objects X_1 , X_2 and transformations $H_1 \xrightarrow{*} X_1$, $H_2 \xrightarrow{*} X_2$ via P with $\mathcal{F}(X_1) \cong X' \cong \mathcal{F}(X_2)$. Thus, it holds that $\mathcal{F}(X_1) \cong \mathcal{F}(X_2)$ and the assumption that \mathcal{F} creates morphisms (and hence also isomorphisms) implies that $X_1 \cong X_2$. Therefore, we obtain that $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ is locally confluent in AS_1 .



□

The theorem above requires for the creation part that all *translated* transformation spans are locally confluent. However, the current tool support by AGG *only* tackles the problem of determining whether *all* possible transformation spans of the considered transformation system are locally confluent. In the rest of this section, we are describing a procedure, which is capable of determining local confluence for translated transformation spans only. This procedure is built along the lines of the standard analysis approach and consists of testing whether all feasible critical pairs of a transformation system are strictly confluent.

An obvious sufficient condition for local confluence is the parallel independence of all pairs of direct transformation steps. The parallel independence together with the sequential independence, which is related to parallel independence according to the Local Church-Rosser Theorem, can be translated and created by an \mathcal{M} -functor as follows.

Theorem 7 (Translation and Creation of Parallel and Sequential Independence of Transformations without Nested Application Conditions [207]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$ and $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$.

An \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ translates and creates parallel and sequential independence of transformations if \mathcal{F} creates morphisms².

Proof.

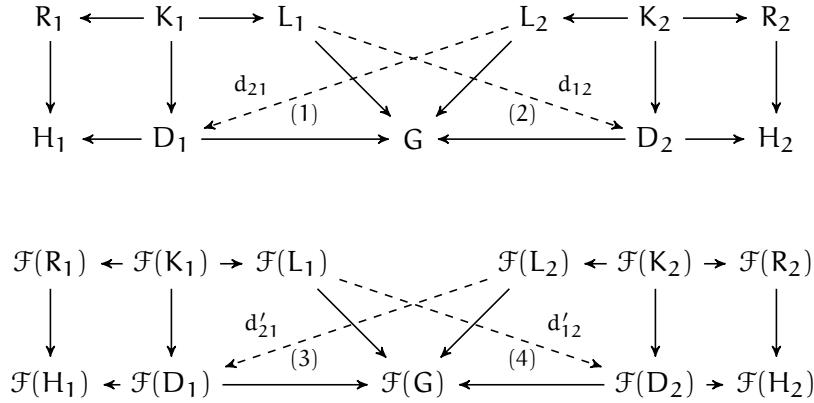
- \mathcal{F} translates parallel (sequential) independence of transformations:

Assume that direct transformations $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$ are parallel independent in AS_1 with the match morphisms $m_1 : L_1 \rightarrow G$, $m_2 : L_2 \rightarrow G$ and rules $\rho_1, \rho_2 \in P$. This means, according to Definition 9, that there are morphisms $d_{12} : L_1 \rightarrow D_2$, $d_{21} : L_2 \rightarrow D_1$ in $\text{Mor}_{\mathbf{C}_1}$ such that triangles (1) and (2) commute. Since \mathcal{F} preserves commuting diagrams by the general functor property, we have that triangles (3) and (4) commute as well, implying the parallel independence of the corresponding translated direct transformations in AS_2 by Definition 9. Thus, we have that \mathcal{F} translates parallel independence of transformations. The proof for the case of sequential independence works analogously.

- \mathcal{F} creates parallel (sequential) independence of transformations:

Consider two parallel independent transformations $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(H_1)$ and $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$ in AS_2 with the match morphisms $\mathcal{F}(m_1) : \mathcal{F}(L_1) \rightarrow \mathcal{F}(G)$, $\mathcal{F}(m_2) : \mathcal{F}(L_2) \rightarrow \mathcal{F}(G)$ and rules $\mathcal{F}(\rho_1), \mathcal{F}(\rho_2) \in \mathcal{F}(P)$. According to Definition 9, this means that we have morphisms $d'_{12} : \mathcal{F}(L_1) \rightarrow \mathcal{F}(D_2)$, $d'_{21} : \mathcal{F}(L_2) \rightarrow \mathcal{F}(D_1)$ in $\text{Mor}_{\mathbf{C}_2}$ such that triangles (3), (4) commute. This leads to the corresponding unique morphisms $d_{12} : L_1 \rightarrow D_2$ and $d_{21} : L_2 \rightarrow D_1$ in $\text{Mor}_{\mathbf{C}_1}$ making triangles (1) and (2) commute, because \mathcal{F} creates morphisms uniquely and preserves composition. Thus, according to Definition 9, we have parallel independence of transformations $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$ in AS_1 with the match morphisms m_1 and m_2 .

The proof for the case of sequential independence works analogously.



□

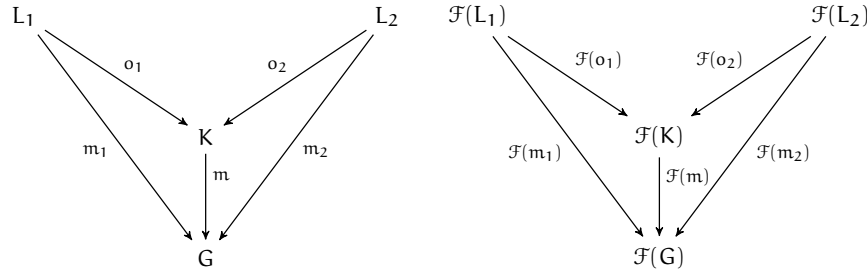
As the next step, we introduce an additional requirement, namely the compatibility of an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ with pair factorization, needed for one of our main conceptual results concerning the functorial transfer of local confluence for transformation systems with possibly parallel dependent transformation steps. This requirement states that $(\mathbf{C}_i, \mathcal{M}_i)$ has $\mathcal{E}'_i - \mathcal{M}_i$ pair factorization based on $\mathcal{E}_i - \mathcal{M}_i$ -factorization for $i \in \{1, 2\}$ as well as that \mathcal{F} preserves $\mathcal{E}' - \mathcal{M}$ pair factorization. In more detail, $(\mathbf{C}_i, \mathcal{M}_i)$ has $\mathcal{E}'_i - \mathcal{M}_i$ pair factorization if each morphism pair $(m_1 : L_1 \rightarrow G, m_2 : L_2 \rightarrow G)$

² Note that if we formulate this theorem for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R creates \mathcal{M} -morphisms as given in Definition 45. The proof for the adapted theorem works analogously to that of the current theorem.

with the common codomain can be decomposed uniquely up to isomorphism to $(m_1 = m \circ o_1, m_2 = m \circ o_2)$ with a pair (o_1, o_2) of jointly epimorphic morphisms and $m \in \mathcal{M}_i$. The formal definition for the compatibility of an \mathcal{M} -functor with pair factorization is given in the following.

Definition 51 (*\mathcal{F} is Compatible with Pair Factorization [210]*).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$. Then we say that \mathcal{F} is compatible with pair factorization if $(\mathbf{C}_1, \mathcal{M}_1)$ has $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization, $(\mathbf{C}_2, \mathcal{M}_2)$ has $\mathcal{E}'_2 - \mathcal{M}_2$ pair factorization, and \mathcal{F} preserves $\mathcal{E}' - \mathcal{M}$ pair factorization, i.e., for each $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization $(m_1 = m \circ o_1, m_2 = m \circ o_2)$ in $(\mathbf{C}_1, \mathcal{M}_1)$ also $(\mathcal{F}(m_1) = \mathcal{F}(m) \circ \mathcal{F}(o_1), \mathcal{F}(m_2) = \mathcal{F}(m) \circ \mathcal{F}(o_2))$ is an $\mathcal{E}'_2 - \mathcal{M}_2$ pair factorization in $(\mathbf{C}_2, \mathcal{M}_2)$ (see the diagram below).



For the case if $\mathcal{M}'_1 \not\subseteq \mathcal{M}_1$, we have to require in Definition 51 an additional $\mathcal{M}_1 - \mathcal{M}'_1$ pushout-pullback decomposition property (see Definition 72) for AS_1 . But note that this case does not apply in our applications.

The following lemma lists the properties that are required for the preservation of an $\mathcal{E}' - \mathcal{M}$ pair factorization by an \mathcal{M} -functor.

Lemma 16 (*\mathcal{F} Preserves $\mathcal{E}' - \mathcal{M}$ Pair Factorization*).

Consider \mathcal{M} -adhesive categories $(\mathbf{C}_i, \mathcal{M}_i)$ with binary coproducts and $\mathcal{E}_i - \mathcal{M}_i$ -factorizations for $i \in \{1, 2\}$ as well as an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$. Then \mathcal{F} preserves $\mathcal{E}' - \mathcal{M}$ pair factorization based on an $\mathcal{E} - \mathcal{M}$ -factorization if \mathcal{F} preserves coproducts and epimorphisms.

Proof.

The detailed proof for this lemma is given in [Appendix B on page 284](#). □

The following remark captures the special cases for the usage of compatibility of \mathcal{M} -functors with pair factorization in the context of this thesis.

Remark 13 (*Compatibility of Restricted \mathcal{M} -Functors with Pair Factorization*).

Note that in Sections 10.1 and 12.2, we use Definition 51 and Lemmas 1, 16 also for restricted \mathcal{M} -functors as well as for restricted functors defined between the categories of transformation systems, which are not necessarily \mathcal{M} -adhesive.

In general, we do not have to analyze all possible critical pairs of $\mathcal{F}(P)$ in the target transformation system, but just those, which are images of transformation steps in the source transformation system. We call this special kind of critical pairs *\mathcal{F} -reachable critical pairs*. Considering \mathcal{F} -reachable critical pairs only simplifies the analysis, because, in general, there are possibly critical pairs of $\mathcal{F}(P)$, which are not \mathcal{F} -reachable. Intuitively,

an \mathcal{F} -reachable critical pair is a critical pair in the target transformation system where all objects and morphisms are \mathcal{F} -images.

Definition 52 (\mathcal{F} -Reachable Critical Pair [209]).

Consider an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$. An \mathcal{F} -reachable critical pair of rules $\mathcal{F}(\rho_1)$ and $\mathcal{F}(\rho_2)$ is a critical pair in AS_2 of the form

$$\begin{array}{ccccccc}
 \mathcal{F}(R_1) & \xleftarrow{\mathcal{F}(r_1)} & \mathcal{F}(K_1) & \xrightarrow{\mathcal{F}(l_1)} & \mathcal{F}(L_1) & & \mathcal{F}(L_2) \xleftarrow{\mathcal{F}(l_2)} \mathcal{F}(K_2) \xrightarrow{\mathcal{F}(r_2)} \mathcal{F}(R_2) \\
 \downarrow & & \downarrow & & \searrow \mathcal{F}(o_1) & \swarrow \mathcal{F}(o_2) & \downarrow \\
 \mathcal{F}(P_1) & \xleftarrow{\mathcal{F}(w_1)} & \mathcal{F}(N_1) & \xrightarrow{\mathcal{F}(v_1)} & \mathcal{F}(K) & \xleftarrow{\mathcal{F}(v_2)} & \mathcal{F}(N_2) \xrightarrow{\mathcal{F}(w_2)} \mathcal{F}(P_2)
 \end{array}$$

where all morphisms of the type $\mathcal{F}(A) \rightarrow \mathcal{F}(B)$ have the form $\mathcal{F}(f)$ for some $f : A \rightarrow B$.

An example for an \mathcal{F} -reachable critical pair in the context of the concrete \mathcal{M} -functor \mathcal{F}_{HG} , translating hypergraphs into the corresponding typed attributed graphs according to Definition 63, is given in Figure 56 in Section 7.3.

We use the following lemma to ensure that a critical pair of translated transformation steps is \mathcal{F} -reachable. Intuitively, we already have an \mathcal{F} -reachable critical pair if we know that the overlapping of the left-hand sides of the translated rules is an \mathcal{F} -image.

Lemma 17 (\mathcal{F} -Reachable Critical Pairs).

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $\text{AS}_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (direct) transformations according to Theorem 1 and morphisms³. Then each critical pair of rules $\mathcal{F}(\rho_1)$ and $\mathcal{F}(\rho_2)$ with overlapping $\bar{K} \cong \mathcal{F}(K)$ in AS_2 is already \mathcal{F} -reachable (up to isomorphism).

$$\begin{array}{ccccccc}
 \mathcal{F}(R_1) & \xleftarrow{\mathcal{F}(r_1)} & \mathcal{F}(K_1) & \xrightarrow{\mathcal{F}(l_1)} & \mathcal{F}(L_1) & & \mathcal{F}(L_2) \xleftarrow{\mathcal{F}(l_2)} \mathcal{F}(K_2) \xrightarrow{\mathcal{F}(r_2)} \mathcal{F}(R_2) \\
 \downarrow & & \downarrow & & \searrow \bar{o}_1 & \swarrow \bar{o}_2 & \downarrow \\
 \bar{P}_1 & \xleftarrow{\bar{w}_1} & \bar{N}_1 & \xrightarrow{\bar{v}_1} & \bar{K} & \xleftarrow{\bar{v}_2} & \bar{N}_2 \xrightarrow{\bar{w}_2} \bar{P}_2
 \end{array}$$

Proof.

The detailed proof for this lemma is given in Appendix B on page 285. \square

The following lemma states that for each translated pair of parallel dependent transformation steps there is the corresponding embedded \mathcal{F} -reachable critical pair.

Lemma 18 (Completeness of \mathcal{F} -Reachable Critical Pairs).

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $\text{AS}_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that is compatible with pair factorization, and a translated parallel dependent transformation span $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$. Then there is an \mathcal{F} -reachable critical pair $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ of rules $\mathcal{F}(\rho_1)$, $\mathcal{F}(\rho_2)$ and an embedding given below.

³ Note that if we formulate this lemma for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R creates (direct) transformations according to Theorem 2 and \mathcal{M} -morphisms according to Definition 45. The proof for the adapted lemma works analogously to that of the current lemma.

$$\begin{array}{ccccc}
\mathcal{F}(P_1) & \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(\sigma_1)} & \mathcal{F}(K) & \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(\sigma_2)} & \mathcal{F}(P_2) \\
\downarrow & & \downarrow \mathcal{F}(m) & & \downarrow \\
\mathcal{F}(H_1) & \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} & \mathcal{F}(G) & \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} & \mathcal{F}(H_2)
\end{array}
\quad (1) \quad (2)$$

Proof.

The detailed proof for this lemma is given in [Appendix B](#) on [page 286](#). \square

Now we formulate and prove our next important theoretical result concerning the creation of local confluence based on \mathcal{F} -reachable critical pairs for the case of transformations without nested application conditions.

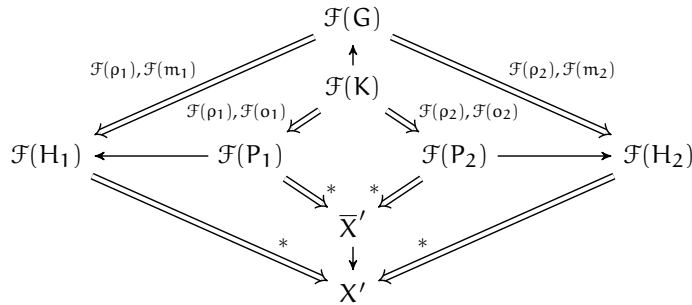
Theorem 8 (Creation of Local Confluence Based on \mathcal{F} -Reachable Critical Pairs for Rules without Nested Application Conditions [209, 213]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (direct) transformations according to Theorem 1 and morphisms as well as is compatible with pair factorization⁴. Then AS_1 is locally confluent for all transformation spans $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ if all \mathcal{F} -reachable critical pairs of $\mathcal{F}(\rho_1)$ and $\mathcal{F}(\rho_2)$ in AS_2 are strictly confluent.

Proof.

If two transformations $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$ are parallel independent then we have local confluence by Local Church-Rosser Theorem (see Fact 1).

Otherwise, we can construct for each pair of transformations $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ in AS_1 with a critical pair $P_1 \xleftarrow{\rho_1, \sigma_1} K \xrightarrow{\rho_2, \sigma_2} P_2$ the translated pair $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$ in AS_2 with the corresponding \mathcal{F} -reachable critical pair $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(\sigma_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(\sigma_2)} \mathcal{F}(P_2)$ using Lemma 18. This \mathcal{F} -reachable critical pair is by assumption strictly confluent leading to $\mathcal{F}(P_1) \xRightarrow{*} \bar{X}' \xRightarrow{*} \mathcal{F}(P_2)$ and the strict confluence implies the existence of X' in \mathbf{C}_2 with $\mathcal{F}(H_1) \xRightarrow{*} X' \xRightarrow{*} \mathcal{F}(H_2)$ using the proof of the Local Confluence Theorem (see Fact 1). This fact implies local confluence of $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$ and hence also that AS_2 is locally confluent for all translated transformation spans $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$. Now, using Theorem 6, we obtain that also AS_1 is locally confluent for all transformation spans $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$, which was to be shown.



\square

⁴ Note that if we formulate this theorem for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R creates (direct) transformations according to Theorem 2 and \mathcal{M} -morphisms according to Definition 45. The requirement concerning the compatibility of \mathcal{F} with pair factorization remains unchanged, but refers now to the restricted \mathcal{M} -functor. The proof for the adapted theorem works analogously to that of the current theorem.

4.2 \mathcal{F} -TRANSFER OF LOCAL CONFLUENCE FOR TRANSFORMATIONS WITH NESTED APPLICATION CONDITIONS

In this section, we focus on the development of theoretical results allowing us to verify on functorial way local confluence of \mathcal{M} -adhesive transformation systems containing rules with nested application conditions. As the first step, we extend several essential notions from Subsection 2.3.1 defining them on \mathcal{F} -images of an \mathcal{M} -functor. Subsequently, we introduce the notions of \mathcal{F} -reachable (weak) critical pairs and different kinds of compatibility of \mathcal{M} -functors with nested application conditions, which form the foundation for our intermediate results, like \mathcal{F} -transfer of strict confluence, preservation of \mathcal{F} -reachable (weak) critical pairs, compatibility of \mathcal{M} -functors with **Shift**- and **L**-transformations as well as dependencies between different kinds of compatibilities with nested application conditions, which altogether allow us to formulate and to prove our main theoretical result of this section concerning \mathcal{F} -transfer of local confluence for transformations with nested application conditions.

The first theorem, which we consider in this section, concerns, similarly to Section 4.1, how we can deduce on functorial way the local confluence of a target \mathcal{M} -adhesive transformation system from the local confluence of the source \mathcal{M} -adhesive transformation system and vice versa for the case of transformations with nested application conditions.

Theorem 9 (Translation and Creation of Local Confluence for Transformations with Nested Application Conditions).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that translates and creates (direct) transformations with nested application conditions according to Theorem 3 as well as creates (\mathcal{M} -)morphisms⁵. Then AS_1 is locally confluent for all transformation spans $H_1 \xrightarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ iff AS_2 is locally confluent for all translated transformation spans $\mathcal{F}(H_1) \xrightarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$.

Proof.

The proof for this theorem works analogously to the proof of Theorem 6. □

Similarly to Section 4.1, we describe in the following a procedure, which is capable of determining local confluence for the translated transformation spans with nested application conditions using the standard analysis approach introduced in [96] involving the strict confluence and AC-compatibility checks of all feasible critical pairs.

For the special case if all direct transformations of a considered \mathcal{M} -adhesive transformation system are pairwise parallel independent, we can verify the \mathcal{F} -transfer of local confluence using the Local Church-Rosser Theorem for transformations with nested application conditions (see Fact 4) together with the following theorem dealing amongst others with translation and creation of parallel independence of transformations. Also the sequential independence of transformations with nested application conditions, which is

⁵ Note that if we formulate this theorem for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R translates and creates (direct) transformations with nested application conditions according to Remark 12 and creates \mathcal{M} -morphisms according to Definition 45. The proof for the adapted theorem works analogously to that of the current theorem.

related to parallel independence according to the Local Church-Rosser Theorem, can be translated and created using \mathcal{M} -functors.

Theorem 10 (Translation and Creation of Parallel and Sequential Independence of Transformations with Nested Application Conditions).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$ where P are rules with nested application conditions. Then an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ translates and creates parallel and sequential independence of transformations if \mathcal{F} creates (\mathcal{M} -) morphisms⁶.

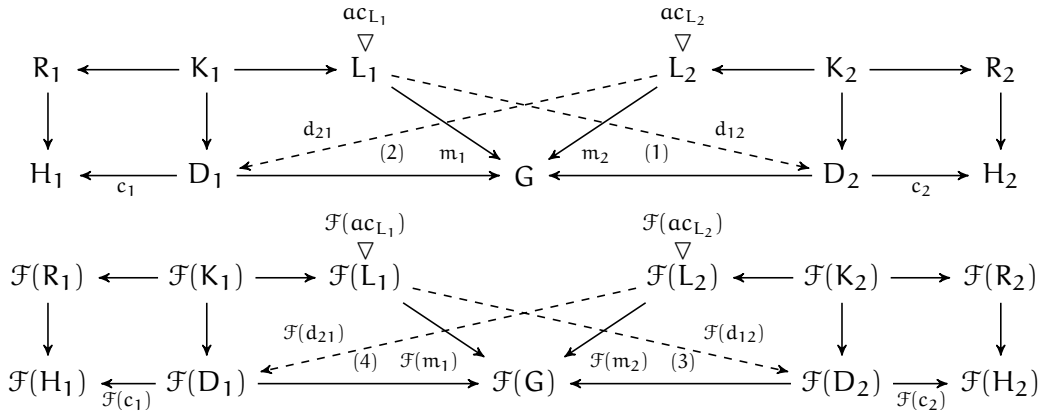
Proof.

- \mathcal{F} translates parallel (sequential) independence of transformations:

Assume that direct transformations $G \xRightarrow{\rho_1, m_1} H_1$ and $G \xRightarrow{\rho_2, m_2} H_2$ are parallel independent in AS_1 with the match morphisms $m_1 : L_1 \rightarrow G$, $m_2 : L_2 \rightarrow G$ and rules $\rho_1 = (p_1 = (L_1 \leftarrow K_1 \rightarrow R_1), ac_{L_1})$, $\rho_2 = (p_2 = (L_2 \leftarrow K_2 \rightarrow R_2), ac_{L_2})$. This means according to Definition 24 that there are the morphisms $d_{12} : L_1 \rightarrow D_2$, $d_{21} : L_2 \rightarrow D_1$ in $\text{Mor}_{\mathbf{C}_1}$ such that the triangles (1) and (2) commute and it holds that $c_2 \circ d_{12} \models ac_{L_1}$, $c_1 \circ d_{21} \models ac_{L_2}$. Since \mathcal{F} preserves commuting diagrams by the general functor property, we have that the triangles (3) and (4) commute as well. It remains to show that $\mathcal{F}(c_2) \circ \mathcal{F}(d_{12}) \models \mathcal{F}(ac_{L_1})$ and $\mathcal{F}(c_1) \circ \mathcal{F}(d_{21}) \models \mathcal{F}(ac_{L_2})$ for the translated nested application conditions $\mathcal{F}(ac_{L_1})$ and $\mathcal{F}(ac_{L_2})$. It holds the following:

$$\begin{aligned} & (c_2 \circ d_{12} \models ac_{L_1}) \wedge (c_1 \circ d_{21} \models ac_{L_2}) \\ & \xRightarrow{\text{Lem. 11}} (\mathcal{F}(c_2 \circ d_{12}) \models \mathcal{F}(ac_{L_1})) \wedge (\mathcal{F}(c_1 \circ d_{21}) \models \mathcal{F}(ac_{L_2})) \\ & \xRightarrow{\text{funct. prop.}} (\mathcal{F}(c_2) \circ \mathcal{F}(d_{12}) \models \mathcal{F}(ac_{L_1})) \wedge (\mathcal{F}(c_1) \circ \mathcal{F}(d_{21}) \models \mathcal{F}(ac_{L_2})) \end{aligned}$$

Thus, we have by Definition 24 that \mathcal{F} translates parallel independence of transformations. The proof for the sequential independence works analogously.



- \mathcal{F} creates parallel (sequential) independence of transformations:

Consider two parallel independent transformations $\mathcal{F}(G) \xRightarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(H_1)$ and $\mathcal{F}(G) \xRightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$ in AS_2 with the match morphisms $\mathcal{F}(m_1) : \mathcal{F}(L_1) \rightarrow \mathcal{F}(G)$, $\mathcal{F}(m_2) : \mathcal{F}(L_2) \rightarrow \mathcal{F}(G)$ and translated rules $\mathcal{F}(\rho_1) = (\mathcal{F}(p_1) = (\mathcal{F}(L_1) \leftarrow \mathcal{F}(K_1) \rightarrow$

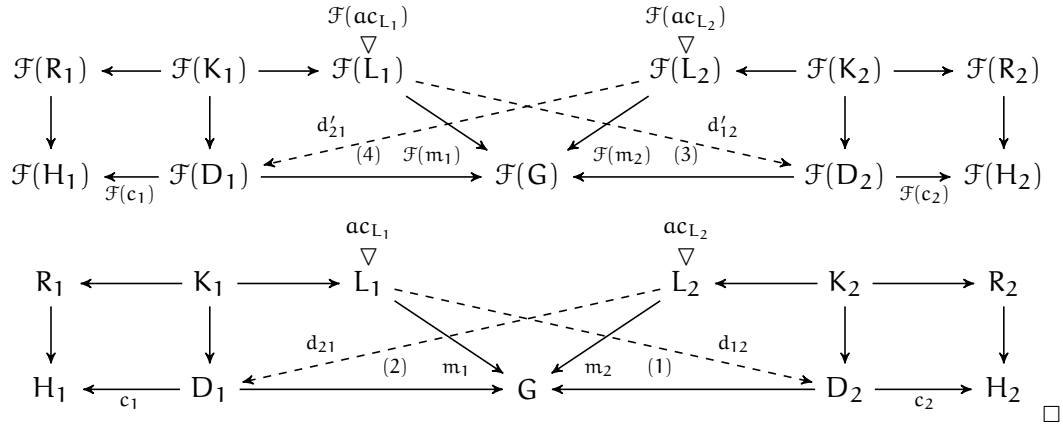
⁶ Note that if we formulate this theorem for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R creates \mathcal{M} -morphisms according to Definition 45. The proof for the adapted theorem works analogously to that of the current theorem.

$\mathcal{F}(R_1)), \mathcal{F}(\text{ac}_{L_1}))$, $\mathcal{F}(\rho_2) = (\mathcal{F}(p_2) = (\mathcal{F}(L_2) \leftarrow \mathcal{F}(K_2) \rightarrow \mathcal{F}(R_2)), \mathcal{F}(\text{ac}_{L_2}))$. According to Definition 24 this means that we have the morphisms $d'_{12} : \mathcal{F}(L_1) \rightarrow \mathcal{F}(D_2)$, $d'_{21} : \mathcal{F}(L_2) \rightarrow \mathcal{F}(D_1)$ in $\text{Mor}_{\mathcal{C}_2}$ such that the triangles (3), (4) commute and it additionally holds that $\mathcal{F}(c_2) \circ d'_{12} \models \mathcal{F}(\text{ac}_{L_1})$, $\mathcal{F}(c_1) \circ d'_{21} \models \mathcal{F}(\text{ac}_{L_2})$ for the translated nested application conditions $\mathcal{F}(\text{ac}_{L_1})$ and $\mathcal{F}(\text{ac}_{L_2})$. This leads to the corresponding unique morphisms $d_{12} : L_1 \rightarrow D_2$ and $d_{21} : L_2 \rightarrow D_1$ in $\text{Mor}_{\mathcal{C}_1}$ making the triangles (1) and (2) commute, because \mathcal{F} creates morphisms uniquely and preserves composition. It remains now to show that $(c_2 \circ d_{12} \models \text{ac}_{L_1})$ and $(c_1 \circ d_{21} \models \text{ac}_{L_2})$. Since $d'_{12} = \mathcal{F}(d_{12})$ and $d'_{21} = \mathcal{F}(d_{21})$ by the morphism creation property, we have the following:

$$\begin{aligned} & (\mathcal{F}(c_2) \circ \mathcal{F}(d_{12}) \models \mathcal{F}(\text{ac}_{L_1})) \wedge (\mathcal{F}(c_1) \circ \mathcal{F}(d_{21}) \models \mathcal{F}(\text{ac}_{L_2})) \\ & \xrightarrow{\text{funct. prop.}} (\mathcal{F}(c_2 \circ d_{12}) \models \mathcal{F}(\text{ac}_{L_1})) \wedge (\mathcal{F}(c_1 \circ d_{21}) \models \mathcal{F}(\text{ac}_{L_2})) \\ & \xrightarrow{\text{Lem. 11}} (c_2 \circ d_{12} \models \text{ac}_{L_1}) \wedge (c_1 \circ d_{21} \models \text{ac}_{L_2}) \end{aligned}$$

Thus, according to Definition 24, we have the parallel independence of transformations $G \xrightarrow{\rho_1, m_1} H_1$ and $G \xrightarrow{\rho_2, m_2} H_2$ in AS_1 with the match morphisms m_1 and m_2 .

The proof for the case of sequential independence works analogously.



In this section, we use the special kind of derived rules and derived nested application conditions of transformations translated by an \mathcal{M} -functor \mathcal{F} . This kind of rules and nested application conditions extends the corresponding notions from [96] recalled in Subsection 2.3.1 and is defined on the corresponding \mathcal{F} -images. A translated plain derived rule $p(\mathcal{F}(t))$ is a single rule from $\mathcal{F}(G_0)$ to $\mathcal{F}(G_n)$ containing all changes that should be done during the translated transformation $\mathcal{F}(t)$. A translated derived nested application condition $\text{ac}(\mathcal{F}(t))$ combines all nested application conditions of the translated transformation $\mathcal{F}(t)$ into a single nested application condition over the object $\mathcal{F}(G_0)$. Finally, a translated derived rule $\rho(\mathcal{F}(t))$ consists of a translated plain derived rule and a translated derived nested application condition of $\mathcal{F}(t)$.

Definition 53 (Translated Plain Derived Rule).

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathcal{C}_1, \mathcal{M}_1, P)$, $\text{AS}_2 = (\mathcal{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathcal{C}_1, \mathcal{M}_1) \rightarrow (\mathcal{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms⁷. For a transformation $\mathcal{F}(t) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n)$, the translated plain derived rule $p(\mathcal{F}(t))$ is defined by the span $\mathcal{F}(G_0) \leftarrow \mathcal{F}(D_0) \rightarrow \mathcal{F}(G_1)$ for

⁷ Pullback of \mathcal{M} -morphisms means that both cospan morphisms of the pullback diagram are in \mathcal{M} .

$n = 1$ and by iterated pullback construction leading to the span $\mathcal{F}(G_0) \leftarrow \mathcal{F}(D) \rightarrow \mathcal{F}(G_n)$ for $n \geq 2$ as depicted in the diagram below with dotted arrows where all morphisms are inclusions.

$$\begin{array}{ccccc}
 & & \mathcal{F}(D) & & \\
 & \swarrow & \downarrow & \searrow & \\
 \mathcal{F}(G_0) & \xleftarrow{=} & \mathcal{F}(D') & \xrightarrow{=} & \mathcal{F}(G_{n-1}) & \xleftarrow{=} & \mathcal{F}(D_n) & \xrightarrow{=} & \mathcal{F}(G_n)
 \end{array}$$

(PB)

Definition 54 (Translated Derived Nested Application Condition).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms, and a translated AC-disregarding transformation $\mathcal{F}(t) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n)$. The translated derived nested application condition $\text{ac}(\mathcal{F}(t))$ over the object $\mathcal{F}(G_0)$ is inductively defined as follows:

- For $\mathcal{F}(t)$ of length 0 with $\mathcal{F}(G_0) \cong \mathcal{F}(G'_0)$ and $\mathcal{F}(G'_0)$ from the extension diagram given in Definition 73 in Appendix A, let $\text{ac}(\mathcal{F}(t)) = \text{true}$.
- For $\mathcal{F}(t) : \mathcal{F}(G_0) \xrightarrow{\mathcal{F}(p_1), \mathcal{F}(m_1)} \mathcal{F}(G_1)$, let $\text{ac}(\mathcal{F}(t)) = \text{Shift}(\mathcal{F}(m_1), \mathcal{F}(\text{ac}_{L_1}))$ (see the picture (1) below).
- For $\mathcal{F}(t) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n) \Rightarrow \mathcal{F}(G_{n+1})$ with $n \geq 1$, let $\text{ac}(\mathcal{F}(t)) = \text{ac}(\mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n)) \wedge \mathbf{L}(\mathcal{F}(p_n^*), \text{ac}(\mathcal{F}(G_n) \Rightarrow \mathcal{F}(G_{n+1})))$ where $\mathcal{F}(p_n^*) = (\mathcal{F}(G_0) \leftarrow \mathcal{F}(D) \rightarrow \mathcal{F}(G_n))$ is the translated plain derived rule $p(\mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n))$ and $\text{ac}(\mathcal{F}(G_n) \Rightarrow \mathcal{F}(G_{n+1})) = \text{Shift}(\mathcal{F}(m_{n+1}), \mathcal{F}(\text{ac}_{L_{n+1}}))$ (see the picture (2) below).

$$\begin{array}{ccccc}
 & \mathcal{F}(\text{ac}_{L_1}) & & & \\
 & \nabla & & & \\
 \mathcal{F}(p_1) : & \mathcal{F}(L_1) & \xleftarrow{\quad} & \mathcal{F}(K_1) & \xrightarrow{\quad} & \mathcal{F}(R_1) \\
 & \downarrow \mathcal{F}(m_1) & & \downarrow & & \downarrow \\
 & \mathcal{F}(G_0) & \xleftarrow{\quad} & \mathcal{F}(D_0) & \xrightarrow{\quad} & \mathcal{F}(G_1)
 \end{array} \quad (1)$$

$$\begin{array}{ccccccc}
 \mathcal{F}(\text{ac}_{L_1}) & & & & \mathcal{F}(\text{ac}_{L_{n+1}}) & & \\
 \nabla & & & & \nabla & & \\
 \mathcal{F}(L_1) & \xleftarrow{\quad} & \mathcal{F}(K_1) & \xrightarrow{\quad} & \mathcal{F}(R_1) & & \\
 \downarrow \mathcal{F}(m_1) & & \downarrow & & \searrow & & \\
 \mathcal{F}(G_0) & \xleftarrow{\quad} & \mathcal{F}(D_0) & \xrightarrow{\quad} & \mathcal{F}(G_1) & \xrightarrow{*} & \mathcal{F}(G_n) \\
 & \underbrace{\hspace{10em}}_{\mathcal{F}(p_n^*)} & & & \mathcal{F}(L_{n+1}) & \xleftarrow{\quad} & \mathcal{F}(K_{n+1}) & \xrightarrow{\quad} & \mathcal{F}(R_{n+1}) \\
 & & & & \downarrow \mathcal{F}(m_{n+1}) & & \downarrow & & \downarrow \\
 & & & & \mathcal{F}(G_{n+1}) & \xleftarrow{\quad} & \mathcal{F}(D_{n+1}) & \xrightarrow{\quad} & \mathcal{F}(G_{n+2})
 \end{array} \quad (2)$$

Definition 55 (Translated Derived Rule).

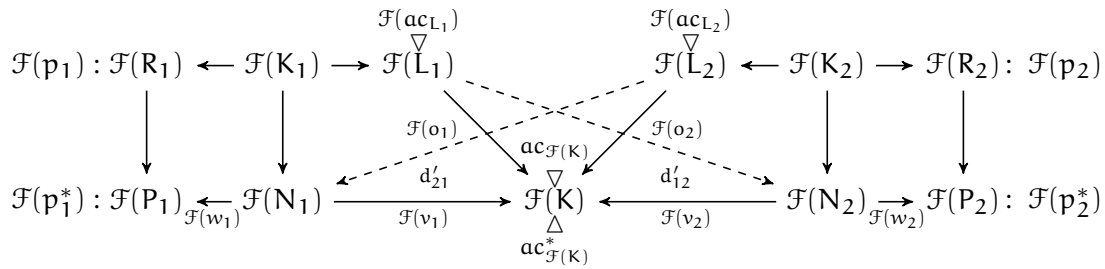
Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms. The translated derived rule of a translated AC-disregarding transformation $\mathcal{F}(t) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n)$ has the form $\rho(\mathcal{F}(t)) = (p(\mathcal{F}(t)), \text{ac}(\mathcal{F}(t)))$

where $p(\mathcal{F}(t))$ is the translated plain derived rule and $ac(\mathcal{F}(t))$ is the translated derived nested application condition of $\mathcal{F}(t)$.

In the following, we adapt the notions of a weak critical pair and a critical pair with nested application conditions introduced in Subsection 2.3.2 to the case of \mathcal{F} -reachable critical pairs. We will use these notions later on to define which requirements have to be satisfied to obtain the local confluence property for translated transformations with nested application conditions.

Definition 56 (\mathcal{F} -Reachable Weak Critical Pair of Transformations with Nested Application Conditions [213]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms, and translated derived rules $\mathcal{F}(\rho_1) = (\mathcal{F}(p_1), \mathcal{F}(ac_{L_1}))$, $\mathcal{F}(\rho_2) = (\mathcal{F}(p_2), \mathcal{F}(ac_{L_2}))$ where $\mathcal{F}(p_1)$, $\mathcal{F}(p_2)$ are translated plain derived rules and $\mathcal{F}(ac_{L_1})$, $\mathcal{F}(ac_{L_2})$ are translated nested application conditions. An \mathcal{F} -reachable weak critical pair of rules $\mathcal{F}(\rho_1)$ and $\mathcal{F}(\rho_2)$ is a pair $\mathcal{F}(P_1) \xrightarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ of AC-disregarding transformations in AS_2 where all objects are of the form $\mathcal{F}(X)$ for some $X \in \text{Obj}_{\mathbf{C}_1}$, all morphisms of the type $\mathcal{F}(A) \rightarrow \mathcal{F}(B)$ are of the form $\mathcal{F}(f)$ for some morphism $f : A \rightarrow B$, and the pair of morphisms $(\mathcal{F}(o_1), \mathcal{F}(o_2))$ is in \mathcal{E}'_2 .



Every \mathcal{F} -reachable weak critical pair induces nested application conditions $ac_{\mathcal{F}(K)}$ and $ac^*_{\mathcal{F}(K)}$ on $\mathcal{F}(K)$ defined by

$$ac_{\mathcal{F}(K)} = \mathbf{Shift}(\mathcal{F}(o_1), \mathcal{F}(ac_{L_1})) \wedge \mathbf{Shift}(\mathcal{F}(o_2), \mathcal{F}(ac_{L_2})),$$

called translated extension nested application condition, and

$$ac^*_{\mathcal{F}(K)} = \neg(ac^*_{\mathcal{F}(K), d'_{21}} \wedge ac^*_{\mathcal{F}(K), d'_{12}}),$$

called translated conflict-inducing nested application condition with $ac^*_{\mathcal{F}(K), d'_{12}}$ and $ac^*_{\mathcal{F}(K), d'_{21}}$ given as follows:

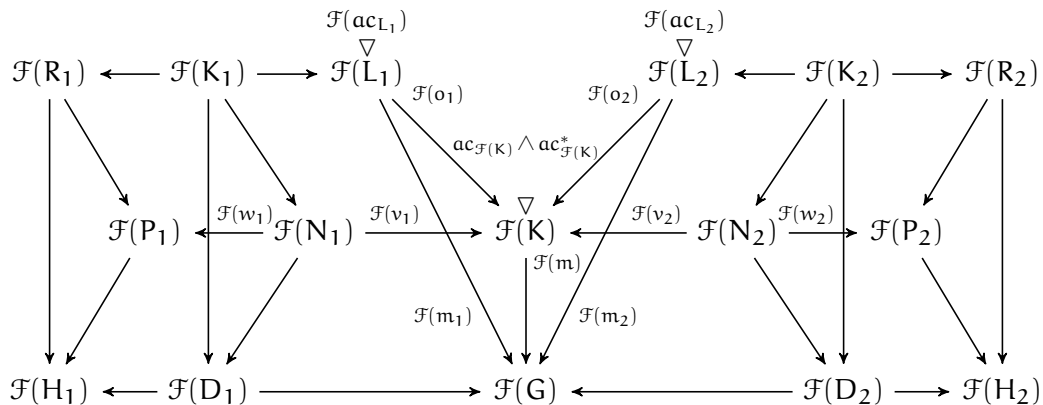
if $(\exists d'_{12} : \mathcal{F}(L_1) \rightarrow \mathcal{F}(N_2). \mathcal{F}(v_2) \circ d'_{12} = \mathcal{F}(o_1))$ then
 $ac^*_{\mathcal{F}(K), d'_{12}} = \mathbf{L}(\mathcal{F}(p_2^*), \mathbf{Shift}(\mathcal{F}(w_2) \circ d'_{12}, \mathcal{F}(ac_{L_1})))$ else
 $ac^*_{\mathcal{F}(K), d'_{12}} = \text{false}$
 if $(\exists d'_{21} : \mathcal{F}(L_2) \rightarrow \mathcal{F}(N_1). \mathcal{F}(v_1) \circ d'_{21} = \mathcal{F}(o_2))$ then
 $ac^*_{\mathcal{F}(K), d'_{21}} = \mathbf{L}(\mathcal{F}(p_1^*), \mathbf{Shift}(\mathcal{F}(w_1) \circ d'_{21}, \mathcal{F}(ac_{L_2})))$ else
 $ac^*_{\mathcal{F}(K), d'_{21}} = \text{false}$

where the translated plain derived rules $\mathcal{F}(p_1^*) = (\mathcal{F}(K) \xleftarrow{\mathcal{F}(v_1)} \mathcal{F}(N_1) \xrightarrow{\mathcal{F}(w_1)} \mathcal{F}(P_1))$ and $\mathcal{F}(p_2^*) = (\mathcal{F}(K) \xleftarrow{\mathcal{F}(v_2)} \mathcal{F}(N_2) \xrightarrow{\mathcal{F}(w_2)} \mathcal{F}(P_2))$ are defined by the corresponding DPOs.

As the next step, we define an \mathcal{F} -reachable critical pair extended to the case of transformations with nested application conditions.

Definition 57 (\mathcal{F} -Reachable Critical Pair of Transformations with Nested Application Conditions [213]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms, translated derived rules $\mathcal{F}(\rho_1) = (\mathcal{F}(p_1), \mathcal{F}(ac_{L_1}))$, $\mathcal{F}(\rho_2) = (\mathcal{F}(p_2), \mathcal{F}(ac_{L_2}))$ where $\mathcal{F}(p_1)$, $\mathcal{F}(p_2)$ are translated plain derived rules and $\mathcal{F}(ac_{L_1})$, $\mathcal{F}(ac_{L_2})$ are translated nested application conditions. An \mathcal{F} -reachable critical pair of translated derived rules $\mathcal{F}(\rho_1)$ and $\mathcal{F}(\rho_2)$ is an \mathcal{F} -reachable weak critical pair $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ with induced translated extension and conflict-inducing nested application conditions on $\mathcal{F}(K)$, $ac_{\mathcal{F}(K)}$ and $ac_{\mathcal{F}(K)}^*$, respectively, if there is a morphism $\mathcal{F}(m) : \mathcal{F}(K) \rightarrow \mathcal{F}(G) \in \mathcal{M}_2$ such that $\mathcal{F}(m) \models ac_{\mathcal{F}(K)} \wedge ac_{\mathcal{F}(K)}^*$ and $\mathcal{F}(m_i) = \mathcal{F}(m) \circ \mathcal{F}(o_i)$ for $i \in \{1, 2\}$ satisfies the gluing condition, i.e., $\mathcal{F}(m_i)$ has a pushout complement $\mathcal{F}(D_i)$ with respect to the translated plain derived rule $\mathcal{F}(p_i)$.



The following lemma shows that two nested application conditions are compatible if their translations by an \mathcal{M} -functor are compatible.

Lemma 19 (Creation of Compatibility of Nested Application Conditions).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (\mathcal{M}) -morphisms⁸. Then \mathcal{F} creates compatibility of nested application conditions, i.e.,

$$(\mathcal{F}(ac_P) \Rightarrow \mathcal{F}(ac'_P)) \text{ implies } (ac_P \Rightarrow ac'_P).$$

⁸ Note that if we formulate this lemma for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R creates \mathcal{M} -morphisms according to Definition 45. The proof for the adapted lemma works analogously to that of the current lemma.

Proof.

The detailed proof for this lemma is given in [Appendix B](#) on [page 287](#). \square

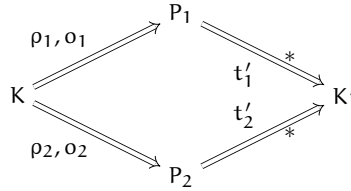
As the next step, we introduce the notion of $\mathcal{F}(\text{AC})$ -compatibility, which corresponds to the application of an \mathcal{M} -functor \mathcal{F} to the AC-compatibility given in Definition 28 in Subsection 2.3.2.

Definition 58 ($\mathcal{F}(\text{AC})$ -Compatibility of AC-Disregarding Transformations).

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $\text{AS}_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms, and a critical pair $P_1 \xrightarrow{\rho_1, \mathcal{O}_1} K \xrightarrow{\rho_2, \mathcal{O}_2} P_2$ of derived rules ρ_1 and ρ_2 with induced extension and conflict-inducing nested application conditions on K , ac_K and ac_K^* , respectively. Then the $\mathcal{F}(\text{AC})$ -compatibility is defined by the following implication

$$(\mathcal{F}(\text{ac}_K) \wedge \mathcal{F}(\text{ac}_K^*)) \Rightarrow (\mathcal{F}(\text{ac}(t_1)) \wedge \mathcal{F}(\text{ac}(t_2)))$$

where $t_i \triangleq K \xrightarrow{\rho_i, \mathcal{O}_i} P_i \xrightarrow{t'_i, *} K'$ are extended AC-disregarding transformations with derived nested application conditions $\text{ac}(t_i)$ on K for $i \in \{1, 2\}$.



The following lemma describes the dependency between the AC-compatibility and the $\mathcal{F}(\text{AC})$ -compatibility of AC-disregarding transformations.

Lemma 20 ($\mathcal{F}(\text{AC})$ -Compatibility Implies AC-Compatibility).

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $\text{AS}_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms and creates (\mathcal{M}) -morphisms⁹. Then two AC-disregarding transformations are AC-compatible if they are $\mathcal{F}(\text{AC})$ -compatible.

Proof.

The detailed proof for this lemma is given in [Appendix B](#) on [page 287](#). \square

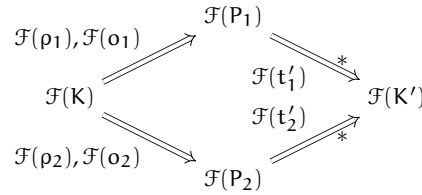
The next kind of compatibility, the $\text{AC}(\mathcal{F})$ -compatibility, corresponds to the AC-compatibility defined directly on the translated transformations in AS_2 .

Definition 59 ($\text{AC}(\mathcal{F})$ -Compatibility of AC-Disregarding Transformations [213]).

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $\text{AS}_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms, and an \mathcal{F} -reachable critical pair $\mathcal{F}(P_1) \xrightarrow{\mathcal{F}(\rho_1), \mathcal{F}(\mathcal{O}_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(\mathcal{O}_2)} \mathcal{F}(P_2)$ of translated derived rules $\mathcal{F}(\rho_1)$ and $\mathcal{F}(\rho_2)$ with induced extension and conflict-inducing nested application conditions on $\mathcal{F}(K)$, $\text{ac}_{\mathcal{F}(K)}$ and $\text{ac}_{\mathcal{F}(K)}^*$, respectively. Then the $\text{AC}(\mathcal{F})$ -compatibility is defined by the following implication

$$(\text{ac}_{\mathcal{F}(K)} \wedge \text{ac}_{\mathcal{F}(K)}^*) \Rightarrow (\text{ac}(\mathcal{F}(t_1)) \wedge \text{ac}(\mathcal{F}(t_2)))$$

where $\mathcal{F}(t_i) \triangleq \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_i), \mathcal{F}(\sigma_i)} \mathcal{F}(P_i) \xrightarrow{\mathcal{F}(t'_i)}^* \mathcal{F}(K')$ are translated extended AC-disregarding transformations with translated derived nested application conditions $\text{ac}(\mathcal{F}(t_i))$ on $\mathcal{F}(K)$ for $i \in \{1, 2\}$.



In the following, we adapt the Definition 29 introducing the strict AC-confluence of critical pairs to the case of \mathcal{F} -reachable critical pairs. We use this property later on to formulate our main result of this section concerning the creation of local confluence based on \mathcal{F} -reachable critical pairs for transformations with nested application conditions.

Definition 60 (Strict $\text{AC}(\mathcal{F})$ -Confluence of Critical Pairs [213]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms. An \mathcal{F} -reachable critical pair $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(\sigma_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(\sigma_2)} \mathcal{F}(P_2)$ of translated derived rules $\mathcal{F}(\rho_1)$ and $\mathcal{F}(\rho_2)$ with induced translated extension and conflict-inducing nested application conditions on $\mathcal{F}(K)$, $\text{ac}_{\mathcal{F}(K)}$ and $\text{ac}_{\mathcal{F}(K)}^*$, respectively, is called strictly $\text{AC}(\mathcal{F})$ -confluent if this pair is plain strictly confluent, i.e., strictly confluent in the sense of Definition 14 with AC-disregarding transformations $\mathcal{F}(t'_1)$ and $\mathcal{F}(t'_2)$ such that the extended AC-disregarding transformations $\mathcal{F}(t_i) \triangleq \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_i), \mathcal{F}(\sigma_i)} \mathcal{F}(P_i) \xrightarrow{\mathcal{F}(t'_i)}^* \mathcal{F}(K')$ with translated derived nested application conditions $\text{ac}(\mathcal{F}(t_i))$ on $\mathcal{F}(K)$ for $i \in \{1, 2\}$ are $\text{AC}(\mathcal{F})$ -compatible.

An example for constructing a concrete \mathcal{F} -reachable critical pair with nested application conditions in the category of typed attributed graphs over the hypergraph specific type graph $HGTG$ and showing its strict $\text{AC}(\mathcal{F})$ -confluence is given in Example 16.

The following lemma states that \mathcal{M} -functors preserve and create the plain strict confluence diagrams under certain assumptions. We need this property to be able to prove our central theorem of this section concerning the creation of local confluence based on \mathcal{F} -reachable critical pairs for rules with nested application conditions.

Lemma 21 (Preservation and Creation of Plain Strict Confluence).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms as well as creates morphisms⁹, and local confluence diagrams (1) in AS_1 and (2) in AS_2 with $\mathcal{F}(1) = (2)$. Then the diagram (1) is plain strictly confluent iff the diagram (2) is plain strictly confluent.

⁹ Note that if we formulate this lemma for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R creates \mathcal{M} -morphisms according to Definition 45. The requirement concerning the preservation of pullbacks of \mathcal{M} -morphisms remains unchanged, but refers now to the restricted \mathcal{M} -functor. The proof for the adapted lemma works analogously to that of the current lemma.



9

Definition 61 (*Preservation and Creation of \mathcal{E}' -Instances*).

1. \mathcal{F} translates \mathcal{E}'_1 -instances (a', b') into \mathcal{E}'_2 -instances (a'', b'') (or shortly \mathcal{F} preserves \mathcal{E}' -instances) if the following holds:

10 In general, Definition 61 can also be used in the context of non- \mathcal{M} -adhesive transformation systems with distinguished classes of monomorphisms.

2. \mathcal{F} creates \mathcal{E}'_1 -instances (a', b') from $\overline{\mathcal{E}}'_2$ -instances (a'', b'') (or shortly \mathcal{F} creates \mathcal{E}' -instances) if for all spans $(C \xleftarrow{a} P \xrightarrow{b} P')$ it holds:

$$\begin{aligned} & \forall (a'', b'') \in \overline{\mathcal{E}}'_2. (2) \text{ commutes } \wedge b'' \in \mathcal{M}_2 \Rightarrow \\ & \exists (a', b') \in \mathcal{E}'_1. a'' = \mathcal{F}(a') \wedge b'' = \mathcal{F}(b') \wedge (1) \text{ commutes } \wedge b' \in \mathcal{M}_1. \end{aligned}$$

$$\begin{array}{ccc} P & \xrightarrow{b} & P' \\ a \downarrow & (1) & \downarrow a' \\ C & \xrightarrow{b'} & C' \end{array} \qquad \begin{array}{ccc} \mathcal{F}(P) & \xrightarrow{\mathcal{F}(b)} & \mathcal{F}(P') \\ \mathcal{F}(a) \downarrow & (2) & \downarrow a'' = \mathcal{F}(a') \\ \mathcal{F}(C) & \xrightarrow{b'' = \mathcal{F}(b')} & C'' = \mathcal{F}(C') \end{array}$$

We need the following two compatibility properties to show later on the dependency between $\mathcal{F}(\text{AC})$ -compatibility and $\text{AC}(\mathcal{F})$ -compatibility. To ensure the compatibility of \mathcal{F} with the **Shift**-transformation, we must interpret the class \mathcal{E}' , occurring in the definition of **Shift** (see Remark 4), by $\mathcal{F}(\mathcal{E}'_1)$ for the target transformation system of the \mathcal{M} -functor. This guarantees that equivalent nested application conditions are computed by the **Shift**-transformations for the source and the target transformation systems of the \mathcal{M} -functor. Taking the set of all jointly epimorphic morphisms for the target transformation system would lead to possibly larger nested application conditions in particular for the case of nested application conditions of the form $ac_P = \exists(a, ac_C)$, because the **Shift**-transformation would also include the cases where a nested application condition is shifted using non- \mathcal{F} -images a', b' (see Remark 4).

Lemma 22 (\mathcal{F} is Compatible with Shift-Transformation).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves and creates \mathcal{E}' -instances according to Definition 61. Then \mathcal{F} is compatible with the **Shift**-transformation, i.e.,

$$\forall ac_P, b : P \rightarrow P'. \mathcal{F}(\text{Shift}(b, ac_P)) = \text{Shift}(\mathcal{F}(b), \mathcal{F}(ac_P)).$$

Proof.

This lemma can be proved by induction over the depth of a nested application condition. For detailed proof see [Appendix B](#) on page 289. \square

Lemma 23 (\mathcal{F} is Compatible with L-Transformation).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that translates and creates rule applicability according to Theorem 3 as well as preserves pullbacks of \mathcal{M} -morphisms¹¹. Then \mathcal{F} is compatible with the **L**-transformation, i.e.,

$$\forall ac_R, \rho \in P \cup P^*. \mathcal{F}(\text{L}(\rho, ac_R)) = \text{L}(\mathcal{F}(\rho), \mathcal{F}(ac_R))$$

¹¹ Note that if we formulate this lemma for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R translates and creates rule applicability for the case of \mathcal{M} -matches according to Remark 12. The requirement concerning the preservation of pullbacks of \mathcal{M} -morphisms remains unchanged, but refers now to the restricted \mathcal{M} -functor. The proof for the adapted lemma works analogously to that of the current lemma.

where P^* is a set of derived rules in AS_1 .

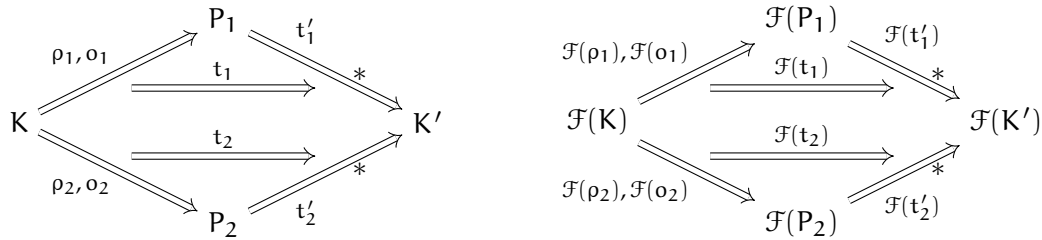
Proof.

This lemma can be proved by induction over the depth of a nested application condition. For detailed proof see [Appendix B](#) on [page 291](#). \square

In the next lemma we describe the dependency between $\mathcal{F}(AC)$ -compatibility and $AC(\mathcal{F})$ -compatibility of AC -disregarding transformations.

Lemma 24 ($AC(\mathcal{F})$ -Compatibility Implies $\mathcal{F}(AC)$ -Compatibility).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates morphisms and is compatible with **Shift**- and **L**-transformations according to [Lemmas 22](#) and [23](#)¹². Then two AC -disregarding transformations $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ are $\mathcal{F}(AC)$ -compatible if their corresponding translations by the \mathcal{M} -functor \mathcal{F} are $AC(\mathcal{F})$ -compatible.



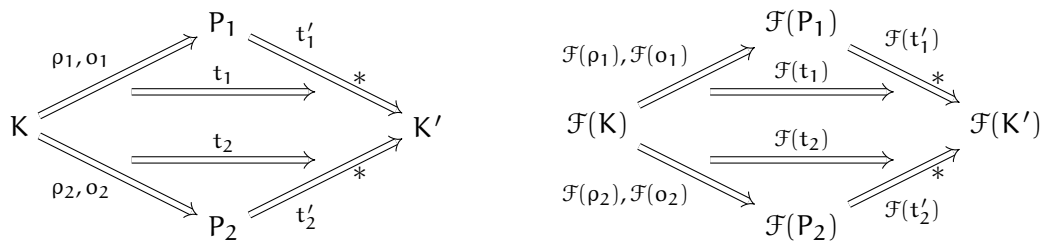
Proof.

The detailed proof for this lemma is given in [Appendix B](#) on [page 293](#). \square

Using the dependency between $\mathcal{F}(AC)$ -compatibility and $AC(\mathcal{F})$ -compatibility together with the fact that $\mathcal{F}(AC)$ -compatibility implies AC -compatibility according to [Lemma 20](#), we can show in the subsequent [Lemma 25](#) that the $AC(\mathcal{F})$ -compatibility of AC -disregarding transformations in the target transformation system of an \mathcal{M} -functor leads to the corresponding AC -compatibility of transformations in the source transformation system. We use this property later on to prove our main theoretical result of this section.

Lemma 25 ($AC(\mathcal{F})$ -Compatibility Implies AC -Compatibility [[213](#)]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (\mathcal{M}) -morphisms and is compatible with **Shift**- and **L**-transformations according to [Lemmas 22](#) and [23](#)¹². Then two AC -disregarding transformations $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ are AC -compatible if their corresponding translations by the \mathcal{M} -functor \mathcal{F} are $AC(\mathcal{F})$ -compatible.



Proof.

The detailed proof for this lemma is given in [Appendix B](#) on [page 298](#). □

The lemma, given in the following, states that an \mathcal{M} -functor \mathcal{F} preserves both, weak critical pairs and critical pairs, under certain assumptions.

Lemma 26 (Preservation of (Weak) Critical Pairs [213]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, a (weak) critical pair $P_1 \xleftarrow{\rho_1, \sigma_1} K \xrightarrow{\rho_2, \sigma_2} P_2$ in AS_1 , and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (\mathcal{M}) -morphisms and is compatible with $\mathcal{E}' - \mathcal{M}$ pair factorization as well as with **Shift**- and **L**-transformations¹³. Then $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(\sigma_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(\sigma_2)} \mathcal{F}(P_2)$ is the corresponding \mathcal{F} -reachable (weak) critical pair in AS_2 .

Proof.

The detailed proof for this lemma is given in [Appendix B](#) on [page 298](#). □

In the following theorem, which is our main theoretical result of this section, we deduce local confluence of the source transformation system of an \mathcal{M} -functor from the strict $AC(\mathcal{F})$ -confluence of the corresponding \mathcal{F} -reachable critical pairs in the target transformation system. Examples for using this theorem for the local confluence analysis of a concrete hypergraph resp. PTI net transformation system via critical pairs with nested application conditions are given in [Sections 7.3](#) and [10.3](#), respectively.

Theorem 11 (Creation of Local Confluence Based on \mathcal{F} -Reachable Critical Pairs for Rules with Nested Application Conditions [213]).

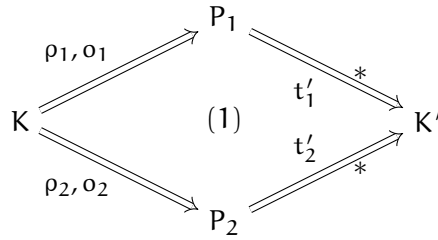
Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that is compatible with $\mathcal{E}' - \mathcal{M}$ pair factorization as well as **Shift**- and **L**-transformations and creates (direct) transformations according to [Theorem 3](#) and (\mathcal{M}) -morphisms¹⁴. Then the \mathcal{M} -adhesive transformation system AS_1 is locally confluent for all pairs of direct transformations $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ with nested application conditions if all \mathcal{F} -reachable critical pairs with nested application conditions of $\mathcal{F}(\rho_1)$ and $\mathcal{F}(\rho_2)$ are strictly $AC(\mathcal{F})$ -confluent.

Proof.

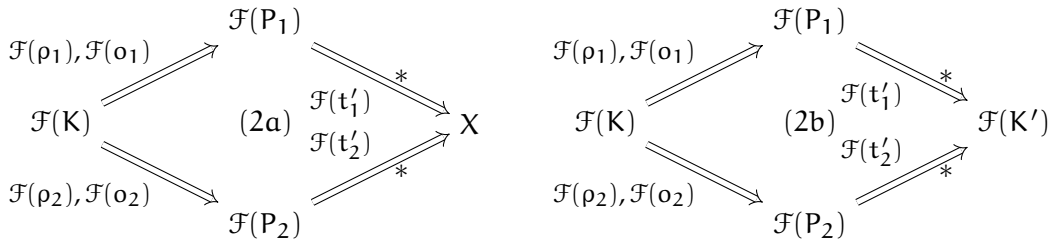
In order to show local confluence for AS_1 , it suffices to show by [Fact 5](#) that all its critical pairs are strictly AC -confluent. This means according to [Definition 29](#) that all critical pairs of ρ_1

-
- ¹² Note that if we formulate this lemma for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R creates \mathcal{M} -morphisms according to [Definition 45](#). The requirements concerning the compatibility of \mathcal{F} with **Shift**- and **L**-transformations remain unchanged, but refer now to the restricted \mathcal{M} -functor. The proof for the adapted lemma works analogously to that of the current lemma.
- ¹³ Note that if we formulate this lemma for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R creates \mathcal{M} -morphisms according to [Definition 45](#). The requirements concerning the compatibility of \mathcal{F} with $\mathcal{E}' - \mathcal{M}$ pair factorization as well as with **Shift**- and **L**-transformations remain unchanged, but refer now to the restricted \mathcal{M} -functor. The proof for the adapted lemma works analogously to that of the current lemma.
- ¹⁴ Note that if we formulate this theorem for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R creates (direct) transformations according to [Remark 12](#) and \mathcal{M} -morphisms according to [Definition 45](#). The requirements concerning the compatibility of \mathcal{F} with $\mathcal{E}' - \mathcal{M}$ pair factorization as well as with **Shift**- and **L**-transformations remain unchanged, but refer now to the restricted \mathcal{M} -functor. The proof for the adapted theorem works analogously to that of the current theorem.

and ρ_2 in AS_1 are plain strictly confluent and AC-compatible, i.e., consider any critical pair $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ in AS_1 , then we have to show the plain strict confluence of diagram (1) given below as well as the AC-compatibility of the extended AC-disregarding transformations $t_i \triangleq K \xrightarrow{\rho_i, o_i} P_i \xrightarrow{t'_i} K'$ with the derived nested application conditions $ac(t_i)$ on K for $i \in \{1, 2\}$ given by the following implication: $(ac_K \wedge ac_K^*) \Rightarrow (ac(t_1) \wedge ac(t_2))$ (see Definition 28).



Fix a critical pair $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ of ρ_1 and ρ_2 in AS_1 , then by Lemma 26 $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ is an \mathcal{F} -reachable critical pair of $\mathcal{F}(\rho_1)$ and $\mathcal{F}(\rho_2)$ in AS_2 . By assumption of the current theorem, this \mathcal{F} -reachable critical pair is strictly $AC(\mathcal{F})$ -confluent. This means according to Definition 60 that we have the plain strict confluence diagram (2a) as given below with the $AC(\mathcal{F})$ -compatibility of the extended AC-disregarding transformations $\mathcal{F}(t_i) \triangleq \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_i), \mathcal{F}(o_i)} \mathcal{F}(P_i) \xrightarrow{\mathcal{F}(t'_i)} X$ for $i \in \{1, 2\}$. From the diagram (2a) we obtain the plain strict confluence diagram of the form (2b), because \mathcal{F} creates transformations and morphisms (and hence also isomorphisms) by assumption, such that the $AC(\mathcal{F})$ -compatibility described by the following implication holds: $(ac_{\mathcal{F}(K)} \wedge ac_{\mathcal{F}(K)}^*) \Rightarrow (ac(\mathcal{F}(t_1)) \wedge ac(\mathcal{F}(t_2)))$ with the extended AC-disregarding transformations $\mathcal{F}(t_i) \triangleq \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_i), \mathcal{F}(o_i)} \mathcal{F}(P_i) \xrightarrow{\mathcal{F}(t'_i)} \mathcal{F}(K')$ and the derived nested application conditions $ac(\mathcal{F}(t_i))$ on $\mathcal{F}(K)$ for $i \in \{1, 2\}$.



Now, applying Lemma 21, we obtain that the diagram (1) is also a plain strict confluence diagram. Furthermore, by application of Lemma 25, the $AC(\mathcal{F})$ -compatibility of the extended AC-disregarding transformations $\mathcal{F}(t_i) \triangleq \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_i), \mathcal{F}(o_i)} \mathcal{F}(P_i) \xrightarrow{\mathcal{F}(t'_i)} \mathcal{F}(K')$ from the diagram (2b) implies the AC-compatibility of the corresponding extended AC-disregarding transformations $t_i \triangleq K \xrightarrow{\rho_i, o_i} P_i \xrightarrow{t'_i} K'$ from the diagram (1) (in both cases for $i \in \{1, 2\}$), which was to be shown. \square

4.3 \mathcal{F} -TRANSFER OF TERMINATION, CONFLUENCE, AND FUNCTIONAL BEHAVIOR

In this section, we provide the general theory helping us to verify semantical properties like termination, confluence, and functional behavior using \mathcal{M} -functors for \mathcal{M} -adhesive transformation systems containing rules without or with nested application conditions. Firstly, we introduce the notion of \mathcal{F} -termination needed for the termination analysis of the target transformation system of an \mathcal{M} -functor. Secondly, we show that we can derive the \mathcal{F} -termination of a target transformation system of an \mathcal{M} -functor from the termination of its source transformation system and vice versa. Finally, we state under which requirements the source transformation system of an \mathcal{M} -functor is confluent and has functional behavior.

For the target transformation system of an \mathcal{M} -functor, we consider the notion of \mathcal{F} -termination. A transformation system is \mathcal{F} -terminating if there is no infinite sequence such that the start object of the considered sequence is an \mathcal{F} -image. We need this restriction, because the \mathcal{F} -termination requires termination for sequences starting with some translated object $\mathcal{F}(G_0)$ instead of a general object G'_0 .

Definition 62 (\mathcal{F} -Termination of a Translated \mathcal{M} -Adhesive Transformation System [209]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$. A translated \mathcal{M} -adhesive transformation system AS_2 is called \mathcal{F} -terminating if there is no infinite sequence $\mathcal{F}(G_0) \xrightarrow{\mathcal{F}(\rho_1), m'_1} G'_1 \xrightarrow{\mathcal{F}(\rho_2), m'_2} G'_2 \xrightarrow{\mathcal{F}(\rho_3), m'_3} \dots$ with $\mathcal{F}(\rho_1), \mathcal{F}(\rho_2), \mathcal{F}(\rho_3), \dots \in \mathcal{F}(P)$ and match morphisms $m'_1, m'_2, m'_3, \dots \in \text{Mor}_{\mathbf{C}_2}$.

\mathcal{M} -functors transfer termination according to the following theorem.

Theorem 12 (\mathcal{F} -Transfer of Termination [209]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that translates and creates (direct) transformations¹⁵ according to Theorem 1. Then an \mathcal{M} -adhesive transformation system AS_1 is terminating iff the corresponding translated \mathcal{M} -adhesive transformation system AS_2 is \mathcal{F} -terminating.

Proof (by Contraposition).

Consider a non-terminating sequence $\mathcal{F}(G_0) \xrightarrow{\mathcal{F}(\rho_1), m'_1} G'_1 \xrightarrow{\mathcal{F}(\rho_2), m'_2} G'_2 \xrightarrow{\mathcal{F}(\rho_3), m'_3} \dots$ in AS_2 (see the right part of the diagram below). We generate stepwise a non-terminating sequence $G_0 \xrightarrow{\rho_1, m_1} G_1 \xrightarrow{\rho_2, m_2} G_2 \xrightarrow{\rho_3, m_3} \dots$ in AS_1 (see the left part of the diagram below) with $m'_i = \mathcal{F}(m_i)$ and $G'_j = \mathcal{F}(G_j)$ for $i \in \{1, 2, 3\}$, $j \in \{1, 2\}$ by pointwise application of the assumption that the given \mathcal{M} -functor creates direct transformations.

Analogously, for a given non-terminating sequence $G_0 \xrightarrow{\rho_1, m_1} G_1 \xrightarrow{\rho_2, m_2} G_2 \xrightarrow{\rho_3, m_3} \dots$ in AS_1 (see the left part of the diagram below), we generate stepwise a non-terminating sequence $\mathcal{F}(G_0) \xrightarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G_1) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(G_2) \xrightarrow{\mathcal{F}(\rho_3), \mathcal{F}(m_3)} \dots$ in AS_2 (see the right part of the diagram below) by pointwise application of the assumption that the given \mathcal{M} -functor translates direct transformations.

¹⁵ Note that if we formulate this theorem for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R translates and creates (direct) transformations according to Theorem 2. The proof for the adapted theorem works analogously to that of the current theorem.

$$\begin{array}{ccc}
G_0 & \xrightarrow{\mathcal{F}} & \mathcal{F}(G_0) \\
\Downarrow \rho_1, m_1 & & \Downarrow \mathcal{F}(\rho_1), m'_1 = \mathcal{F}(m_1) \\
G_1 & \xrightarrow{\mathcal{F}} & G'_1 = \mathcal{F}(G_1) \\
\Downarrow \rho_2, m_2 & & \Downarrow \mathcal{F}(\rho_2), m'_2 = \mathcal{F}(m_2) \\
G_2 & \xrightarrow{\mathcal{F}} & G'_2 = \mathcal{F}(G_2) \\
\Downarrow \rho_3, m_3 & & \Downarrow \mathcal{F}(\rho_3), m'_3 = \mathcal{F}(m_3) \\
\vdots & & \vdots
\end{array}$$

□

The following remark summarizes, which technical properties are required to be able to extend the Theorem 12 to transformations with nested application conditions.

Remark 14 (*\mathcal{F} -Transfer of Termination for the Case of Transformations with Nested Application Conditions*).

We can extend Theorem 12, formulated for rules without nested application conditions, to the case of transformations with nested application conditions requiring that an \mathcal{M} -functor \mathcal{F} translates and creates (direct) transformations with nested application conditions according to Theorem 3¹⁶. The proof for this extension works analogously to the proof of Theorem 12.

As we already know from Subsubsection 2.2.2.3, a terminating transformation system is called confluent if the application of its rules in any order yields a result, which is unique up to isomorphism. The rules of a confluent transformation system are stable in the sense that they provide a unique starting point for further computations. In the next step we combine the results concerning the creation of local confluence and the \mathcal{F} -transfer of termination, which are needed to deduce the creation of confluence and functional behavior. For this reason we formulate and prove the following theorem.

Theorem 13 (*\mathcal{F} -Transfer of Confluence and Functional Behavior [209]*).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that translates and creates (direct) transformations according to Theorem 1 as well as creates morphisms¹⁷. Then AS_1 is locally confluent and terminating iff AS_2 is locally confluent for all translated transformation spans and \mathcal{F} -terminating. Moreover, AS_1 is confluent and has functional behavior if AS_2 is locally confluent for all translated transformation spans and \mathcal{F} -terminating.

Proof.

Assume that AS_1 is locally confluent and terminating. Using Theorems 6 and 12 we obtain that also AS_2 is locally confluent for all translated transformation spans and \mathcal{F} -terminating.

Analogously, assume that AS_2 is locally confluent for all translated transformation spans and \mathcal{F} -terminating. Using Theorems 6 and 12 we obtain that also AS_1 is locally confluent and

¹⁶ Note that considering a restricted \mathcal{M} -functor, we require for the mentioned theorem extension that \mathcal{F}_R translates and creates (direct) transformations with nested application conditions according to Remark 12. The corresponding proof works then analogously to that of the mentioned theorem extension.

¹⁷ Note that if we formulate this theorem for a restricted \mathcal{M} -functor, we require that \mathcal{F}_R translates and creates (direct) transformations according to Theorem 2 and creates \mathcal{M} -morphisms according to Definition 45. The proof for the adapted theorem works analogously to that of the current theorem.

terminating. Furthermore, local confluence and termination of AS_1 imply according to Remark 2 that AS_1 is confluent and has functional behavior. \square

The Theorem 13 can be adapted to transformations with nested application conditions as described in the following remark.

Remark 15 (\mathcal{F} -Transfer of Confluence and Functional Behavior for the Case of Transformations with Nested Application Conditions).

We can extend Theorem 13, formulated for rules without nested application conditions, to the case of transformations with nested application conditions requiring that an \mathcal{M} -functor \mathcal{F} translates and creates (direct) transformations with nested application conditions according to Theorem 3 as well as creates (\mathcal{M} -)morphisms¹⁸. The proof for this extension works analogously to the proof of Theorem 13 using Theorem 9 and Remark 14.

¹⁸ Note that considering a restricted \mathcal{M} -functor, we require for the mentioned theorem extension that \mathcal{F}_R translates and creates (direct) transformations with nested application conditions according to Remark 12 and creates \mathcal{M} -morphisms according to Definition 45. The corresponding proof works then analogously to that of the mentioned theorem extension.

Part III

APPLICATION TO HYPERGRAPH TRANSFORMATION SYSTEMS

FUNCTOR BETWEEN CATEGORIES OF HYPERGRAPHS AND TYPED ATTRIBUTED GRAPHS

In this and the next two chapters, we focus on the application of the general approach from Chapters 3 and 4 to the \mathcal{M} -adhesive categories of hypergraphs and typed attributed graphs presented in Subsections 2.4.2 and 2.4.1, respectively. In Section 5.1 we introduce the subcategory $\mathbf{AGraphs}_{HGTG}$ of typed attributed graphs over the specific typed attributed graph $HGTG$ allowing us to represent hypergraphs as typed attributed graphs. We use subsequently this subcategory in Section 5.2 as the target category for the functor construction that defines the translation of objects and morphisms from the \mathcal{M} -adhesive category of hypergraphs $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ into the corresponding objects and morphisms from the \mathcal{M} -adhesive category of typed attributed graphs $(\mathbf{AGraphs}_{HGTG}, \mathcal{M}_2)$ where \mathcal{M}_1 and \mathcal{M}_2 denote the classes of all injective morphisms from the categories $\mathbf{HyperGraphs}$ and $\mathbf{AGraphs}_{HGTG}$, respectively.

5.1 TYPED ATTRIBUTED GRAPHS OVER THE HYPERGRAPH TYPE GRAPH $HGTG$

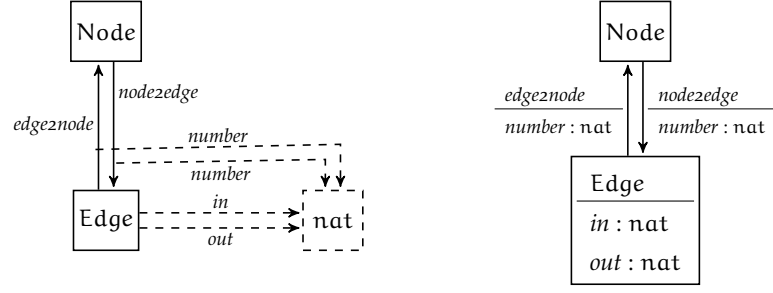
In this section, we introduce the category of typed attributed graphs, which we call $\mathbf{AGraphs}_{HGTG}$, constructed over the specific typed attributed graph $HGTG$ allowing for the hypergraph representation. For our functor construction, we use the subcategory $\mathbf{AGraphs}_{HGTG}^{\mathbb{N}}$ of $\mathbf{AGraphs}_{HGTG}$ where all objects are restricted to the data type \mathbf{NAT} , $V_D^G = \mathbb{N}$, and all data type morphisms as well as V_D -components of the E-graph morphisms are restricted to identities. Morphisms in $\mathbf{AGraphs}_{HGTG}^{\mathbb{N}}$ are defined component-wise and are *type-compatible*.

The construction of pushouts and pullbacks in $\mathbf{AGraphs}_{HGTG}^{\mathbb{N}}$ can be obtained componentwise, as in $\mathbf{AGraphs}_{HGTG}$, with the identical data type component.

As already mentioned in Subsection 2.4.1, the category $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ is \mathcal{M} -adhesive for each type graph ATG where \mathcal{M} -morphisms are injective with the isomorphic data type part. Hence also the special case of $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ with $ATG = HGTG$ and the subclass $\mathcal{M} = \mathcal{M}_2$ of all injective typed attributed graph morphisms is \mathcal{M} -adhesive. Moreover, the fact that $(\mathbf{AGraphs}_{HGTG}, \mathcal{M}_2)$ is an \mathcal{M} -adhesive category implies that also $(\mathbf{AGraphs}_{HGTG}^{\mathbb{N}}, \mathcal{M}_2^{\mathbb{N}})$ is an \mathcal{M} -adhesive category with the class \mathcal{M}_2 restricted to $\mathcal{M}_2^{\mathbb{N}}$ with the identical data type component.

The initial object $\emptyset^{\mathbb{N}}$ in $\mathbf{AGraphs}_{HGTG}^{\mathbb{N}}$ is empty excepting the data type part and $\emptyset^{\mathbb{N}} = \mathcal{F}_{HG}(\emptyset)$ where \emptyset is an initial hypergraph. In the following, we use the short notation $(\mathbf{AGraphs}_{HGTG}, \mathcal{M}_2)$ for the \mathcal{M} -adhesive category of typed attributed graphs over $HGTG$ instead of the long notation $(\mathbf{AGraphs}_{HGTG}^{\mathbb{N}}, \mathcal{M}_2^{\mathbb{N}})$.

We want to express hypergraphs as typed attributed graphs to be able to analyze them using the tool AGG. For this reason we use the hypergraph specific type graph $HGTG$, which suites well for description of hypergraphs. $HGTG$ is shown in Figure 29 as an E-graph on the left (where nodes and edges correspond to the graph nodes (V_G) and graph edges (E_G), respectively, dashed nodes are the data nodes (V_D), and dashed edges

Figure 29: Attributed type graph $HGTG$

represent the edges for node and edge attribution) and in its short attributed notation on the right.

The meaning of every depicted element of $HGTG$ is as follows. Nodes of the types *Node* and *Edge* represent hypergraph nodes and hyperedges, respectively. Edges of the types *node2edge*, *edge2node* represent hyperedge tentacles and are attributed by a number *number*, which contains the position of a node in the source resp. target string of the considered hyperedge. Nodes of the type *Edge* have two attributes *in* and *out* giving the number of nodes in the pre- and postdomain of a hyperedge, which are needed to ensure the preservation of an *Edge* node's environment using typed attributed graph morphisms. All node and edge attributes are typed over natural numbers represented by the data node *nat*.

5.2 TRANSLATION OF HYPERGRAPHS INTO TYPED ATTRIBUTED GRAPHS

We start this section with the formal definition of the functor $\mathcal{F}_{HG} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{HGTG}$. For this purpose, we have to define \mathcal{F}_{HG} on objects and morphisms.

Definition 63 (Functor \mathcal{F}_{HG} [211]).

Consider hypergraphs $G = (V_G, E_G, s_G, t_G)$ and $G_i = (V_{G_i}, E_{G_i}, s_{G_i}, t_{G_i})$ for $i \in \{1, 2\}$.

- **Translation of objects:**

We define the object $\mathcal{F}_{HG}(G) = ((G', NAT), type)^1$ in $\mathbf{AGraphs}_{HGTG}$ with a morphism $type : (G', NAT) \rightarrow (HGTG, D_{fin})$ and an E-graph $G' = (V_G^{G'}, V_D^{G'} = \mathbb{N}, E_G^{G'}, E_{NA}^{G'}, E_{EA}^{G'}, (s_j^{G'}, t_j^{G'})_{j \in \{G, NA, EA\}})$ as follows where we use the following abbreviations: $node2edge \triangleq n2e$, $edge2node \triangleq e2n$, $number \triangleq num$:

$$V_G^{G'} = V_G \uplus E_G \text{ (graph nodes),}$$

$$E_G^{G'} = E_{n2e}^{G'} \uplus E_{e2n}^{G'} \text{ (graph edges) with}$$

$$E_{n2e}^{G'} = \{((v, e), n) \in (V_G \times E_G) \times \mathbb{N} \mid s_G^n(e) = v\},$$

$$E_{e2n}^{G'} = \{((e, v), n) \in (E_G \times V_G) \times \mathbb{N} \mid t_G^n(e) = v\}$$

where $s_G^n(e)$ ($t_G^n(e)$) is the n -th node in the string $s_G(e)$ ($t_G(e)$),

$$E_{NA}^{G'} = E_{in}^{G'} \uplus E_{out}^{G'} \text{ (node attribute edges) with}$$

$$E_{in}^{G'} = \{(e, n, in) \mid (e, n) \in E_G \times \mathbb{N} \wedge |s_G(e)| = n\},$$

$$E_{out}^{G'} = \{(e, n, out) \mid (e, n) \in E_G \times \mathbb{N} \wedge |t_G(e)| = n\}$$

¹ In the following, we also use the short notation $\mathcal{F}_{HG}(G) = G'$.

where $|w|$ is the length of the word w ,

$E_{EA}^{G'} = E_s^{G'} \uplus E_t^{G'}$ (edge attribute edges) with

$$E_s^{G'} = \{(n, (v, e)) \in \mathbb{N} \times (V_G \times E_G) \mid s_G^n(e) = v\}^2,$$

$$E_t^{G'} = \{(n, (e, v)) \in \mathbb{N} \times (E_G \times V_G) \mid t_G^n(e) = v\}^2,$$

(and the corresponding source and target functions)

$s_G^{G'}, t_G^{G'} : E_G^{G'} \rightarrow V_G^{G'}$ defined by $s_G^{G'}((x, y), n) = x$, $t_G^{G'}((x, y), n) = y$,

$s_{NA}^{G'} : E_{NA}^{G'} \rightarrow V_G^{G'}$ defined by $s_{NA}^{G'}(e, n, x) = e$,

$t_{NA}^{G'} : E_{NA}^{G'} \rightarrow \mathbb{N}$ defined by $t_{NA}^{G'}(e, n, x) = n$,

$s_{EA}^{G'} : E_{EA}^{G'} \rightarrow E_G^{G'}$ defined by $s_{EA}^{G'}(n, (x, y)) = ((x, y), n)$,

$t_{EA}^{G'} : E_{EA}^{G'} \rightarrow \mathbb{N}$ defined by $t_{EA}^{G'}(n, (x, y)) = n$.

To simplify the notation, we flatten nested tuples most of the time, i.e., the tuple $((x, y), z)$ is written (x, y, z) .

The **AGraphs**_{HGTG}-morphism type $: (G', NAT) \rightarrow (HGTG, D_{fin})$ is given by the final morphism of data types from NAT to the final algebra D_{fin} and $type^{G'} : G' \rightarrow HGTG$ is given by the E-graph morphism $type^{G'} = (type_{V_G}, type_{V_D}, type_{E_G}, type_{E_{NA}}, type_{E_{EA}})$ where

$type_{V_G} : V_G^{G'} \rightarrow V_G^{HGTG}$ with $x \mapsto \text{Node}$ (if $x \in V_G$), $x \mapsto \text{Edge}$ (if $x \in E_G$),

$type_{V_D} : \mathbb{N} \rightarrow D_{fin, nat}$ with $x \mapsto \text{nat}$ (if $x \in \mathbb{N}$),

$type_{E_G} : E_G^{G'} \rightarrow E_G^{HGTG}$ with $x \mapsto y$ for $x \in E_y^{G'}$ and $y \in \{n2e, e2n\}$,

$type_{E_{NA}} : E_{NA}^{G'} \rightarrow E_{NA}^{HGTG}$ with $x \mapsto y$ for $x \in E_y^{G'}$ and $y \in \{in, out\}$,

$type_{E_{EA}} : E_{EA}^{G'} \rightarrow E_{EA}^{HGTG}$ with $x \mapsto \text{num}$.

- **Translation of morphisms:**

For an arbitrary hypergraph morphism $f : G_1 \rightarrow G_2$ with $f = (f_V : V_{G_1} \rightarrow V_{G_2}, f_E : E_{G_1} \rightarrow E_{G_2})$, we define $\mathcal{F}_{HG}(f) : \mathcal{F}_{HG}(G_1) \rightarrow \mathcal{F}_{HG}(G_2)$ with $\mathcal{F}_{HG}(G_i) = (V_G^{G_i}, \mathbb{N}, E_G^{G_i}, E_{NA}^{G_i}, E_{EA}^{G_i}, (s_j^{G_i}, t_j^{G_i})_{j \in \{G, NA, EA\}})$ for $i \in \{1, 2\}$ by $\mathcal{F}_{HG}(f) = f' = (f'_{V_G}, f'_{V_D} = id_{\mathbb{N}}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}})$ where

$f'_{V_G} : V_G^{G_1} \rightarrow V_G^{G_2}$ with $V_G^{G_i} = V_{G_i} \uplus E_{G_i}$ for $i \in \{1, 2\}$ is given by $f'_{V_G} = f_V \uplus f_E$,

$f'_{E_G} : E_G^{G_1} \rightarrow E_G^{G_2}$ with $E_G^{G_i} = E_{n2e}^{G_i} \uplus E_{e2n}^{G_i}$ for $i \in \{1, 2\}$ is given by

$$f'_{E_G}(v, e, n) = (f_V(v), f_E(e), n) \text{ for } (v, e, n) \in E_{n2e}^{G_1},$$

$$f'_{E_G}(e, v, n) = (f_E(e), f_V(v), n) \text{ for } (e, v, n) \in E_{e2n}^{G_1},$$

$f'_{E_{NA}} : E_{NA}^{G_1} \rightarrow E_{NA}^{G_2}$ with $E_{NA}^{G_i} = E_{in}^{G_i} \uplus E_{out}^{G_i}$ for $i \in \{1, 2\}$ is given by

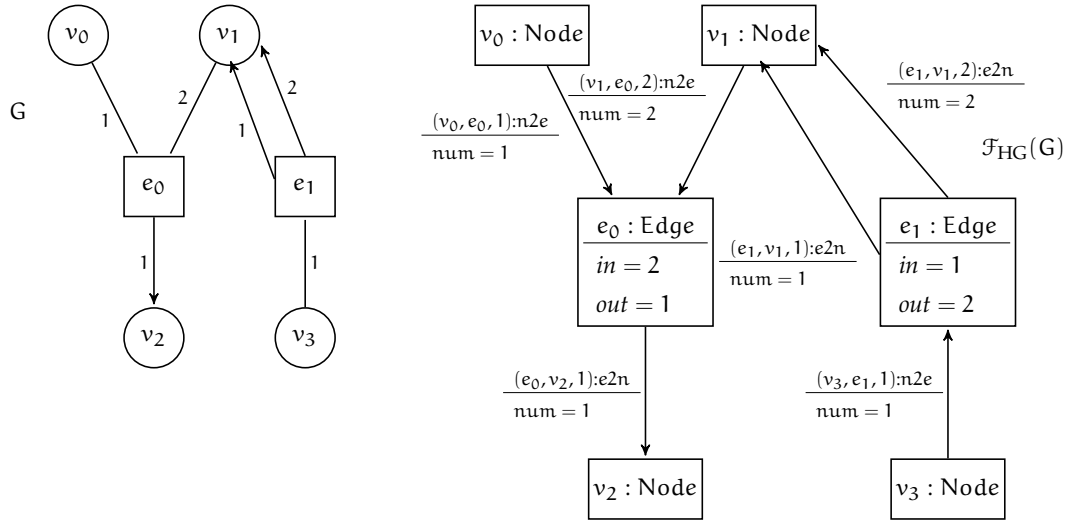
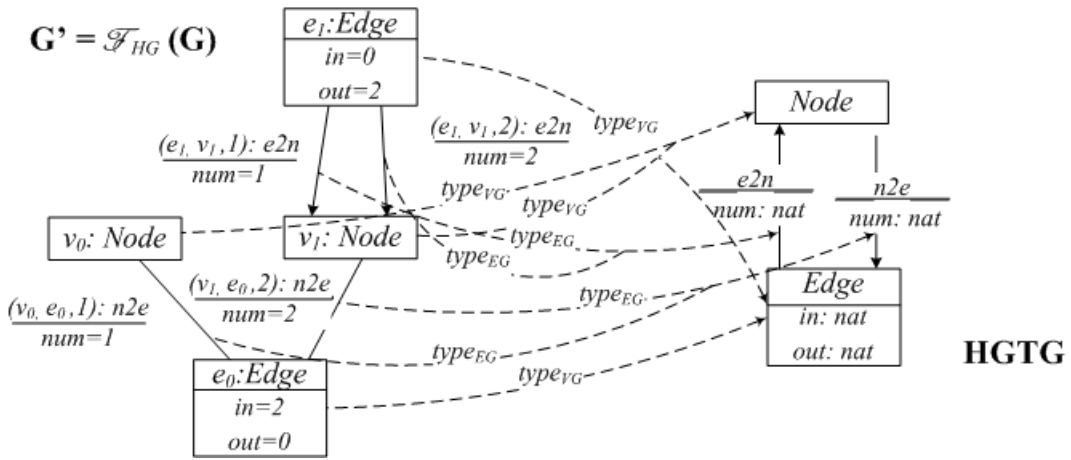
$$f'_{E_{NA}}(e, n, x) = (f_E(e), n, x) \text{ for } (e, n, x) \in E_{in}^{G_1} \uplus E_{out}^{G_1} \text{ and } x \in \{in, out\},$$

$f'_{E_{EA}} : E_{EA}^{G_1} \rightarrow E_{EA}^{G_2}$ with $E_{EA}^{G_i} = E_s^{G_i} \uplus E_t^{G_i}$ for $i \in \{1, 2\}$ is given by

$$f'_{E_{EA}}(n, v, e) = (n, f_V(v), f_E(e)) \text{ for } (n, v, e) \in E_s^{G_1} \text{ and } n \in \mathbb{N},$$

$$f'_{E_{EA}}(n, e, v) = (n, f_E(e), f_V(v)) \text{ for } (n, e, v) \in E_t^{G_1} \text{ and } n \in \mathbb{N}.$$

² Where $E_s^{G'} \cong E_{n2e}^{G'}$ and $E_t^{G'} \cong E_{e2n}^{G'}$.

Figure 30: Hypergraph G and its corresponding typed attributed graph $\mathcal{F}_{HG}(G)$ Figure 31: Example for the $\mathbf{AGraphs}_{HGTG}$ -morphism $type$ components $type_{VG}$ and $type_{EG}$

An example for using the functor \mathcal{F}_{HG} on objects is shown in Figure 30 where the typed attributed graph to the right is the translation of the corresponding hypergraph to the left. As usual in the hypergraph notation, only the target nodes of a hyperedge are marked by arrows. Another example for the translation of objects by \mathcal{F}_{HG} is given in Figure 34 where four depicted typed attributed graphs result from the translation of the corresponding hypergraphs in Figure 33.

An example for the $type$ -morphism components $type_{VG}$ and $type_{EG}$ (depicted by dashed arrows) is given in Figure 31. We do not show the other three $type$ -morphism components $type_{VD}$, $type_{ENA}$, and $type_{EEA}$ in order to improve the clarity of the illustration.

An example for the morphism translation by \mathcal{F}_{HG} is shown in Figure 32. For more examples of the morphism translation see again Figure 33 and Figure 34 where the morphisms in Figure 34 are translations of the corresponding morphisms in Figure 33. It is important to ensure that the translation of morphisms, defined in the way described before, is well-defined for all hypergraph morphisms $f : G_1 \rightarrow G_2$. The formulation of the sufficient properties for the well-definedness of the morphism translation is the aim of the next lemma.

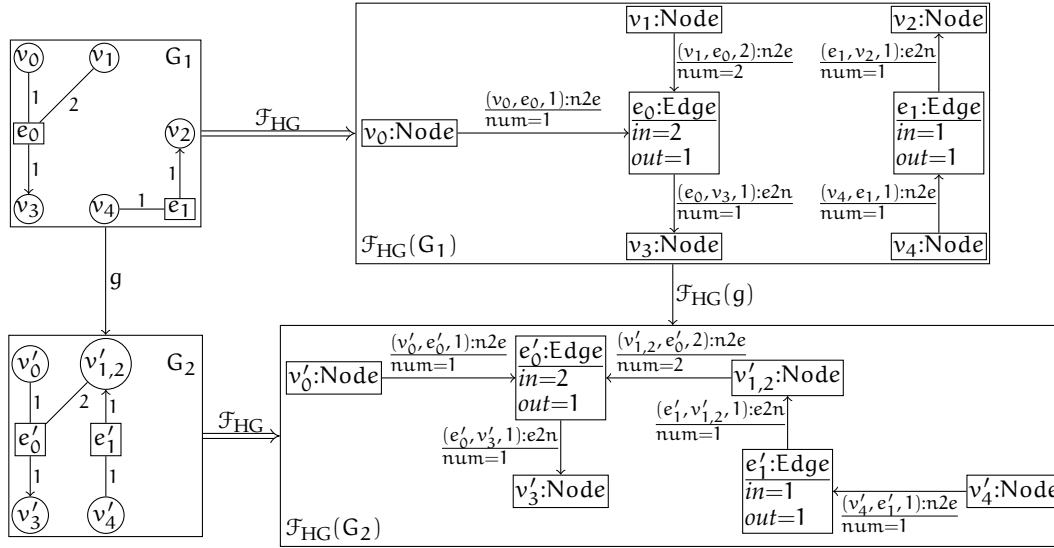


Figure 32: Translation of a hypergraph morphism g into the corresponding typed attributed graph morphism $\mathcal{F}_{\text{HG}}(g)$

Lemma 27 (Well-Definedness of Hypergraph Morphism Translation).

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, and the functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Then for each hypergraph morphism $f : G_1 \rightarrow G_2$ the corresponding typed attributed graph morphism $\mathcal{F}_{\text{HG}}(f) : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ is well-defined in $\mathbf{AGraphs}_{\text{HGTG}}$. Moreover, \mathcal{F}_{HG} preserves compositionality, injective morphisms, inclusions, and identities.

Proof.

The detailed proof for this lemma is given in [Appendix C](#) on [page 322](#) showing the following steps:

1. The components of $\mathcal{F}_{\text{HG}}(f)$ are well-defined w.r.t. the codomain.
2. The components of $\mathcal{F}_{\text{HG}}(f)$ are compatible with the source and target functions.
3. The components of $\mathcal{F}_{\text{HG}}(f)$ are compatible with the typing morphisms.
4. Compositionality axiom holds for \mathcal{F}_{HG} .
5. $f \in \mathcal{M}_1$ (inclusion, identity) implies that $\mathcal{F}_{\text{HG}}(f) \in \mathcal{M}_2$ (inclusion, identity).

□

Using Lemma 27 we obtain that \mathcal{F}_{HG} is a functor, because \mathcal{F}_{HG} is well-defined as well as preserves identities and composition.

BEHAVIOR ANALYSIS OF HYPERGRAPH TRANSFORMATION SYSTEMS

In this chapter, we show that we can apply the theoretical results concerning translation and creation of rule applicability without or with nested application conditions, (direct) transformations, parallel and sequential independence of transformations as well as \mathcal{F} -bisimilarity and \mathcal{F} -transfer of bisimilarity to the concrete functor \mathcal{F}_{HG} between the categories of hypergraphs and typed attributed graphs. For this reason, we show in Section 6.1 that \mathcal{F}_{HG} is an \mathcal{M} -functor as well as in Sections 6.2 and 6.3 that the remaining requirements of the theoretical results from Chapter 3 are satisfied by \mathcal{F}_{HG} leading altogether to the instantiation of the mentioned theoretical results for hypergraph transformation systems. Moreover, the instantiation of these results guarantees the behavioral equivalence of the \mathcal{F}_{HG} -related parts of the source and the target transformation systems.

6.1 TRANSLATION OF HYPERGRAPH TRANSFORMATIONS

Our first step in this section is to show the preservation of pushouts along injective morphisms by the functor \mathcal{F}_{HG} . Using this result together with the preservation of injective morphisms by \mathcal{F}_{HG} , which was already shown in Lemma 27 in the previous chapter, we obtain that \mathcal{F}_{HG} is an \mathcal{M} -functor and thus we can show that \mathcal{F}_{HG} translates applicability of hypergraph rules without or with nested application conditions, (direct) hypergraph transformations as well as parallel and sequential independence of hypergraph transformations by application of the general theory.

We can apply the results of our general theory to \mathcal{F}_{HG} only if \mathcal{F}_{HG} is an \mathcal{M} -functor. The remaining technical property that has to be shown for \mathcal{F}_{HG} to become an \mathcal{M} -functor, is the preservation of pushouts along injective morphisms, which intuitively means that if we have a pushout along injective hypergraph morphisms in the category **HyperGraphs** then, applying \mathcal{F}_{HG} to this diagram, we obtain a pushout along injective typed attributed graph morphisms in the category **AGraphs_{HGTG}**. As we already know from Definition 41, each functor, which is an \mathcal{M} -functor, has to satisfy this technical property.

Lemma 28 (\mathcal{F}_{HG} Preserves Pushouts along Injective Morphisms [211]).

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ introduced in Definition 63, hypergraphs G_i for $i \in \{0, 1, 2, 3\}$ with hypergraph morphisms $b = (b_V, b_E)$, $c = (c_V, c_E)$, $g = (g_V, g_E)$, $h = (h_V, h_E)$, and typed attributed graphs $\mathcal{F}_{\text{HG}}(G_i)$ for $i \in \{0, 1, 2, 3\}$ with typed attributed graph morphisms $\mathcal{F}_{\text{HG}}(b) = b' = (b'_{V_G}, b'_{V_D}, b'_{E_G}, b'_{E_{NA}}, b'_{E_{EA}})$, $\mathcal{F}_{\text{HG}}(c) = c' = (c'_{V_G}, c'_{V_D}, c'_{E_G}, c'_{E_{NA}}, c'_{E_{EA}})$, $\mathcal{F}_{\text{HG}}(g) = g' = (g'_{V_G}, g'_{V_D}, g'_{E_G}, g'_{E_{NA}}, g'_{E_{EA}})$, $\mathcal{F}_{\text{HG}}(h) = h' = (h'_{V_G}, h'_{V_D}, h'_{E_G}, h'_{E_{NA}}, h'_{E_{EA}})$. If (1) is a pushout in **HyperGraphs** with $b \in \mathcal{M}_1$ then we have that (2) is a pushout in **AGraphs_{HGTG}** with $\mathcal{F}_{\text{HG}}(b) \in \mathcal{M}_2$.

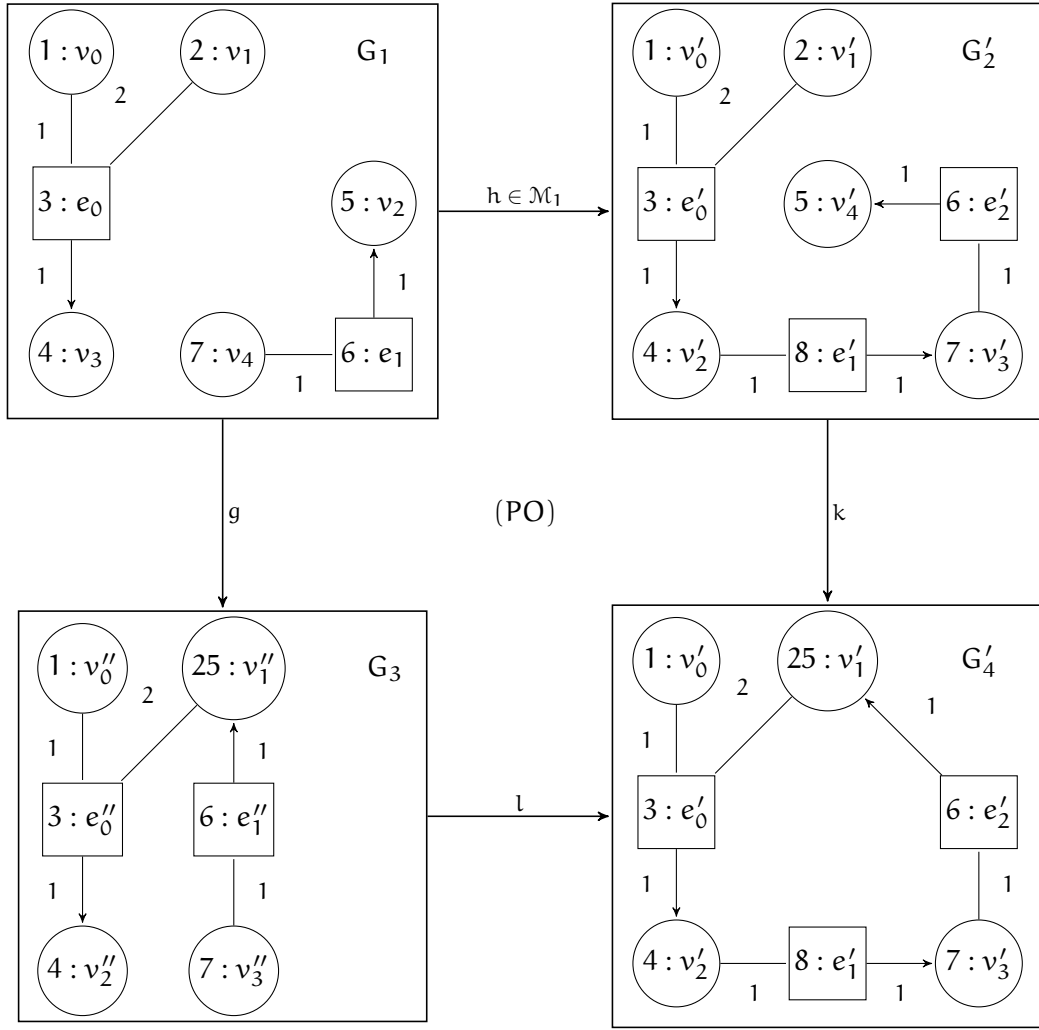


Figure 33: Pushout in **HyperGraphs** with an injective hypergraph morphism $h : G_1 \rightarrow G'_2$

$$\begin{array}{ccc}
 G_0 & \xrightarrow{b} & G_1 \\
 c \downarrow & (1) & \downarrow g \\
 G_2 & \xrightarrow{h} & G_3
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathcal{F}_{\text{HG}}(G_0) & \xrightarrow{\mathcal{F}_{\text{HG}}(b)=b'} & \mathcal{F}_{\text{HG}}(G_1) \\
 \mathcal{F}_{\text{HG}}(c)=c' \downarrow & (2) & \downarrow \mathcal{F}_{\text{HG}}(g)=g' \\
 \mathcal{F}_{\text{HG}}(G_2) & \xrightarrow{\mathcal{F}_{\text{HG}}(h)=h'} & \mathcal{F}_{\text{HG}}(G_3)
 \end{array}$$

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 328](#). □

An example for the translation of pushouts along injective hypergraph morphisms is given in [Figure 33](#) and [Figure 34](#) where a pushout in **HyperGraphs** is given in [Figure 33](#) and the corresponding translated pushout in **AGraphs_{HGTG}** is given in [Figure 34](#).

After the validation of the sufficient property for the translation of rule applicability, (direct) hypergraph transformations as well as parallel and sequential independence, we now formulate and prove our next theorem, which is one of the main results concerning our hypergraph application. This theorem states intuitively that transformation steps,

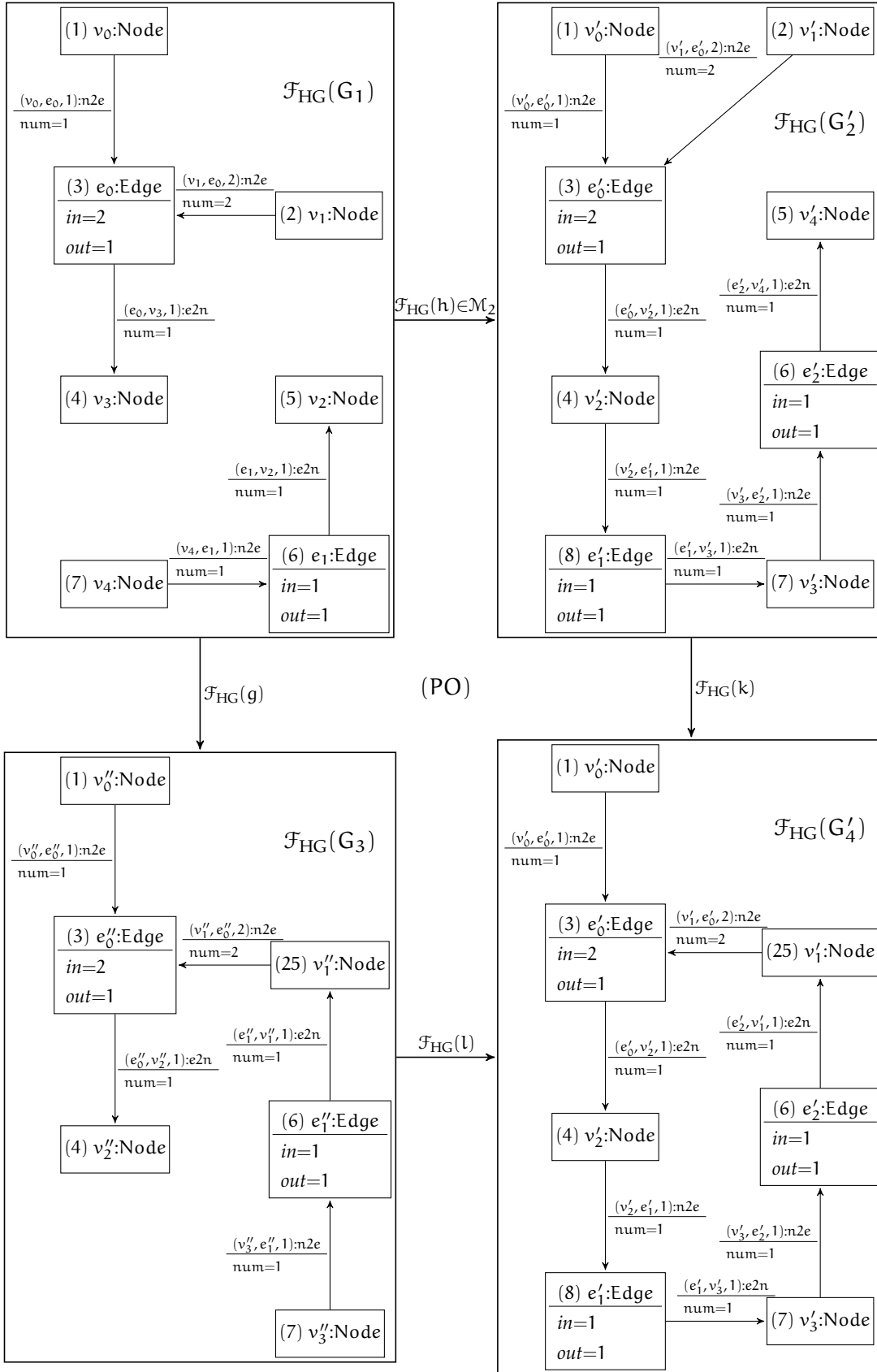


Figure 34: Translated pushout in $\mathbf{AGraphs}_{\mathbf{HGTG}}$ with an injective typed attributed graph morphism $\mathcal{F}_{\mathbf{HG}}(h) : \mathcal{F}_{\mathbf{HG}}(G_1) \rightarrow \mathcal{F}_{\mathbf{HG}}(G'_2)$

which are possible in a hypergraph transformation system, are also possible in the corresponding typed attributed graph transformation system using our concrete functor \mathcal{F}_{HG} . Note that the following theorem is applicable to hypergraph transformation systems containing rules without or with nested application conditions.

Theorem 14 (Translation of Hypergraph Transformations into Typed Attributed Graph Transformations [211]).

Consider \mathcal{M} -adhesive transformation systems $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$ and $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P))$. The functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ introduced in Definition 63 translates applicability of hypergraph rules without or with nested application conditions, construction of (direct) hypergraph transformations as well as parallel and sequential independence of hypergraph transformations.

Proof.

In order to use the results from Theorems 1 and 7 for the case of transformations without nested application conditions as well as Theorems 3 and 10 for the case of transformations with nested application conditions, we only have to show that \mathcal{F}_{HG} is an \mathcal{M} -functor according to Definition 41. In fact, we have $\mathcal{F}_{\text{HG}}(\mathcal{M}_1) \subseteq \mathcal{M}_2$, i.e., \mathcal{F}_{HG} preserves injectivity of morphisms by Lemma 27 and \mathcal{F}_{HG} preserves pushouts along injective hypergraph morphisms according to Lemma 28. \square

In the following, we show for our running example how hypergraph transformation rules can be translated into the corresponding typed attributed graph rules using the \mathcal{M} -functor \mathcal{F}_{HG} as well as give an example for a translated transformation step and satisfaction of a translated nested application condition.

Example 14 (Hypergraph Transformations Translated by \mathcal{F}_{HG}).

In this example, we consider again the hypergraph transformation system describing a simple distributed system with mobility already introduced in Example 7.

Since the \mathcal{M} -functor \mathcal{F}_{HG} introduced in Definition 63 is not capable to handle the hyperedge labels used in the context of the example, we have first to extend \mathcal{F}_{HG} to the translation of the hyperedge labels into the `String` attributes of the corresponding hyperedge node representation.

In Figure 38 and Figure 39 the transformation rules from Example 7 translated by \mathcal{F}_{HG} are given for the hypergraph transformation system without and with nested application conditions, respectively. Note that for better readability we indicate in both figures only the edge attribute `num` on graph edges avoiding to include the typing of the edge as well. Furthermore, Figure 37 shows a translated transformation step for the application of the translated rule $\mathcal{F}_{\text{HG}}(\text{enterServer})$, while the corresponding hypergraph transformation step for the application of a hypergraph transformation rule `enterServer` is given in Figure 35.

Moreover, in Figure 36 we give an example for satisfaction of the NAC of the translated rule $\mathcal{F}_{\text{HG}}(\text{enterServer})$ from Figure 39. In the upper part of the picture the considered NAC is given where $\mathcal{F}_{\text{HG}}(\text{L}_{\text{enterServer}})$ represents the left-hand side of the translated rule $\mathcal{F}_{\text{HG}}(\text{enterServer})$, while the lower part of the figure depicts the translated graph for that we want to check the satisfaction. As we already know from Subsection 3.1.2, a translated NAC is satisfied when there is no injective morphism from the NAC into the corresponding translated graph making the arising triangle commute. Obviously, this is the case for our example, because there is no morphism $q' : \mathcal{F}_{\text{HG}}(C) \rightarrow \mathcal{F}_{\text{HG}}(G)$ (depicted by the scratched dashed arrow) making the depicted triangle commute. Hence, we have that the match morphism $\mathcal{F}_{\text{HG}}(m) : \mathcal{F}_{\text{HG}}(\text{L}_{\text{enterServer}}) \rightarrow \mathcal{F}_{\text{HG}}(G)$ satisfies the NAC of the rule $\mathcal{F}_{\text{HG}}(\text{enterServer})$.

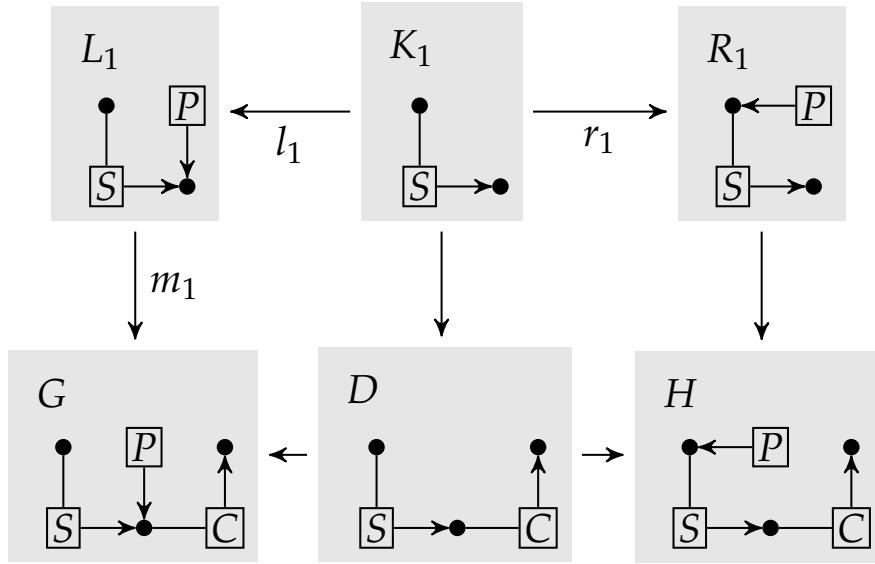
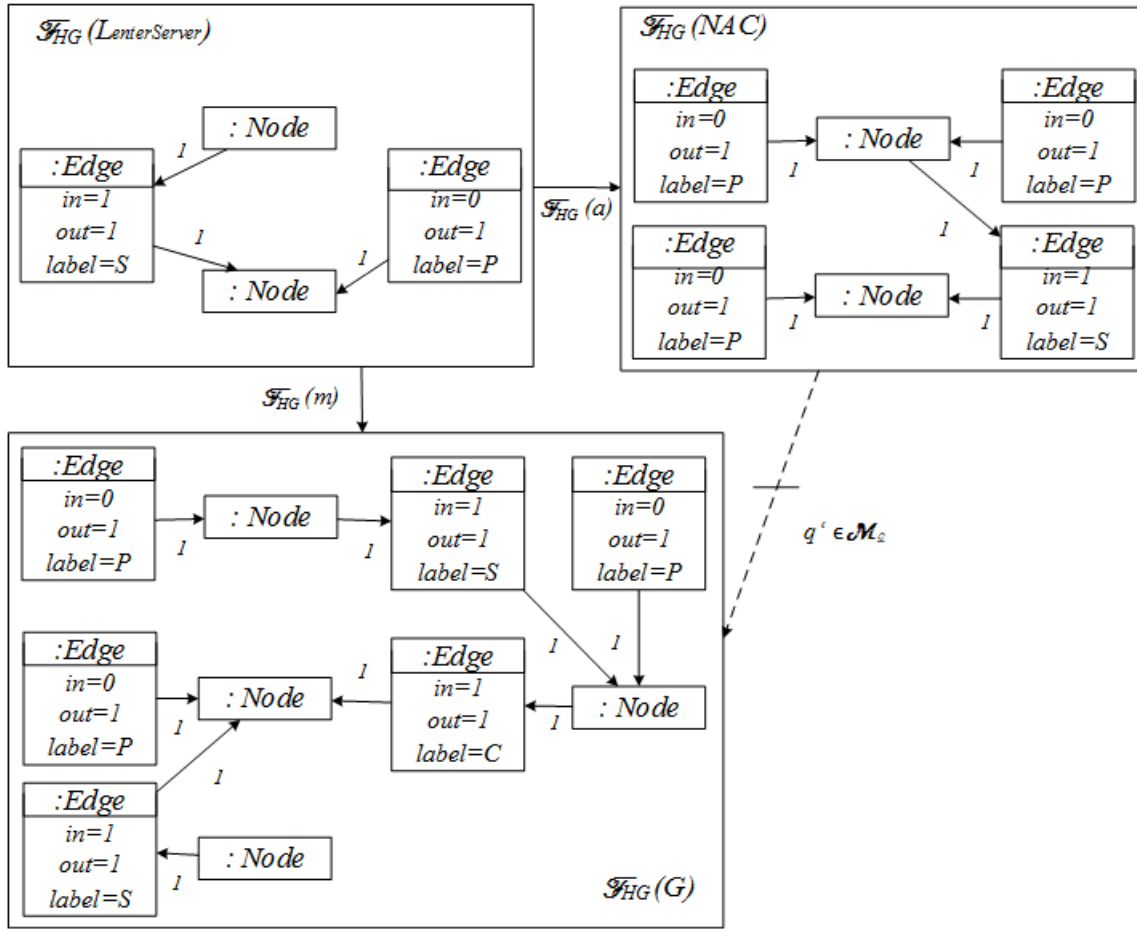
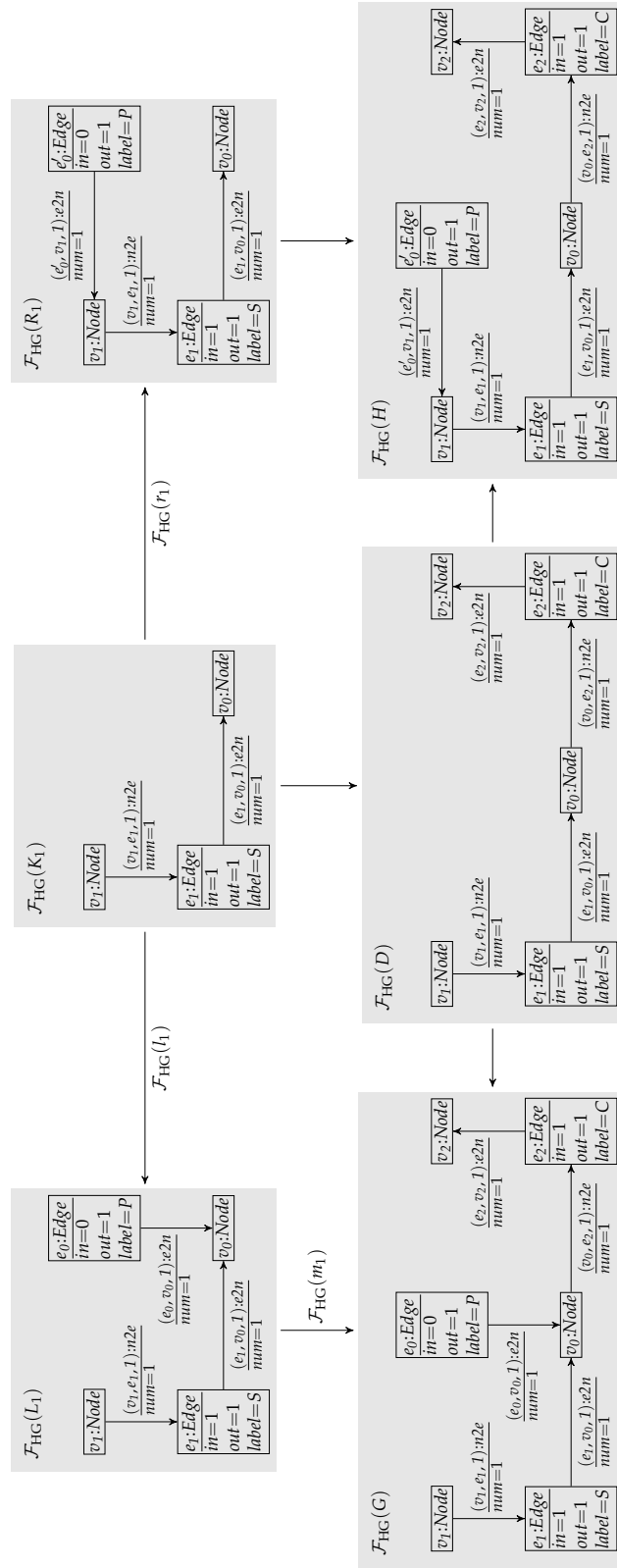


Figure 35: Hypergraph transformation step for the application of the rule enterServer

Figure 36: Satisfaction of the translated NAC of the rule $\mathcal{J}_{\text{HG}}(\text{enterServer})$

Figure 37: Translated transformation step for the application of the rule $\mathcal{J}_{\text{HG}}(\text{enterServer})$

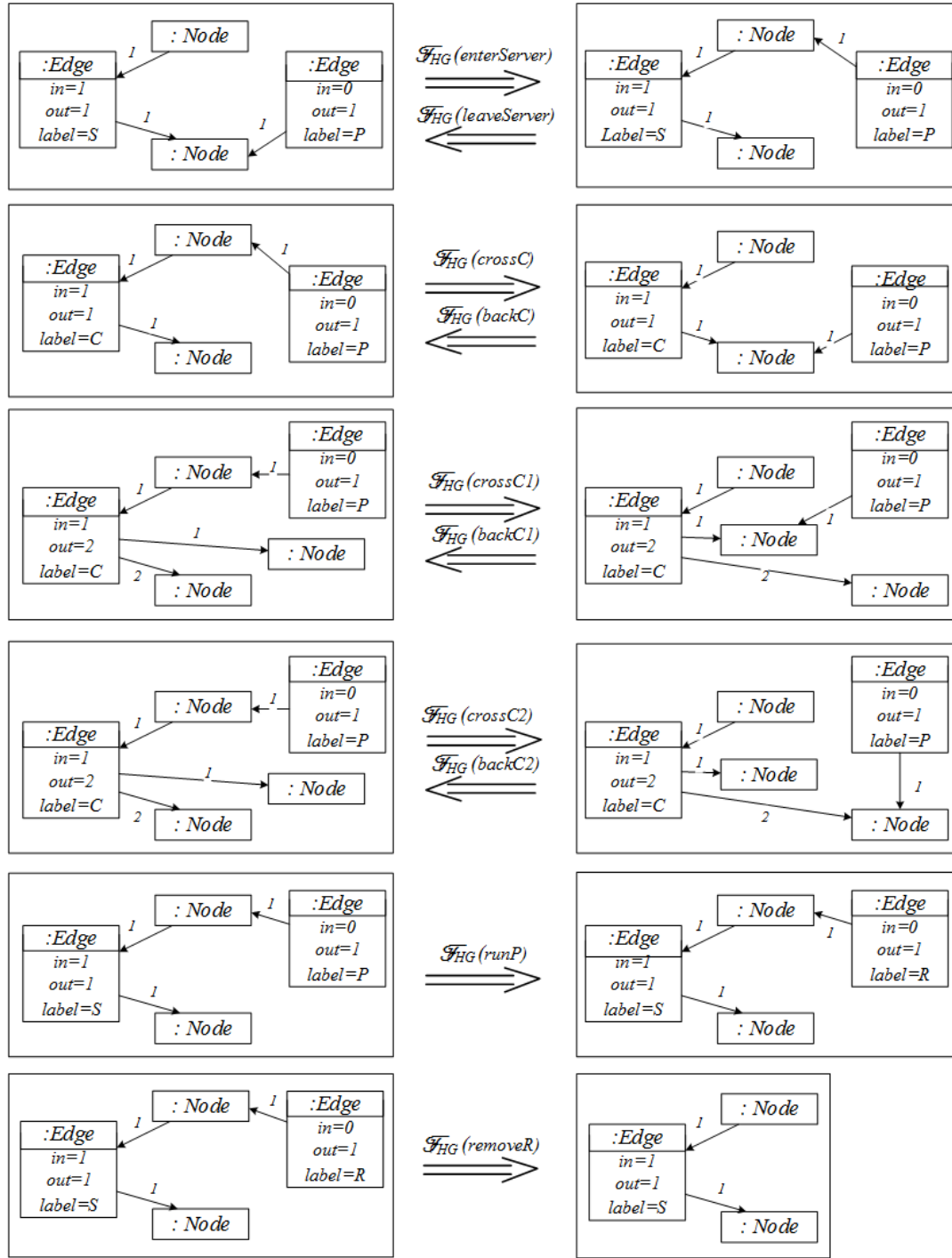


Figure 38: Translated transformation rules without nested application conditions of the Mobile Processes system

6.2 CREATION OF HYPERGRAPH TRANSFORMATIONS

In this section, we are going to verify sufficient properties, which allow for application of the creation part of Theorems 1 and 3 to the \mathcal{M} -functor \mathcal{F}_{HG} introduced in Definition 63. We show as the first step that \mathcal{F}_{HG} creates morphisms and injective morphisms, followed by the proof that \mathcal{F}_{HG} preserves initial pushouts. Subsequently, we formulate and prove our next important result concerning the hypergraph application stating that \mathcal{F}_{HG} creates applicability of hypergraph rules without or with nested application conditions, (direct) hypergraph transformations as well as parallel and sequential independence of hypergraph transformations.

The first of the sufficient properties, which we want to show, is the ability of \mathcal{F}_{HG} to create morphisms. The proof for the uniqueness of morphism creation is based on the additional technical property introduced and shown in the following lemma. This lemma intuitively means that each typed attributed graph morphism $f' : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ is uniquely determined by its V_G -component $f'_{V_G} : V_G^{G_1} \rightarrow V_G^{G_2}$.

Lemma 29 (Uniquely Determined \mathcal{F}_{HG} -Images [211]).

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63, typed attributed graphs $\mathcal{F}_{\text{HG}}(G_1)$, $\mathcal{F}_{\text{HG}}(G_2)$, and a morphism $f' : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ with $f'_{V_D} = \text{id}_N$. Then we have that f' is uniquely determined by the V_G -component $f'_{V_G} : V_G^{G_1} \rightarrow V_G^{G_2}$ with $V_G^{G_i} = V_{G_i} \uplus E_{G_i}$ for $i \in \{1, 2\}$.

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on page 332. □

In the next lemma we state and prove that the \mathcal{M} -functor \mathcal{F}_{HG} creates morphisms. This is the case if morphisms are created uniquely and are well-defined.

Lemma 30 (\mathcal{F}_{HG} Creates Morphisms [211]).

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ and $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, typed attributed graphs $\mathcal{F}_{\text{HG}}(G_1)$ and $\mathcal{F}_{\text{HG}}(G_2)$ as well as a morphism $f' : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ that is compatible with typing morphisms. Then the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ creates a unique morphism $f : G_1 \rightarrow G_2$ such that $\mathcal{F}_{\text{HG}}(f) = f'$ or formally written:

$$\exists! f : G_1 \rightarrow G_2. \mathcal{F}_{\text{HG}}(f) = f'.$$

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on page 333. □

For the case if we consider transformations with nested application conditions, we need additionally to show that \mathcal{F}_{HG} creates injective hypergraph morphisms.

Lemma 31 (\mathcal{F}_{HG} Creates Injective Morphisms [213]).

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, typed attributed graphs $\mathcal{F}_{\text{HG}}(G_1)$, $\mathcal{F}_{\text{HG}}(G_2)$, and an injective typed attributed graph morphism $f' : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ that is compatible with typing morphisms. Then the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs}$

$\rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ creates a unique injective hypergraph morphism $f : G_1 \rightarrow G_2$ such that $\mathcal{F}_{\text{HG}}(f) = f'$ or formally written:

$$\exists! f : G_1 \rightarrow G_2 \text{ in } \mathcal{M}_1. \mathcal{F}_{\text{HG}}(f) = f'.$$

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 335](#). \square

To prove the property that \mathcal{F}_{HG} preserves initial pushouts, we have to construct two special boundary objects $\mathcal{F}_{\text{HG}}(B)$ and B' in the category of typed attributed graphs. $\mathcal{F}_{\text{HG}}(B)$ is an \mathcal{F}_{HG} -translation of a boundary object in the category of hypergraphs, while B' is a boundary object in the category of typed attributed graphs over the hypergraph specific type graph HGTG constructed similar to [Fact 6](#). In the following [Lemmas 32](#) and [33](#), we introduce both mentioned boundary object constructions.

Lemma 32 (Application of \mathcal{F}_{HG} to a Hypergraph Boundary Object B over a Morphism $f : L \rightarrow G$).

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, a boundary object $B = (V_B, E_B, s_B, t_B)$ over a given morphism $f : L \rightarrow G$ in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ constructed according to [Lemma 5](#), and the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from [Definition 63](#). Then the application of \mathcal{F}_{HG} to the boundary object B results in the following typed attributed graph:

$$\begin{aligned} \mathcal{F}_{\text{HG}}(B) = B^* &= ((B_0^*, \text{NAT}), \text{type}^{B^*}) \text{ with} \\ B_0^* &= (V_G^{B_0^*}, V_D^{B_0^*} = \mathbb{N}, E_G^{B_0^*}, E_{\text{NA}}^{B_0^*}, E_{\text{EA}}^{B_0^*}, (s_j^{B_0^*}, t_j^{B_0^*})_{j \in \{G, \text{NA}, \text{EA}\}}) \text{ where} \\ V_G^{B_0^*} &= V_B \uplus E_B = \text{IP}_V \cup \text{IP}_{V_E} \cup \text{DP}_V \cup \text{IP}_E, \\ E_G^{B_0^*} &= E_{n2e}^{B_0^*} \uplus E_{e2n}^{B_0^*} \text{ with} \\ E_{n2e}^{B_0^*} &= \{((v, e), n) \in (V_B \times E_B) \times \mathbb{N} \mid s_B^n(e) = v\}, \\ E_{e2n}^{B_0^*} &= \{((e, v), n) \in (E_B \times V_B) \times \mathbb{N} \mid t_B^n(e) = v\}, \\ E_{\text{NA}}^{B_0^*} &= E_{in}^{B_0^*} \uplus E_{out}^{B_0^*} \text{ with} \\ E_{in}^{B_0^*} &= \{(e, n, in) \mid (e, n) \in E_B \times \mathbb{N} \wedge |s_B(e)| = n\}, \\ E_{out}^{B_0^*} &= \{(e, n, out) \mid (e, n) \in E_B \times \mathbb{N} \wedge |t_B(e)| = n\}, \\ E_{\text{EA}}^{B_0^*} &= E_s^{B_0^*} \uplus E_t^{B_0^*} \text{ with} \\ E_s^{B_0^*} &= \{(n, (v, e)) \in \mathbb{N} \times (V_B \times E_B) \mid s_B^n(e) = v\}, \\ E_t^{B_0^*} &= \{(n, (e, v)) \in \mathbb{N} \times (E_B \times V_B) \mid t_B^n(e) = v\}, \\ s_G^{B_0^*} : E_G^{B_0^*} &\rightarrow V_G^{B_0^*} \text{ defined by } s_G^{B_0^*}((x, y), n) = x, \\ t_G^{B_0^*} : E_G^{B_0^*} &\rightarrow V_G^{B_0^*} \text{ defined by } t_G^{B_0^*}((x, y), n) = y, \\ s_{\text{NA}}^{B_0^*} : E_{\text{NA}}^{B_0^*} &\rightarrow V_G^{B_0^*} \text{ defined by } s_{\text{NA}}^{B_0^*}(e, n, x) = e, \\ t_{\text{NA}}^{B_0^*} : E_{\text{NA}}^{B_0^*} &\rightarrow \mathbb{N} \text{ defined by } t_{\text{NA}}^{B_0^*}(e, n, x) = n, \\ s_{\text{EA}}^{B_0^*} : E_{\text{EA}}^{B_0^*} &\rightarrow E_G^{B_0^*} \text{ defined by } s_{\text{EA}}^{B_0^*}(n, (x, y)) = ((x, y), n), \end{aligned}$$

$t_{EA}^{B_0^*} : E_{EA}^{B_0^*} \rightarrow \mathbb{N}$ defined by $t_{EA}^{B_0^*}(n, (x, y)) = n$,

and $\mathcal{F}_{HG}(b) : \mathcal{F}_{HG}(B) \rightarrow \mathcal{F}_{HG}(L)$ is an inclusion.

$$\begin{array}{ccc} B^* = \mathcal{F}_{HG}(B) & \xrightarrow{\mathcal{F}_{HG}(b)} & \mathcal{F}_{HG}(L) \\ & & \downarrow \mathcal{F}_{HG}(f) \\ & & \mathcal{F}_{HG}(G) \end{array}$$

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 336](#). □

Lemma 33 (Boundary Object in $(\mathbf{AGraphs}_{HG\text{TG}}, \mathcal{M}_2)$ over a Morphism $f' : L' \rightarrow G'$).

Consider a typed attributed graph morphism $f' : L' \rightarrow G'$. The boundary object B' is given by $B' = ((B'_0, \text{NAT}), \text{type}^{B'})$ with the boundary points

$$B'_0 = (V_G^{B'_0}, V_D^{B'_0} = \mathbb{N}, E_G^{B'_0}, E_{NA}^{B'_0}, E_{EA}^{B'_0}, (s_j^{B'_0}, t_j^{B'_0})_{j \in \{G, NA, EA\}}).$$

The components of B'_0 are given by the dangling and identification points as follows:

$$\begin{array}{ccc} B' & \xrightarrow{b'} & L' \\ & & \downarrow f' \\ & & G' \end{array}$$

$$E_{NA}^{B'_0} = \text{IP}_{E_{NA}} = \{a \in E_{NA}^{L'} \mid \exists a' \neq a. a' \in E_{NA}^{L'} \wedge f'_{E_{NA}}(a) = f'_{E_{NA}}(a')\},$$

$$E_{EA}^{B'_0} = \text{IP}_{E_{EA}} = \{a \in E_{EA}^{L'} \mid \exists a' \neq a. a' \in E_{EA}^{L'} \wedge f'_{E_{EA}}(a) = f'_{E_{EA}}(a')\},$$

$$E_G^{B'_0} = \text{DP}_{E_G} \cup \text{IP}_{E_G} \cup s_{EA}^{L'}(\text{IP}_{E_{EA}}) \text{ with}$$

$$\text{DP}_{E_G} = \{a \in E_G^{L'} \mid \exists a' \in E_{EA}^{G'} \setminus f'_{E_{EA}}(E_{EA}^{L'}). f'_{E_G}(a) = s_{EA}^{G'}(a')\},$$

$$\text{IP}_{E_G} = \{a \in E_G^{L'} \mid \exists a' \neq a. a' \in E_G^{L'} \wedge f'_{E_G}(a) = f'_{E_G}(a')\},$$

$$V_G^{B'_0} = \text{DP}_{V_G} \cup \text{IP}_{V_G} \cup s_G^{L'}(E_G^{B'_0}) \cup t_G^{L'}(E_G^{B'_0}) \cup s_{NA}^{L'}(\text{IP}_{E_{NA}}) \text{ with}$$

$$\text{DP}_{V_G} = \{a \in V_G^{L'} \mid [\exists a' \in E_{NA}^{G'} \setminus f'_{E_{NA}}(E_{NA}^{L'}). f'_{V_G}(a) = s_{NA}^{G'}(a')]$$

$$\vee [\exists a' \in E_G^{G'} \setminus f'_{E_G}(E_G^{L'}). f'_{V_G}(a) = s_G^{G'}(a') \vee f'_{V_G}(a) = t_G^{G'}(a')]\},$$

$$\text{IP}_{V_G} = \{a \in V_G^{L'} \mid \exists a' \neq a. a' \in V_G^{L'} \wedge f'_{V_G}(a) = f'_{V_G}(a')\},$$

$$s_G^{B'_0}, t_G^{B'_0} : E_G^{B'_0} \rightarrow V_G^{B'_0} \text{ are restrictions of } s_G^{L'}, t_G^{L'} : E_G^{L'} \rightarrow V_G^{L'},$$

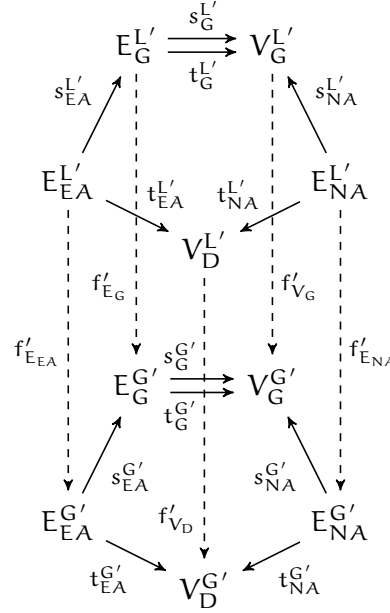
$$s_{NA}^{B'_0} : E_{NA}^{B'_0} \rightarrow V_G^{B'_0} \text{ is a restriction of } s_{NA}^{L'} : E_{NA}^{L'} \rightarrow V_G^{L'},$$

$$t_{NA}^{B'_0} : E_{NA}^{B'_0} \rightarrow \mathbb{N} \text{ is a restriction of } t_{NA}^{L'} : E_{NA}^{L'} \rightarrow \mathbb{N},$$

$$s_{EA}^{B'_0} : E_{EA}^{B'_0} \rightarrow E_G^{B'_0} \text{ is a restriction of } s_{EA}^{L'} : E_{EA}^{L'} \rightarrow E_G^{L'},$$

$$t_{EA}^{B'_0} : E_{EA}^{B'_0} \rightarrow \mathbb{N} \text{ is a restriction of } t_{EA}^{L'} : E_{EA}^{L'} \rightarrow \mathbb{N},$$

and $b' : B' \rightarrow L'$ is an inclusion.



Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 337](#). \square

Note that for the construction of a context object C' in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ we use the general construction that is already introduced in [Fact 7](#) in [Subsection 2.4.1](#).

In [Figure 40](#) we can see, on the one hand, an example for the application of \mathcal{F}_{HG} to the hypergraph boundary object B according to [Lemma 32](#) given by $V_G^{B_0^*} = \{v_3, v_4\}$ and, on the other hand, an example for the construction of the boundary object B' in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ over a general morphism $f' : L' \rightarrow G'$ according to [Lemma 33](#) given by $V_G^{B_0'} = \{v_3, v_4\}$. [Figure 40](#) shows furthermore an example for the construction of the context object C' in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ according to [Fact 7](#) (concerning the boundary object B' constructed according to [Lemma 33](#)) with $V_G^{C_0'} = \{v_2'\}$.

After both sufficient boundary object constructions are introduced in [Lemmas 32](#) and [33](#), we can verify that \mathcal{F}_{HG} preserves initial pushouts as given in the following lemma.

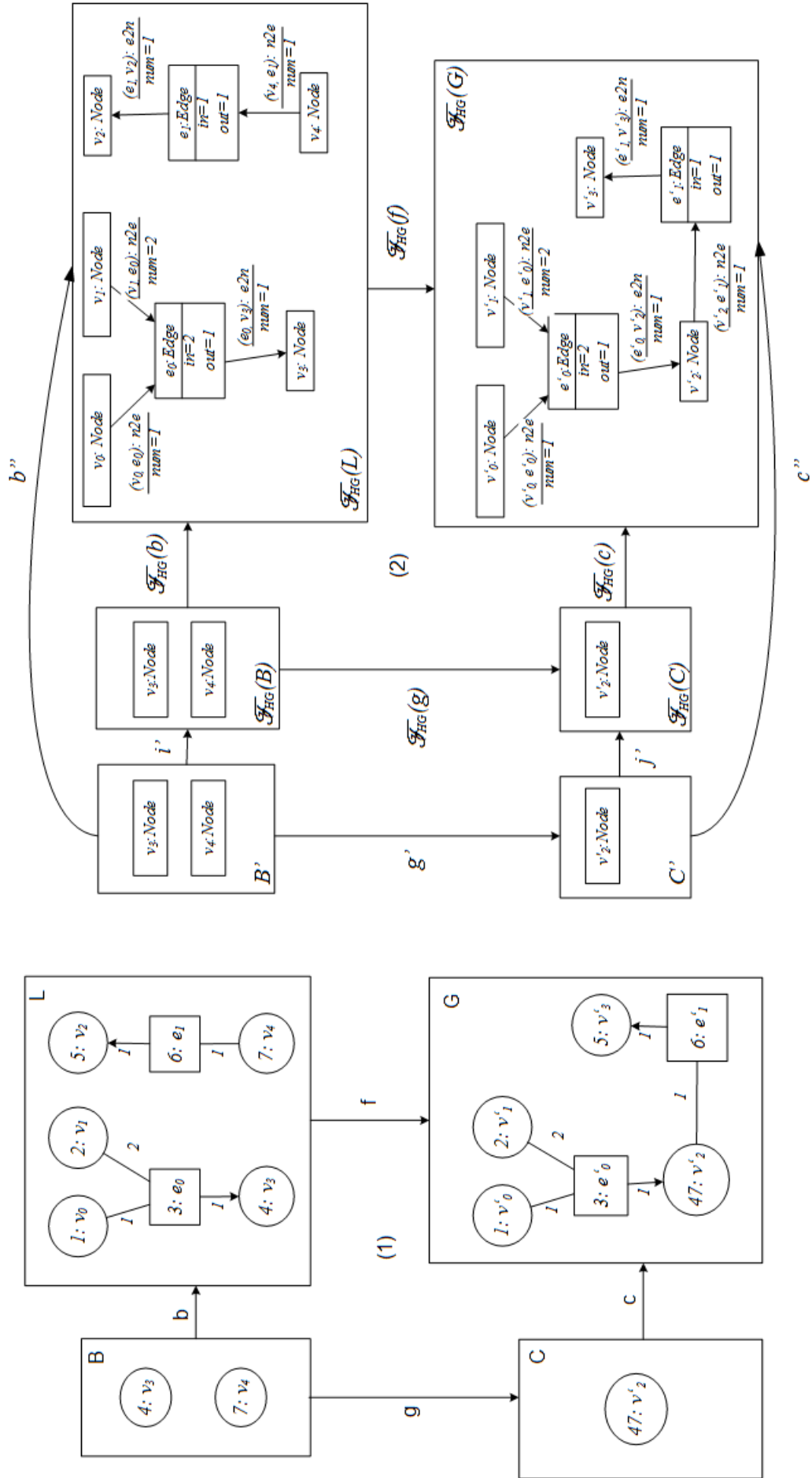
Lemma 34 (\mathcal{F}_{HG} Preserves Initial Pushouts [\[211\]](#)).

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from [Definition 63](#), and let (1) be an initial pushout over $f : L \rightarrow G$ in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$. Then (2) is an initial pushout over $\mathcal{F}_{\text{HG}}(f) : \mathcal{F}_{\text{HG}}(L) \rightarrow \mathcal{F}_{\text{HG}}(G)$ in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$.

$$\begin{array}{ccc}
 B & \xrightarrow{b} & L \\
 \downarrow & (1) & \downarrow f \\
 C & \longrightarrow & G
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{ccc}
 \mathcal{F}_{\text{HG}}(B) & \xrightarrow{\mathcal{F}_{\text{HG}}(b)} & \mathcal{F}_{\text{HG}}(L) \\
 \downarrow & (2) & \downarrow \mathcal{F}_{\text{HG}}(f) \\
 \mathcal{F}_{\text{HG}}(C) & \longrightarrow & \mathcal{F}_{\text{HG}}(G)
 \end{array}$$

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 338](#). \square

Figure 40: Preservation of initial pushouts in $(\mathbf{AGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2)$

An example for the preservation of initial pushouts by \mathcal{F}_{HG} is given in Figure 40 where (1) is an initial pushout in the category of hypergraphs, (2) is the induced pushout over the morphism $\mathcal{F}_{\text{HG}}(f) : \mathcal{F}_{\text{HG}}(L) \rightarrow \mathcal{F}_{\text{HG}}(G)$, and the initial pushout in the category of typed attributed graphs over $\mathcal{F}_{\text{HG}}(f)$ is given by the outer diagram with corners B' , C' , $\mathcal{F}_{\text{HG}}(L)$, and $\mathcal{F}_{\text{HG}}(G)$. Since $i' : B' \rightarrow \mathcal{F}_{\text{HG}}(B)$ and $j' : C' \rightarrow \mathcal{F}_{\text{HG}}(C)$ are isomorphisms, the diagram (2) is an initial pushout over the injective morphism $\mathcal{F}_{\text{HG}}(f)$.

After the satisfaction of all sufficient properties for the creation of rule applicability, (direct) hypergraph transformations as well as parallel and sequential independence of hypergraph transformations is verified, we formulate and prove in the following one of our next main theorems for the hypergraph application. This theorem states intuitively that transformation steps, which are possible in a transformation system translated by \mathcal{F}_{HG} , are also possible in the original hypergraph transformation system. Note that the following theorem is applicable to hypergraph transformation systems containing rules without or with nested application conditions.

Theorem 15 (Creation of Hypergraph Transformations from Typed Attributed Graph Transformations [211]).

Consider \mathcal{M} -adhesive transformation systems $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$ and $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P))$. The functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63 creates applicability of hypergraph rules without or with nested application conditions, construction of (direct) hypergraph transformations as well as parallel and sequential independence of hypergraph transformations.

Proof.

In order to use the results from Theorems 1 and 7 for the case of transformations without nested application conditions as well as Theorems 3 and 10 for the case of transformations with nested application conditions, we have to show that \mathcal{F}_{HG} is an \mathcal{M} -functor according to Definition 41, \mathcal{F}_{HG} creates morphisms as well as injective morphisms, and \mathcal{F}_{HG} preserves initial pushouts. Furthermore, it is required that the category $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ has initial pushouts, which is satisfied according to Lemma 7 in Subsection 2.4.2. In fact, we have that $\mathcal{F}_{\text{HG}}(\mathcal{M}_1) \subseteq \mathcal{M}_2$, i.e., \mathcal{F}_{HG} preserves injectivity of morphisms by Lemma 27 and \mathcal{F}_{HG} preserves pushouts along injective morphisms according to Lemma 28. Moreover, creation of morphisms by \mathcal{F}_{HG} is shown in Lemma 30, creation of injective morphisms by \mathcal{F}_{HG} is shown in Lemma 31, and preservation of initial pushouts by \mathcal{F}_{HG} is shown in Lemma 34. \square

For an example of the creation of hypergraph transformation steps consider again Figure 35 and Figure 37 from the last section. For the translated transformation step given in Figure 37, the result of the creation is the hypergraph transformation step depicted in Figure 35.

6.3 \mathcal{F}_{HG} -TRANSFER OF BISIMILARITY FOR HYPERGRAPH TRANSFORMATION SYSTEMS

In this section, we instantiate our general theoretical results on bisimulation between \mathcal{M} -adhesive transformation systems to our concrete setting of hypergraph transformation systems. That is, we show that the requirements of the respective Theorems 4 and 5 from Section 3.2 are satisfied by the concrete \mathcal{M} -functor \mathcal{F}_{HG} and the categories of hypergraphs and typed attributed graphs. The proof then depends on the various lemmas introduced in earlier sections.

Theorem 16 (\mathcal{F}_{HG} -Bisimilarity and \mathcal{F}_{HG} -Transfer of R-Bisimilarity).

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{HyperGraphs}, \mathcal{M}_1, P_1)$, $\text{AS}_2 = (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P_1))$, $\text{AS}_3 = (\mathbf{HyperGraphs}, \mathcal{M}_1, P_2)$, $\text{AS}_4 = (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P_2))$, a rule relation $R \subseteq P_1 \times P_2$, and the functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ introduced in Definition 63. Then the following two statements hold:

- Let G be a hypergraph from AS_1 . Then G is \mathcal{F}_{HG} -bisimilar to the corresponding typed attributed graph $\mathcal{F}_{\text{HG}}(G)$ in AS_2 , written $G \sim_{\mathcal{F}_{\text{HG}}} \mathcal{F}_{\text{HG}}(G)$.
- Let G be a hypergraph from AS_1 and G' be a hypergraph from AS_3 . Then G is R-bisimilar to G' iff $\mathcal{F}_{\text{HG}}(G)$ is $\mathcal{F}_{\text{HG}}(R)$ -bisimilar to $\mathcal{F}_{\text{HG}}(G')$, written $(G \sim_R G') \Leftrightarrow (\mathcal{F}_{\text{HG}}(G) \sim_{\mathcal{F}_{\text{HG}}(R)} \mathcal{F}_{\text{HG}}(G'))$.

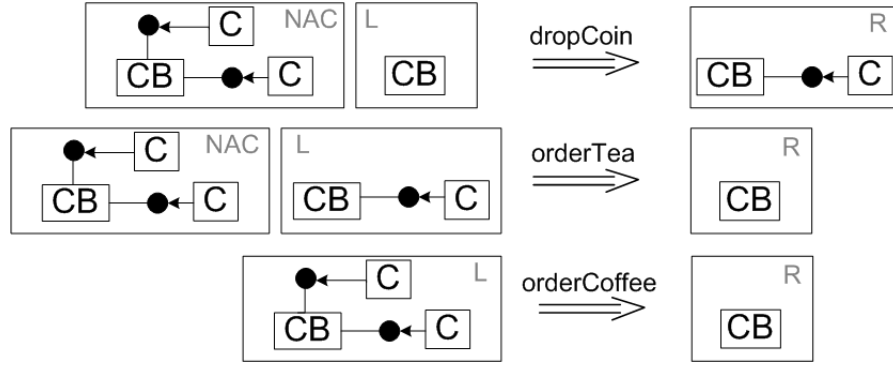
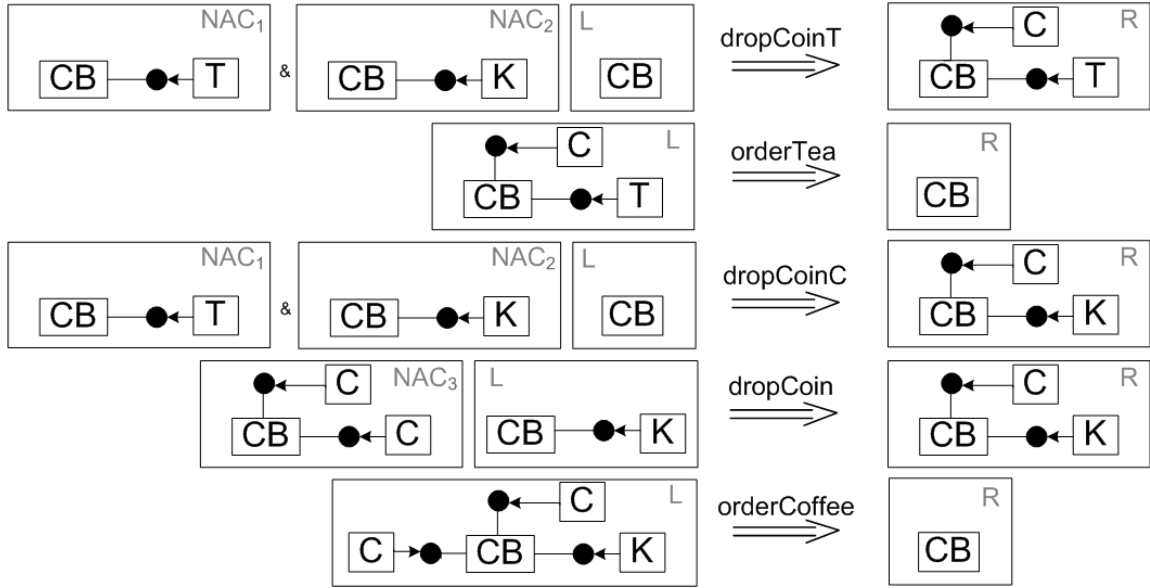
Proof.

In order to use the results from Theorems 4 and 5 for the case of transformation steps without or with nested application conditions, we have to show that \mathcal{F}_{HG} is an \mathcal{M} -functor according to Definition 41, \mathcal{F}_{HG} creates morphisms as well as injective morphisms, and \mathcal{F}_{HG} preserves initial pushouts. Furthermore, it is required that the category $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ has initial pushouts, which is satisfied according to Lemma 7 in Subsection 2.4.2. In fact, we have that $\mathcal{F}_{\text{HG}}(\mathcal{M}_1) \subseteq \mathcal{M}_2$, i.e., \mathcal{F}_{HG} preserves injectivity of morphisms by Lemma 27 and \mathcal{F}_{HG} preserves pushouts along injective morphisms according to Lemma 28. Moreover, creation of morphisms by \mathcal{F}_{HG} is shown in Lemma 30, creation of injective morphisms by \mathcal{F}_{HG} is shown in Lemma 31, and preservation of initial pushouts by \mathcal{F}_{HG} is shown in Lemma 34. \square

In the following, we give an example demonstrating \mathcal{F}_{HG} -bisimilarity and \mathcal{F}_{HG} -transfer of R-bisimilarity for concrete hypergraph transformation systems. We are not considering here our running example on mobile processes since the example used in the following is more appropriate to cover both bisimilarity concepts at once. Nevertheless, for our Mobile Processes example it obviously holds that for each hypergraph derivable by rule application in the original hypergraph transformation system, there is an \mathcal{F}_{HG} -bisimilar typed attributed graph in the corresponding typed attributed graph transformation system implying the behavioral equivalence of the \mathcal{F}_{HG} -related parts of the involved source and target transformation systems.

Example 15 (*Coffee-Tea Machines*).

We consider a classical example of two coffee-tea machines that are functioning differently. Both of the considered machines are equipped with a coin box, in which it is necessary to drop one coin to be able to order a tea or two coins to be able to order a coffee. Furthermore, it is intended for both machines that it is not possible to drop more than two coins successively into a machine without making an order. We represent the possible internal steps of each machine by the corresponding transformation rules. The general behavior of the two machines is as follows: The decision to order tea resp. coffee for the first machine occurs after the dropping of one or two coins into the machine. If only one coin is dropped, only a tea order is possible. By dropping one coin more, it is possible to order a coffee, while a tea order is not possible anymore. For the second machine the beverage decision is made non-deterministically by the machine when the first coin is dropped into the machine. After the decision was made, no switching is possible anymore. If the machine decided to allow a tea order, the machine is waiting for the order to take place just after the dropping of the

Figure 41: Hypergraph transformation rules modeling the behavior of the Coffee-Tea system CT_1 Figure 42: Hypergraph transformation rules modeling the behavior of the Coffee-Tea system CT_2

first coin, otherwise the machine is waiting for the dropping of the second coin, before allowing the ordering of a coffee.

Formally, we model the behavior of the mentioned coffee-tea machines by two different Coffee-Tea transformation systems $CT_1 = (\mathbf{HyperGraphs}, \mathcal{M}_1, P_1)$, $CT_2 = (\mathbf{HyperGraphs}, \mathcal{M}'_1, P_2)$ with $\mathcal{M}_1 = \mathcal{M}'_1$ a class of all injective hypergraph morphisms and sets of hypergraph transformation rules P_1, P_2 . In our machine models, a coin box, coins as well as a decision for a tea resp. coffee order are represented as labeled hyperedges. The meaning of the hyperedge labels is as follows: CT stands for a coin box, C denotes a single coin, a decision for a tea order is represented by the label T, while a decision to order a coffee is represented by the label K.

The behavior of both transformation systems is modeled by the corresponding hypergraph transformation rules given in Figure 41 and Figure 42 for CT_1 and CT_2 , respectively. The transformation rules of the first coffee-tea machine mean the following: The rule dropCoin represents the dropping of a coin into the machine, allowed only if less than two coins are already inside. This restriction is modeled as a NAC for the rule dropCoin. The rules orderTea and orderCoffee allow to order the desired beverage. Note that the tea order is not possible anymore if already two coins are inside the machine, which is given by a NAC for the rule orderTea.

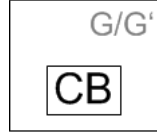


Figure 43: A hypergraph representing the initial state of the Coffee-Tea systems CT_1 and CT_2

For the second coffee-tea machine we have the following rules: The rules `dropCoinT` and `dropCoinC` model, on the one hand, the dropping of one coin into the machine and mark, on the other hand, a beverage decision made non-deterministically by the machine associating hyperedges labeled by T resp. K to the coin box. Both rules are applicable only if no earlier beverage decision already exists. This restriction is given as a conjunction of NAC_1 and NAC_2 for both mentioned rules. The rule `dropCoin` allows for the dropping of the second coin if the machine decided previously to allow a coffee order. Also for the second machine the dropping of the second coin is only possible if less than two coins are already contained in the coin box, which is modeled by NAC_3 for the rule `dropCoin`. The rules `orderTea` and `orderCoffee` allow similar to the first machine for the ordering of the preselected beverage.

Let the hypergraph consisting only of a hyperedge labeled by CB as given in Figure 43 be the initial state for both transformation systems (G for CT_1 and G' for CT_2). Now we want to check whether G is R -bisimilar to G' with $R = \{(\text{dropCoin}, \text{dropCoinT}), (\text{dropCoin}, \text{dropCoinC}), (\text{dropCoin}, \text{dropCoin}), (\text{orderTea}, \text{orderTea}), (\text{orderCoffee}, \text{orderCoffee})\}$. Consider additionally the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Using \mathcal{F}_{HG} we can translate the hypergraph transformation systems CT_1, CT_2 into the typed attributed graph transformation systems $\text{CT}_3 = (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P_1))$, $\text{CT}_4 = (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}'_2, \mathcal{F}_{\text{HG}}(P_2))$ with translated rules $\mathcal{F}_{\text{HG}}(P_1), \mathcal{F}_{\text{HG}}(P_2)$ depicted in an abbreviated form¹ in Figure 44 and Figure 45, respectively. Furthermore, we can translate the start hypergraphs G and G' into the corresponding start typed attributed graphs $\mathcal{F}_{\text{HG}}(G), \mathcal{F}_{\text{HG}}(G')$ such that we additionally have that G is \mathcal{F}_{HG} -bisimilar to $\mathcal{F}_{\text{HG}}(G)$ and G' is \mathcal{F}_{HG} -bisimilar to $\mathcal{F}_{\text{HG}}(G')$ by Theorem 4.

As the next step we want to check whether the typed attributed graphs $\mathcal{F}_{\text{HG}}(G)$ and $\mathcal{F}_{\text{HG}}(G')$ are $\mathcal{F}_{\text{HG}}(R)$ -bisimilar. From Figure 46, which shows an abbreviated form¹ of the reachability graphs of CT_3 and CT_4 depicting all possible transformation steps of CT_3 and CT_4 beginning with the start graphs $\mathcal{F}_{\text{HG}}(G)$ and $\mathcal{F}_{\text{HG}}(G')$, respectively, we can see that $\mathcal{F}_{\text{HG}}(G)$ and $\mathcal{F}_{\text{HG}}(G')$ are not $\mathcal{F}_{\text{HG}}(R)$ -bisimilar. This is the case since in every $\mathcal{F}_{\text{HG}}(R)$ -bisimulation containing $(\mathcal{F}_{\text{HG}}(G), \mathcal{F}_{\text{HG}}(G'))$ the first $\mathcal{F}_{\text{HG}}(\text{dropCoin})$ -step of $\mathcal{F}_{\text{HG}}(G)$ is simulated by an $\mathcal{F}_{\text{HG}}(\text{dropCoinT})$ - or $\mathcal{F}_{\text{HG}}(\text{dropCoinC})$ -step of $\mathcal{F}_{\text{HG}}(G')$; if it is simulated by the $\mathcal{F}_{\text{HG}}(\text{dropCoinT})$ -step, then afterwards the second $\mathcal{F}_{\text{HG}}(\text{dropCoin})$ -step of $\mathcal{F}_{\text{HG}}(G)$ cannot be simulated by $\mathcal{F}_{\text{HG}}(G')$ anymore. If the first $\mathcal{F}_{\text{HG}}(\text{dropCoin})$ -step of $\mathcal{F}_{\text{HG}}(G)$ is simulated by the $\mathcal{F}_{\text{HG}}(\text{dropCoinC})$ -step of $\mathcal{F}_{\text{HG}}(G')$, then afterwards the $\mathcal{F}_{\text{HG}}(\text{orderTea})$ -step of $\mathcal{F}_{\text{HG}}(G)$ cannot be simulated by $\mathcal{F}_{\text{HG}}(G')$ anymore. Therefore, there is no $\mathcal{F}_{\text{HG}}(R)$ -bisimulation containing $(\mathcal{F}_{\text{HG}}(G), \mathcal{F}_{\text{HG}}(G'))$. Now using the contraposition of Theorem 5, we can deduce that hypergraphs G and G' are not R -bisimilar as well.

¹ In Figure 44, Figure 45, and Figure 46 we omit the edge typing and attribution for better readability.

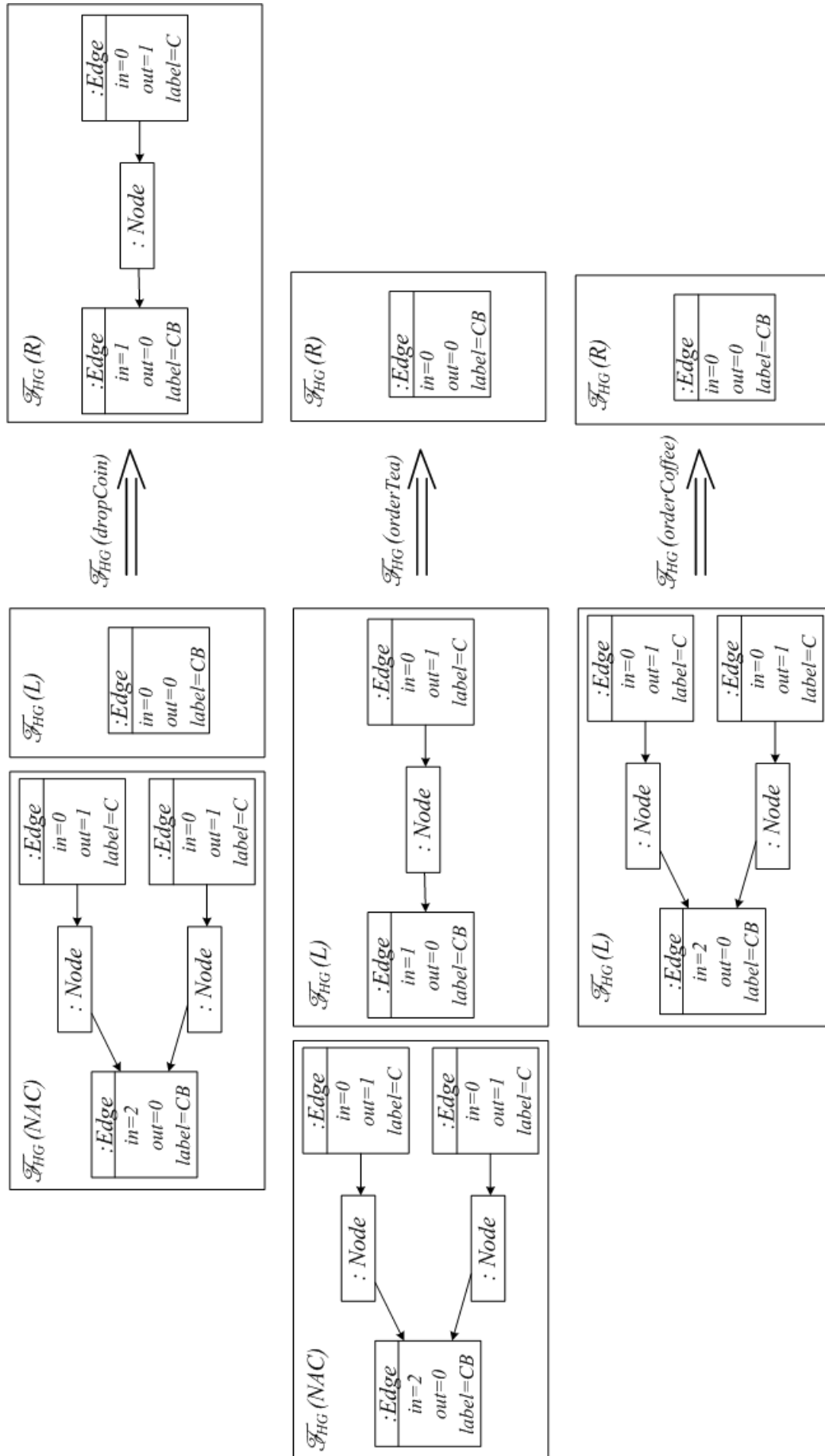


Figure 44: Typed attributed graph transformation rules modeling the behavior of the Coffee-Tea system CT_3

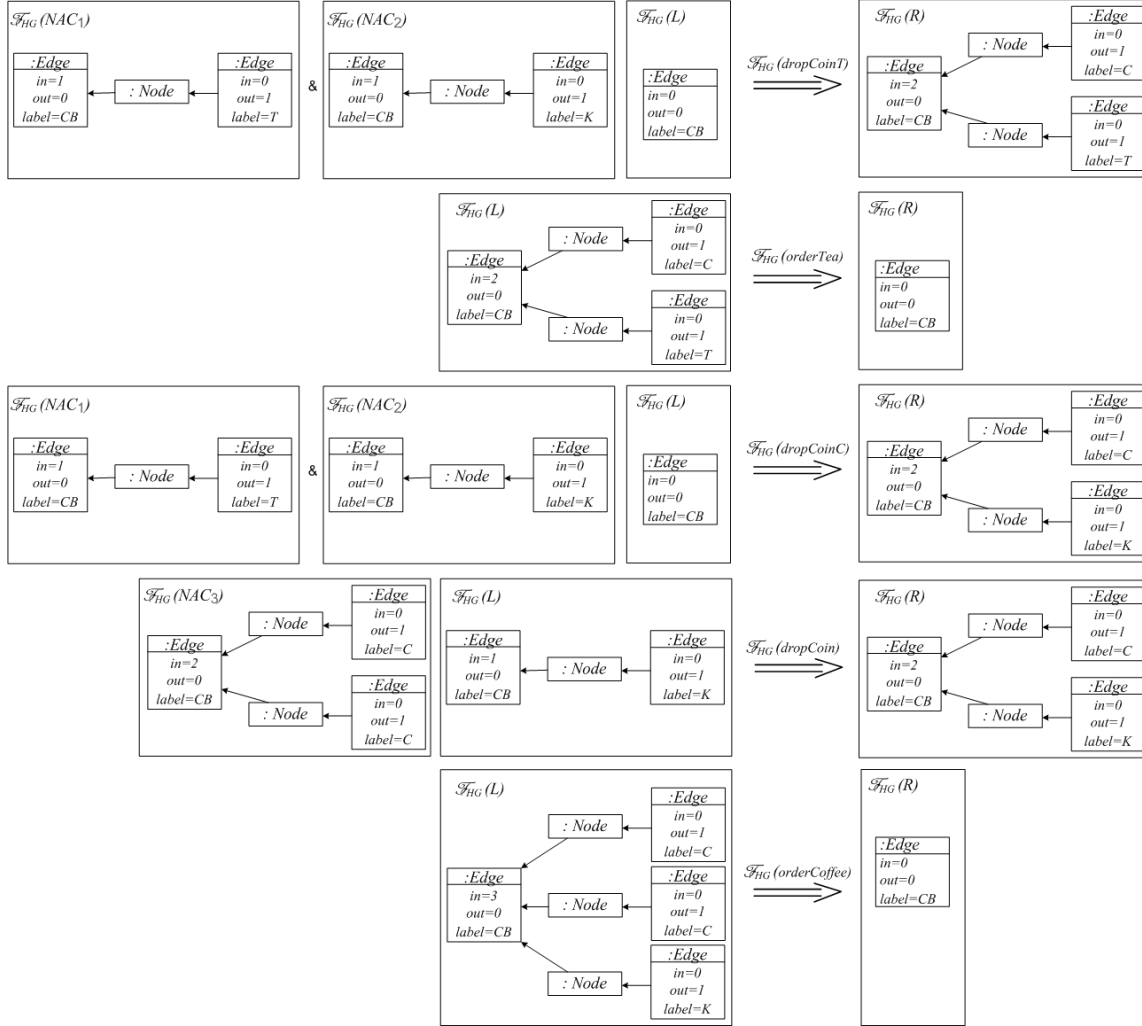
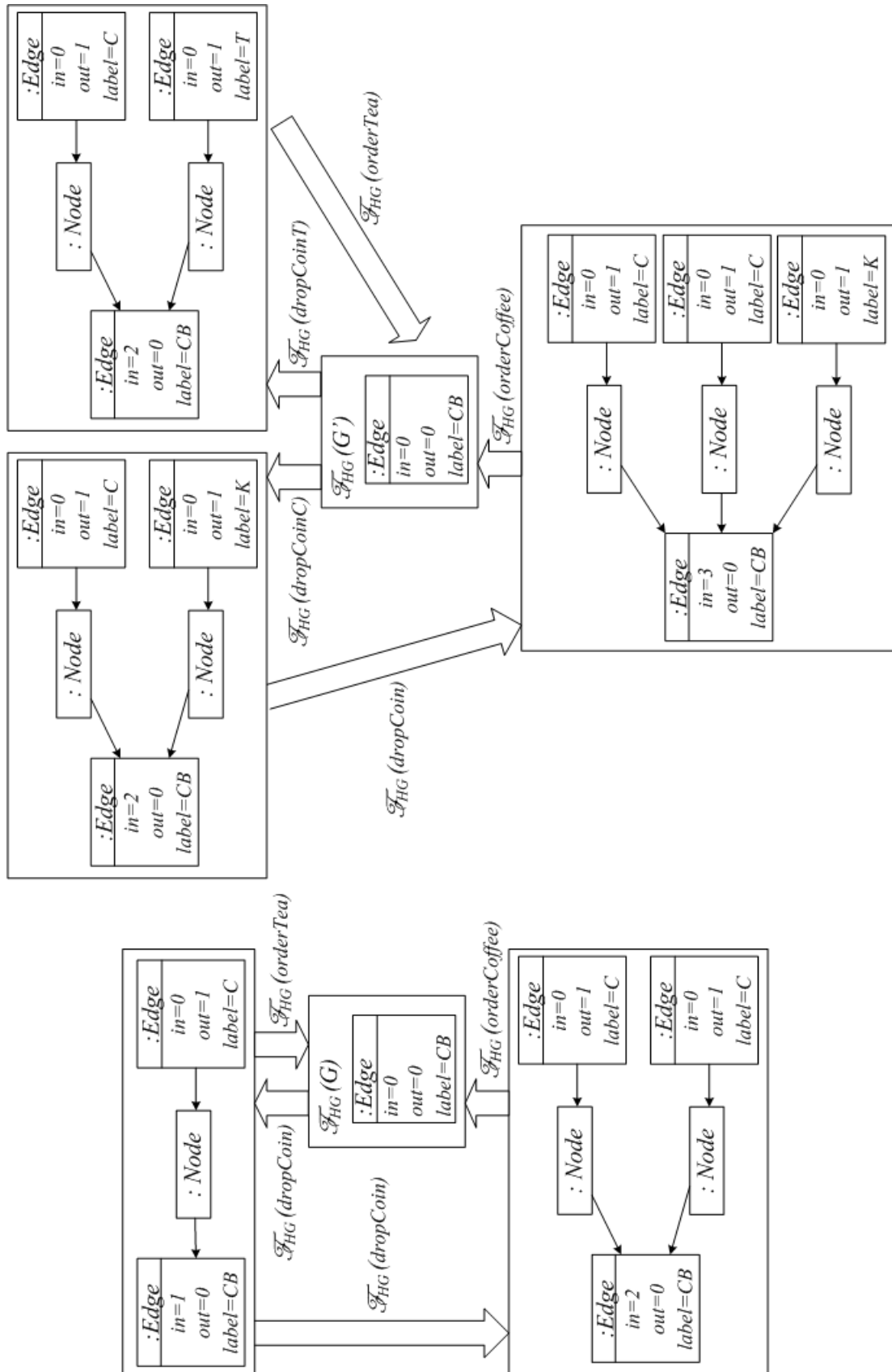


Figure 45: Typed attributed graph transformation rules modeling the behavior of the Coffee-Tea system CT_4

Figure 46: Reachability graphs of CT_3 (given to the left) and CT_4 (given to the right)

CONFLUENCE ANALYSIS OF HYPERGRAPH TRANSFORMATION SYSTEMS

In this chapter, we consider how we can apply theoretical results from Sections 4.1 and 4.2 to our concrete \mathcal{M} -functor \mathcal{F}_{HG} between categories of hypergraphs and typed attributed graphs to obtain local confluence analysis results for hypergraph transformation systems. In Section 7.1 we concern local confluence analysis of hypergraph transformation systems without nested application conditions, while in Section 7.2 we focus on local confluence analysis of hypergraph transformation systems containing rules with nested application conditions. Subsequently, in Section 7.3, we verify local confluence of a concrete small hypergraph transformation system introduced in Example 7 using the AGG-tool and our theoretical results from Sections 7.1 and 7.2. Finally, in Section 7.4 we consider termination, confluence, and functional behavior analysis for hypergraph transformation systems without or with nested application conditions.

7.1 LOCAL CONFLUENCE OF HYPERGRAPH TRANSFORMATION SYSTEMS WITHOUT NESTED APPLICATION CONDITIONS

In this section, we consider in detail how we can analyze local confluence of hypergraph transformation systems using our framework of \mathcal{M} -functors and the corresponding theoretical results concerning the functorial transfer of local confluence for transformations without nested application conditions, which we have already introduced in Section 4.1.

In the following we formulate and show several technical basics and results, which we need to be able to prove the main applicational result of this section concerning the local confluence analysis of hypergraph transformation systems. The most important technical requirement, which we need for this reason, is given in Lemma 38 where we show that the \mathcal{M} -functor \mathcal{F}_{HG} is compatible with pair factorization. According to Definition 51, this means that the source and the target categories of \mathcal{F}_{HG} have $\mathcal{E}' - \mathcal{M}$ pair factorizations and \mathcal{F}_{HG} translates an $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization in **HyperGraphs** into an $\mathcal{E}'_2 - \mathcal{M}_2$ pair factorization in **AGraphs_{HGTG}**.

Since for the source and the target categories of \mathcal{F}_{HG} holds that $\mathcal{M}_i = \mathcal{M}'_i$ for $i \in \{1, 2\}$, we trivially have that the $\mathcal{M} - \mathcal{M}'$ pushout-pullback decomposition property introduced in Definition 72 (see Appendix A) and needed for the completeness of critical pairs (see Fact 2) is fulfilled and we do not need to assume the satisfaction of this property in the subsequent results. Moreover, we use in the following the notion of $\mathcal{E}' - \mathcal{M}$ pair factorization instead of the common $\mathcal{E}' - \mathcal{M}'$ pair factorization as already mentioned in Remark 1.

As the first step we show that the source and the target categories of the \mathcal{M} -functor \mathcal{F}_{HG} have the corresponding $\mathcal{E}' - \mathcal{M}$ pair factorizations. According to Remark 5.26 from [88, p. 122], there is a general construction for an $\mathcal{E}' - \mathcal{M}'$ pair factorization based on coproducts and $\mathcal{E} - \mathcal{M}$ -factorization, which can be applied to hypergraphs and typed attributed graphs. In our concrete context this requires that the categories **HyperGraphs**

and $\mathbf{AGraphs}_{\text{HGTG}}$ have coproducts and $\mathcal{E} - \mathcal{M}$ -factorizations. Since the empty hypergraph \emptyset resp. almost empty typed attributed graph $\mathcal{F}_{\text{HG}}(\emptyset)$ (see more details in Section 5.1) are initial in the categories $\mathbf{HyperGraphs}$ resp. $\mathbf{AGraphs}_{\text{HGTG}}$ and we have pushouts in both categories, we also have coproducts in both categories constructed componentwise as disjoint union. Moreover, as we already know from [88], the category $\mathbf{AGraphs}_{\text{ATG}}$ and hence also $\mathbf{AGraphs}_{\text{HGTG}}$ has $\mathcal{E}_2 - \mathcal{M}_2$ -factorization where \mathcal{M}_2 is the class of all injective typed attributed graph morphisms and \mathcal{E}_2 is the class of all surjective typed attributed graph morphisms. Note that for typed attributed graphs all morphisms are identical in the V_D -component and the data type \mathbb{N} . The corresponding $\mathcal{E}_1 - \mathcal{M}_1$ -factorization for hypergraphs can be constructed as given in the following lemma.

Lemma 35 ($\mathcal{E} - \mathcal{M}$ -Factorization in HyperGraphs).

The \mathcal{M} -adhesive category $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ has an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization according to Definition 10 where \mathcal{M}_1 is the class of all injective hypergraph morphisms and \mathcal{E}_1 is the class of all surjective hypergraph morphisms.

Proof.

The detailed proof of this lemma is given in Appendix C on page 342. \square

To show the compatibility of \mathcal{F}_{HG} with pair factorization, we need to know additionally to the existence of coproducts and $\mathcal{E} - \mathcal{M}$ -factorizations in both categories that \mathcal{F}_{HG} also preserves coproducts and $\mathcal{E} - \mathcal{M}$ -factorizations. The first of these two properties is shown in the lemma below.

Lemma 36 (\mathcal{F}_{HG} Preserves Coproducts).

Consider a hypergraph A , a family of hypergraphs $(A_j)_{j \in I}$, a family of hypergraph morphisms $(i_j : A_j \rightarrow A)_{j \in I}$, a coproduct $(A, (i_j)_{j \in I})$ of $(A_j)_{j \in I}$ in $\mathbf{HyperGraphs}$, and the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Then $(\mathcal{F}_{\text{HG}}(A), (\mathcal{F}_{\text{HG}}(i_j))_{j \in I})$ is a coproduct of $(\mathcal{F}_{\text{HG}}(A_j))_{j \in I}$ in $\mathbf{AGraphs}_{\text{HGTG}}$.

$$\begin{array}{ccc}
 A_j & \xrightarrow{i_j} & A \\
 & \searrow f_j & \downarrow f \\
 & & B
 \end{array}
 \xrightarrow{\mathcal{F}_{\text{HG}}}
 \begin{array}{ccc}
 \mathcal{F}_{\text{HG}}(A_j) & \xrightarrow{\mathcal{F}_{\text{HG}}(i_j)} & \mathcal{F}_{\text{HG}}(A) \\
 & \searrow \mathcal{F}_{\text{HG}}(f_j) & \downarrow \mathcal{F}_{\text{HG}}(f) \\
 & & \mathcal{F}_{\text{HG}}(B)
 \end{array}$$

Proof.

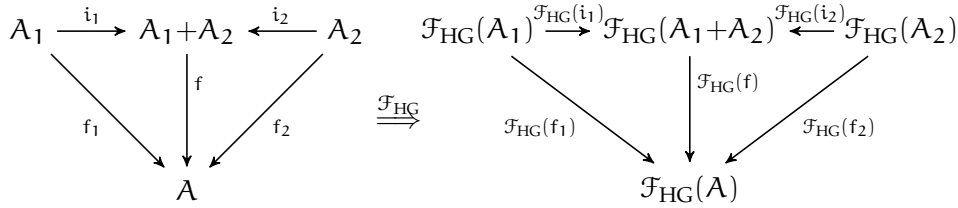
The detailed proof of this lemma is given in Appendix C on page 344. \square

Note that in our approach we use the preservation of coproducts for the binary case only as given in Figure 47 to construct the corresponding $\mathcal{E}' - \mathcal{M}$ pair factorizations for source and target categories of the considered \mathcal{M} -functor.

The second property, which remains to be shown, is that \mathcal{F}_{HG} preserves $\mathcal{E} - \mathcal{M}$ -factorizations. For this purpose, we have to verify that \mathcal{F}_{HG} preserves injective and surjective morphisms. Since \mathcal{F}_{HG} is an \mathcal{M} -functor, we already have that \mathcal{F}_{HG} preserves injective morphisms as shown in Lemma 27. The preservation of surjective morphisms is shown in the lemma below.

Lemma 37 (\mathcal{F}_{HG} Preserves Surjective Morphisms [211]).

Consider two hypergraphs $H_1 = (V_1, E_1, s_1, t_1)$, $H_2 = (V_2, E_2, s_2, t_2)$, a surjective hypergraph

Figure 47: Preservation of binary coproducts by the \mathcal{M} -functor \mathcal{F}_{HG}

morphism $f : H_1 \rightarrow H_2$ with $f = (f_V : V_1 \rightarrow V_2, f_E : E_1 \rightarrow E_2)$, and the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Then the corresponding typed attributed graph morphism $\mathcal{F}_{\text{HG}}(f) : \mathcal{F}_{\text{HG}}(H_1) \rightarrow \mathcal{F}_{\text{HG}}(H_2)$ with $\mathcal{F}_{\text{HG}}(f) = f' = (f'_{V_G} : V_G^{G_1} \rightarrow V_G^{G_2}, f'_{V_D} : \mathbb{N} \rightarrow \mathbb{N}, f'_{E_G} : E_G^{G_1} \rightarrow E_G^{G_2}, f'_{E_{NA}} : E_{NA}^{G_1} \rightarrow E_{NA}^{G_2}, f'_{E_{EA}} : E_{EA}^{G_1} \rightarrow E_{EA}^{G_2})$ is also surjective.

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 345](#). \square

In the following lemma, we summarize the results from the previous three lemmas and show that the \mathcal{M} -functor \mathcal{F}_{HG} is compatible with pair factorization. This technical property is required to obtain local confluence of hypergraph transformation systems based on \mathcal{F}_{HG} -reachable critical pairs.

Lemma 38 (\mathcal{F}_{HG} is Compatible with Pair Factorization [211]).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{HyperGraphs}, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P))$, and the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Then \mathcal{F}_{HG} is compatible with pair factorization.

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 346](#). \square

Now we apply the result concerning the creation of local confluence based on \mathcal{F} -reachable critical pairs from Theorem 8 to the concrete \mathcal{M} -functor \mathcal{F}_{HG} . This is one of our main applicational results allowing us to analyze local confluence of hypergraph transformation systems by analyzing their translated representations as typed attributed graph transformation systems using the AGG-tool.

Theorem 17 (Local Confluence of Hypergraph Transformation Systems without Nested Application Conditions [211]).

Consider a hypergraph transformation system $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P))$, and the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$ is locally confluent for all transformation spans $H_1 \xrightarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ if all \mathcal{F}_{HG} -reachable critical pairs of $\mathcal{F}_{\text{HG}}(\rho_1)$ and $\mathcal{F}_{\text{HG}}(\rho_2)$ are strictly confluent.

Proof.

In Lemma 30 resp. Theorem 15 we have already shown that \mathcal{F}_{HG} creates morphisms resp. (direct) transformations. Moreover, according to Lemma 38 \mathcal{F}_{HG} is compatible with pair factorization, which altogether allows us to apply Theorem 8 with $\mathcal{F} = \mathcal{F}_{\text{HG}}$. \square

7.2 LOCAL CONFLUENCE OF HYPERGRAPH TRANSFORMATION SYSTEMS WITH NESTED APPLICATION CONDITIONS

This section is build up similarly to the previous section concerning this time the verification of local confluence for hypergraph transformation systems with nested application conditions. We are going to verify sufficient properties like translation and creation of jointly surjective morphisms as well as preservation of pullbacks of injective morphisms, which allow us to apply Theorem 11 concerning the creation of local confluence based on \mathcal{F} -reachable critical pairs for rules with nested application conditions to the \mathcal{M} -functor \mathcal{F}_{HG} from Definition 63.

We begin this section with the following two lemmas, in which we show that the \mathcal{M} -functor \mathcal{F}_{HG} between the categories of hypergraphs and typed attributed graphs preserves and creates \mathcal{E}' -instances or, more precisely for the case of hypergraph and typed attributed graph transformations, jointly surjective morphisms. We need these properties to obtain the compatibility of \mathcal{F}_{HG} with **Shift**-transformation, which is one of the requirements of Theorem 11.

Lemma 39 (\mathcal{F}_{HG} Translates Jointly Surjective Morphisms).

Consider \mathcal{E}'_1 - \mathcal{M}_1 pair factorization in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ and \mathcal{E}'_2 - \mathcal{M}_2 pair factorization in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$. Then the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ from Definition 63 translates jointly surjective hypergraph morphisms (α', β') into the corresponding jointly surjective typed attributed graph morphisms (α'', β'') with $\alpha'' = \mathcal{F}_{\text{HG}}(\alpha')$ and $\beta'' = \mathcal{F}_{\text{HG}}(\beta')$.

$$\begin{array}{ccc}
 & P' & \\
 & \downarrow \alpha' & \\
 C & \xrightarrow{\beta'} & C'
 \end{array}
 \qquad
 \begin{array}{ccc}
 & \mathcal{F}_{\text{HG}}(P') & \\
 & \downarrow \alpha'' = \mathcal{F}_{\text{HG}}(\alpha') & \\
 \mathcal{F}_{\text{HG}}(C) & \xrightarrow{\beta'' = \mathcal{F}_{\text{HG}}(\beta')} & C'' = \mathcal{F}_{\text{HG}}(C')
 \end{array}$$

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 347](#). □

Lemma 40 (\mathcal{F}_{HG} Creates Jointly Surjective Morphisms).

Consider \mathcal{E}'_1 - \mathcal{M}_1 pair factorization in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, \mathcal{E}'_2 - \mathcal{M}_2 pair factorization in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, and hypergraph morphisms $\alpha : P \rightarrow C$, $\beta : P \rightarrow P'$. Then the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ from Definition 63 creates jointly surjective hypergraph morphisms (α', β') from the corresponding typed attributed graph morphisms (α'', β'') in $\bar{\mathcal{E}}'_2 = \mathcal{F}_{\text{HG}}(\mathcal{E}'_1)$ if for all spans $(C \xleftarrow{\alpha} P \xrightarrow{\beta} P')$ holds that the diagram (1) below commutes, $\mathcal{F}_{\text{HG}}(\alpha') = \alpha''$, $\mathcal{F}_{\text{HG}}(\beta') = \beta''$, and the injectivity of β'' implies the injectivity of β' .

$$\begin{array}{ccc}
 P & \xrightarrow{\beta} & P' \\
 \alpha \downarrow & (1) & \downarrow \alpha' \\
 C & \xrightarrow{\beta'} & C'
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathcal{F}_{\text{HG}}(P) & \xrightarrow{\mathcal{F}_{\text{HG}}(\beta)} & \mathcal{F}_{\text{HG}}(P') \\
 \mathcal{F}_{\text{HG}}(\alpha) \downarrow & (2) & \downarrow \alpha'' = \mathcal{F}_{\text{HG}}(\alpha') \\
 \mathcal{F}_{\text{HG}}(C) & \xrightarrow{\beta'' = \mathcal{F}_{\text{HG}}(\beta')} & C'' = \mathcal{F}_{\text{HG}}(C')
 \end{array}$$

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 348](#). \square

The next technical property, which must hold for the application of our theoretical approach to the \mathcal{M} -functor \mathcal{F}_{HG} , is the preservation of hypergraph pullbacks of injective morphisms. This property has to be satisfied to be able to deal with translated derived rules as introduced in Definition 55.

Lemma 41 (\mathcal{F}_{HG} Preserves Pullbacks of Injective Morphisms [213]).

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63, hypergraphs G_i for $i \in \{0, 1, 2, 3\}$ with hypergraph morphisms $b = (b_V, b_E)$, $c = (c_V, c_E)$, $g = (g_V, g_E)$, $h = (h_V, h_E)$, and typed attributed graphs $\mathcal{F}_{\text{HG}}(G_i)$ for $i \in \{0, 1, 2, 3\}$ with typed attributed graph morphisms $\mathcal{F}_{\text{HG}}(b) = b' = (b'_{V_G}, b'_{V_D}, b'_{E_G}, b'_{E_{NA}}, b'_{E_{EA}})$, $\mathcal{F}_{\text{HG}}(c) = c' = (c'_{V_G}, c'_{V_D}, c'_{E_G}, c'_{E_{NA}}, c'_{E_{EA}})$, $\mathcal{F}_{\text{HG}}(g) = g' = (g'_{V_G}, g'_{V_D}, g'_{E_G}, g'_{E_{NA}}, g'_{E_{EA}})$, $\mathcal{F}_{\text{HG}}(h) = h' = (h'_{V_G}, h'_{V_D}, h'_{E_G}, h'_{E_{NA}}, h'_{E_{EA}})$. If (1) is a pullback in $\mathbf{HyperGraphs}$ with $g, h \in \mathcal{M}_1$ then we have that (2) is a pullback in $\mathbf{AGraphs}_{\text{HGTG}}$ with $\mathcal{F}_{\text{HG}}(g), \mathcal{F}_{\text{HG}}(h) \in \mathcal{M}_2$.

$$\begin{array}{ccc}
 G_0 & \xrightarrow{b} & G_1 \\
 c \downarrow & (1) & \downarrow g \\
 G_2 & \xrightarrow{h} & G_3
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathcal{F}_{\text{HG}}(G_0) & \xrightarrow{\mathcal{F}_{\text{HG}}(b)=b'} & \mathcal{F}_{\text{HG}}(G_1) \\
 \mathcal{F}_{\text{HG}}(c)=c' \downarrow & (2) & \downarrow \mathcal{F}_{\text{HG}}(g)=g' \\
 \mathcal{F}_{\text{HG}}(G_2) & \xrightarrow{\mathcal{F}_{\text{HG}}(h)=h'} & \mathcal{F}_{\text{HG}}(G_3)
 \end{array}$$

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 348](#). \square

After all needed requirements are shown to be satisfied, we can now apply the result concerning the creation of local confluence based on \mathcal{F} -reachable critical pairs for the case of transformations with nested application conditions from Theorem 11 to the \mathcal{M} -functor \mathcal{F}_{HG} . This result extends the corresponding result from the previous section and allows us to analyze local confluence of hypergraph transformation systems containing rules with nested application conditions by analyzing their corresponding translated representations as typed attributed graph transformation systems using the AGG-tool.

Theorem 18 (Local Confluence of Hypergraph Transformation Systems with Nested Application Conditions [213]).

Consider a hypergraph transformation system $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P))$ where P are rules with nested application conditions, and the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Then $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$ is locally confluent for all transformation spans $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ with nested application conditions if all \mathcal{F}_{HG} -reachable critical pairs with nested application conditions of $\mathcal{F}_{\text{HG}}(\rho_1)$ and $\mathcal{F}_{\text{HG}}(\rho_2)$ are strictly $\text{AC}(\mathcal{F}_{\text{HG}})$ -confluent.

Proof.

In Lemma 30 resp. Theorem 15 we have shown that \mathcal{F}_{HG} creates morphisms resp. (direct) transformations and rule applicability. According to Lemma 31 resp. Lemma 38, we have that \mathcal{F}_{HG}

creates injective morphisms resp. \mathcal{F}_{HG} is compatible with pair factorization. Moreover, by Theorem 14 resp. Lemma 41 it holds that \mathcal{F}_{HG} translates rule applicability resp. preserves pullbacks of injective morphisms. Finally, in Lemma 39 resp. Lemma 40 we have shown that \mathcal{F}_{HG} translates resp. creates jointly surjective morphisms. This altogether allows us to apply Theorem 11 with $\mathcal{F} = \mathcal{F}_{\text{HG}}$. \square

The following example shows how we can construct a concrete \mathcal{F}_{HG} -reachable critical pair and how we can check whether this \mathcal{F}_{HG} -reachable critical pair is strictly $\text{AC}(\mathcal{F}_{\text{HG}})$ -confluent.

Example 16 (Strict $\text{AC}(\mathcal{F}_{\text{HG}})$ -Confluence of an \mathcal{F}_{HG} -Reachable Critical Pair in the Context of Mobile Processes Scenario [213]).

Consider an \mathcal{F} -reachable weak critical pair $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\text{runP}), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{enterServer}), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ for the weak critical pair $P_1 \xleftarrow{\text{runP}, o_1} K \xrightarrow{\text{enterServer}, o_2} P_2$ from Example 8¹ with $\text{ac}_{\mathcal{F}(K)} = \mathcal{F}(\text{ac}_K)$, $\text{ac}_{\mathcal{F}(K)}^* = \mathcal{F}(\text{ac}_K^*)$, and $\mathcal{F} = \mathcal{F}_{\text{HG}}$. Then we have the following translated extension and conflict-inducing application conditions:

- $\text{ac}_{\mathcal{F}(K)} = \mathcal{F}(\text{ac}_K) = \mathcal{F}(\text{ac}_{K_1} \wedge \text{ac}_{K_2}) \stackrel{\text{Def. 47}}{=} \mathcal{F}(\text{ac}_{K_1}) \wedge \mathcal{F}(\text{ac}_{K_2})$
- $\text{ac}_{\mathcal{F}(K)}^* = \mathcal{F}(\text{ac}_K^*) = \mathcal{F}(\text{true}) \stackrel{\text{Def. 47}}{=} \text{true}$

Similar to Example 8, $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\text{runP}), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{enterServer}), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ can be extended by the embedding morphism $\mathcal{F}(m) : \mathcal{F}(K) \rightarrow \mathcal{F}(G)$ with $\mathcal{F}(m) \models \text{ac}_{\mathcal{F}(K)} \wedge \text{ac}_{\mathcal{F}(K)}^*$ and a translated pair of AC-regarding transformations $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\text{runP}), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\text{enterServer}), \mathcal{F}(m_2)} \mathcal{F}(H_2)$. Thus, we have that $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\text{runP}), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{enterServer}), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ is an \mathcal{F} -reachable critical pair.

$$\begin{array}{ccccc} \mathcal{F}(P_1) & \xleftarrow{\mathcal{F}(\text{runP}), \mathcal{F}(o_1)} & \mathcal{F}(K) & \xrightarrow{\mathcal{F}(\text{enterServer}), \mathcal{F}(o_2)} & \mathcal{F}(P_2) \\ \downarrow & & \downarrow \mathcal{F}(m) & & \downarrow \\ \mathcal{F}(H_1) & \xleftarrow{\mathcal{F}(\text{runP}), \mathcal{F}(m_1)} & \mathcal{F}(G) & \xrightarrow{\mathcal{F}(\text{enterServer}), \mathcal{F}(m_2)} & \mathcal{F}(H_2) \end{array}$$

Now we want to show that $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\text{runP}), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{enterServer}), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ is strictly $\text{AC}(\mathcal{F})$ -confluent according to Definition 60. We have the plain strict confluence similarly to Example 8 as given in the diagram below

$$\begin{array}{ccccc} & & \mathcal{F}(K) & & \\ & \swarrow \mathcal{F}(\text{runP}), \mathcal{F}(o_1) & & \searrow \mathcal{F}(\text{enterServer}), \mathcal{F}(o_2) & \\ \mathcal{F}(P_1) & & & & \mathcal{F}(P_2) \\ & \nwarrow \mathcal{F}(\text{runP}), \mathcal{F}(o_1) & & \nearrow \mathcal{F}(\text{leaveServer}), \mathcal{F}(o_3) & \\ & & \mathcal{F}(K) & & \end{array}$$

¹ Note that in Section 7.3 where we describe the AGG-based local confluence analysis of our concrete hypergraph transformation system with NACs and PACs, we can exclude all \mathcal{F}_{HG} -reachable critical pairs of the rules $\mathcal{F}_{\text{HG}}(\text{runP})$ and $\mathcal{F}_{\text{HG}}(\text{enterServer})$ from the detailed strict $\text{AC}(\mathcal{F}_{\text{HG}})$ -confluence analysis because the corresponding overlapping graphs cannot be reached from the translated start hypergraph, which means that these \mathcal{F}_{HG} -reachable critical pairs are incompatible with some invariant of the original hypergraph transformation system. In this example, we nevertheless consider the strict $\text{AC}(\mathcal{F}_{\text{HG}})$ -confluence of the \mathcal{F}_{HG} -reachable critical pair $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\text{runP}), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{enterServer}), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ as an analytic continuation of Example 8.

and the $\text{AC}(\mathcal{F})$ -compatibility, i.e., $(\text{ac}_{\mathcal{F}(\mathcal{K})} \wedge \text{ac}_{\mathcal{F}(\mathcal{K})}^*) \Rightarrow (\text{ac}(\mathcal{F}(t_1)) \wedge \text{ac}(\mathcal{F}(t_2)))$ since $\text{ac}_{\mathcal{F}(\mathcal{K})} \wedge \text{ac}_{\mathcal{F}(\mathcal{K})}^* = \mathcal{F}(\text{ac}_{\mathcal{K}_1}) \wedge \mathcal{F}(\text{ac}_{\mathcal{K}_2})$ (as shown above) implies $\text{ac}(\mathcal{F}(t_1)) \wedge \text{ac}(\mathcal{F}(t_2)) = \mathcal{F}(\text{ac}_{\mathcal{K}_1}) \wedge \mathcal{F}(\text{ac}_{\mathcal{K}_2})$ as shown below.

$$\begin{aligned} \text{ac}(\mathcal{F}(t_1)) &= \text{ac}(\mathcal{F}(\mathcal{K})) \xrightarrow{\mathcal{F}(\text{runP}), \mathcal{F}(\text{o}_1)} \mathcal{F}(\mathcal{P}_1) = \mathcal{F}(\text{ac}_{\mathcal{K}_1}), \\ \text{ac}(\mathcal{F}(t_2)) &= \text{ac}(\mathcal{F}(\mathcal{K})) \xrightarrow{\mathcal{F}(\text{enterServer}), \mathcal{F}(\text{o}_2)} \mathcal{F}(\mathcal{P}_2) \xrightarrow{\mathcal{F}(\text{leaveServer}), \mathcal{F}(\text{o}_3)} \mathcal{F}(\mathcal{K}) \\ &\xrightarrow{\mathcal{F}(\text{runP}), \mathcal{F}(\text{o}_1)} \mathcal{F}(\mathcal{P}_1) = \mathcal{F}(\text{ac}_{\mathcal{K}_2}) \wedge \mathcal{F}(\text{true}) \wedge \mathcal{F}(\text{ac}_{\mathcal{K}_1}), \text{ and} \\ \text{ac}(\mathcal{F}(t_1)) \wedge \text{ac}(\mathcal{F}(t_2)) &= \mathcal{F}(\text{ac}_{\mathcal{K}_1}) \wedge \mathcal{F}(\text{ac}_{\mathcal{K}_2}) \end{aligned}$$

7.3 AGG-BASED LOCAL CONFLUENCE ANALYSIS OF HYPERGRAPH TRANSFORMATION SYSTEMS

In this section, we introduce our workflow for the analysis of local confluence and apply it to our concrete hypergraph transformation systems without and with nested application conditions from Example 7. This workflow, which integrates the application of our theoretical results from Sections 4.1 and 7.1 for the case of rules *without* nested application conditions and the theoretical results from Sections 4.2 and 7.2 for the case of rules *with* nested application conditions, is partially supported by the AGG-tool, which is used for critical pair computation. Note that our approach is tool independent, which means that other tools that are capable to compute and possibly analyze critical pairs (for rules without or with nested application conditions) can be used equivalently. Subsequently, we are only using the AGG-tool for critical pair computation since other existing tools that are capable to compute critical pairs like *Henshin* [69, 6], *VERIGRAPH* [305] or *SyGrAV* [53, 290] do not allow the involved rules to make use of NACs and PACs so far.

In the following we consider the two hypergraph transformation systems without and with nested application conditions that were already introduced in Example 7. This example, inspired by [12], deals with simple distributed systems with mobility containing servers connected by channels as well as processes moving through the network and running on the servers before being removed after their termination. The rules of the hypergraph transformation system *without* nested application conditions are given in Figure 12 and the rules of the hypergraph transformation system *with* nested application conditions are given in Figure 13.

To be able to use our theoretical results for the local confluence analysis of these hypergraph transformation systems, we have to extend the \mathcal{M} -functor \mathcal{F}_{HG} from Definition 63 to the translation of the hyperedge labels into String attributes of the corresponding hyperedge node representation as already done in Example 14. However, all technical properties shown in Subsection 2.4.2 and Sections 6.1, 6.2, 7.1, 7.2 also hold for the \mathcal{M} -adhesive category of labeled hypergraphs and the extended \mathcal{M} -functor $\mathcal{F}_{\text{HG}}^2$.

In fact, we consider in the following four different scenarios for which the corresponding workflows share common steps (see Figure 48). Firstly, we distinguish between hypergraph transformation systems without and with nested application conditions. We

² Note that this holds also for the variants of hypergraphs with labeled nodes and/or labeled hyperedges.

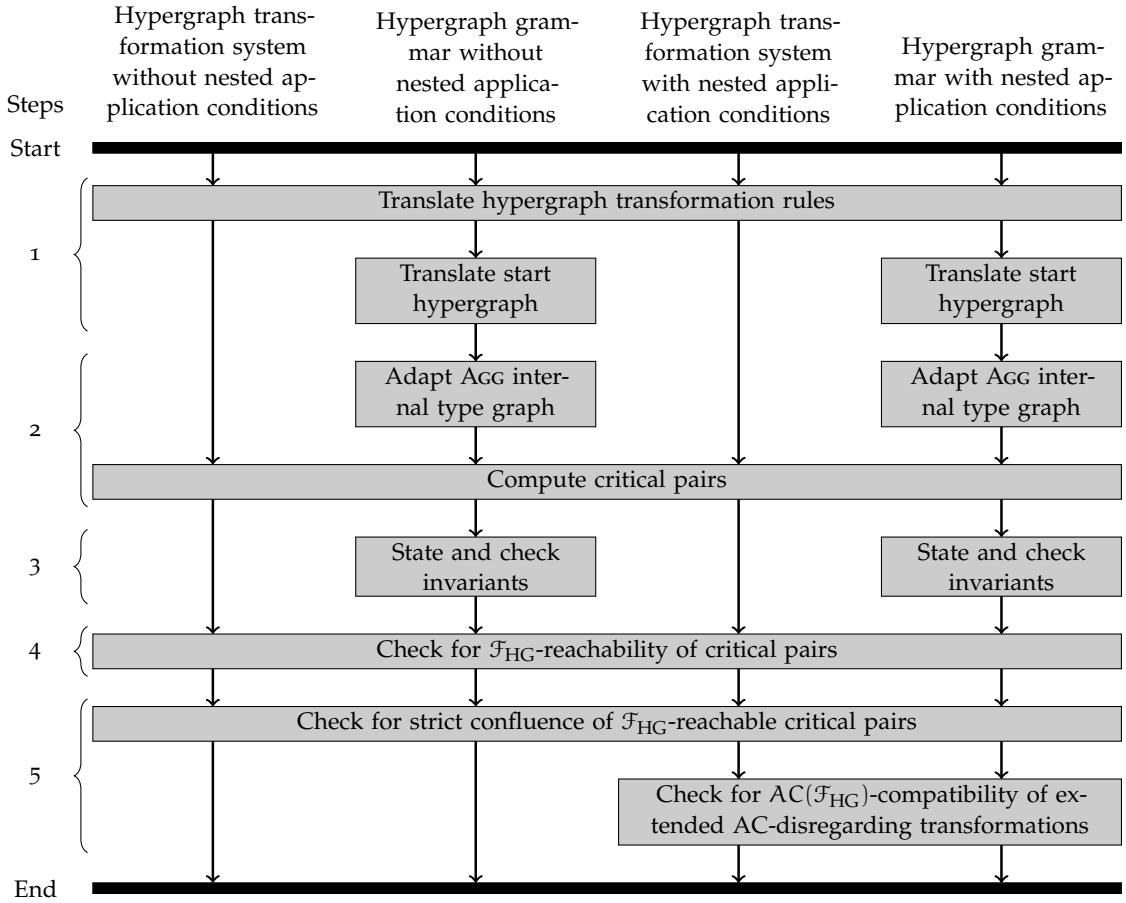


Figure 48: Workflow for local confluence analysis of hypergraph transformation systems and hypergraph grammars without or with nested application conditions

start with the workflow for hypergraph transformation systems *without* nested application conditions and then address the differences for the case in which nested application conditions are used. Secondly, while the theory introduced in this thesis is tailored to transformation systems, it is beneficial to also consider the case where the transformation system is given with a start structure at hand. In the hypergraph context, such combinations of a hypergraph transformation system and a start hypergraph are called hypergraph grammars [88]. Using straightforward adaptations of our theory we can analyze semantical properties also for hypergraph grammars. The core idea is then to analyze these properties not for all hypergraphs that are derivable within the hypergraph transformation system but only for those that are reachable from the given start hypergraph by rule application. The discussion of how the possible usage of a start hypergraph affects our workflow is included along the way.

Our proposed workflow for the local confluence analysis of a hypergraph transformation system *without* nested application conditions consists of the following four steps while the workflow for the analysis of a hypergraph grammar contains one additional intermediate step (see the third step below). Firstly, we apply the \mathcal{M} -functor \mathcal{F}_{HG} from Definition 63 to the considered hypergraph transformation system (or to the considered hypergraph grammar) obtaining the corresponding typed attributed graph transformation system (or the corresponding typed attributed graph grammar). Secondly, we com-

pute all critical pairs of the translated transformation rules using the AGG-tool. In the case of a translated hypergraph grammar we apply invariants during the critical pair computation that are given by the restriction of the AGG internal type graph using multiplicities to specify translated hypergraphs that are provably not reachable from the translated start hypergraph by rule application and that are therefore irrelevant in the context of the analysis of the considered hypergraph grammar. Note that the invariants defined by the restriction of the AGG internal type graph are in general not sufficient to ensure that all computed overlapping graphs are relevant for the further analysis. Hence, for translated hypergraph grammars the satisfaction of additional invariants is checked in the next step to improve the over-approximation of relevant critical pairs. Thirdly, only in the case of a translated hypergraph grammar, we again apply invariants (given e.g. by nested graph conditions [135]) that can be established by invariant verification techniques as e.g. proposed in [66, 67] to exclude further computed critical pairs whose overlapping graphs are provably not reachable from the translated start hypergraph by rule application. Fourthly, we select those computed critical pairs that are \mathcal{F}_{HG} -reachable. This elimination of non- \mathcal{F}_{HG} -reachable critical pairs is required by our theory (see Theorem 17) where critical pairs must not be considered in the further analysis when their overlapping graphs do not correspond to some translated valid hypergraph. Finally, we determine whether all these \mathcal{F}_{HG} -reachable critical pairs are strictly confluent, which is also partially supported by AGG³. If all computed \mathcal{F}_{HG} -reachable critical pairs (that are additionally compatible with invariants for the case of a translated hypergraph grammar) are strictly confluent, we obtain local confluence of the considered hypergraph transformation system (or of the corresponding hypergraph grammar) by application of Theorem 17⁴.

However, the computation of critical pairs using tools, such as AGG in our case, may take too much time, take too much memory, result in too many critical pairs or even not terminate at all. The restriction then to the corresponding hypergraph grammar and the usage of invariants (as discussed in detail above in the second and in the third step of the workflow) can typically result in a decrease of resource requirements for critical pair computation and fewer computed critical pairs.

We now apply our workflow to the hypergraph transformation system *without* nested application conditions from Example 7 and, by additionally using the start hypergraph from Figure 11, to the corresponding hypergraph grammar.

Firstly, we translate the hypergraph transformation rules given in Figure 12 according to Example 14 (see Figure 38 there) together with the start hypergraph from Figure 11 obtaining the corresponding start typed attributed graph.

Secondly, we apply AGG on the obtained set of translated transformation rules to compute all critical pairs of the translated transformation system. The table in Figure 49 shows the number of critical pairs that were computed by AGG for each possible pair of translated transformation rules.

³ Let K be a minimal overlapping of the left-hand sides of two conflicting rules computed by AGG. Let $P_1 \leftarrow K \Rightarrow P_2$ be the corresponding critical pair. Then using AGG we can apply the rules of the considered transformation system to P_1 and P_2 trying to merge them to a unique (up to isomorphism) common result graph. The strictness condition is then to be checked manually.

⁴ Note that when considering a hypergraph grammar we expect that a result similar to Theorem 17 is also valid.

Minimal Conflicts										
first \ second	1	2	3	4	5	6	7	8	9	10
1 F_HG(enterServer)	8	4	2	2	3	3	3	3	4	0
2 F_HG(leaveServer)	4	8	2	2	3	3	3	3	8	0
3 F_HG(crossC)	2	2	8	4	3	3	3	3	2	0
4 F_HG(backC)	2	2	4	8	3	3	3	3	2	0
5 F_HG(crossC_1)	3	3	3	3	31	16	31	16	3	0
6 F_HG(backC_1)	3	3	3	3	16	31	16	16	3	0
7 F_HG(crossC_2)	3	3	3	3	31	16	31	16	3	0
8 F_HG(backC_2)	3	3	3	3	16	16	16	31	3	0
9 F_HG(runP)	4	8	2	2	3	3	3	3	8	0
10 F_HG(removeR)	0	0	0	0	0	0	0	0	0	8

Figure 49: Computed critical pairs for the Mobile Processes system without considering any additional invariants

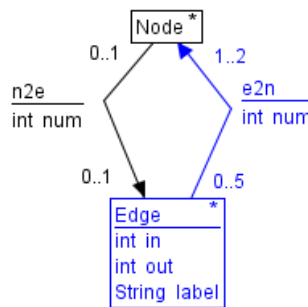
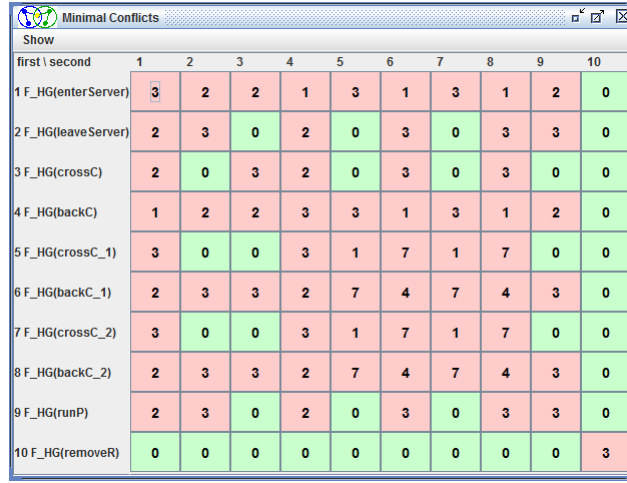


Figure 50: AGG type graph extended by multiplicities for the Mobile Processes system

Trying to reduce the number of computed critical pairs to a minimum for our example, we focused our considerations on the corresponding translated hypergraph grammar and stated, based on the translated transformation rules and the translated start hypergraph, some useful invariants. In particular, we determined invariants in the form of multiplicities for edges⁵ that we integrated into the AGG internal type graph (see Figure 50). Using these invariants during the critical pairs computation by AGG, we considerably reduced the number of computed critical pairs as given in Figure 51. The validity of the stated invariants can be easily verified by inspection of the translated rules from Figure 38 and the start hypergraph from Figure 11 translated by \mathcal{F}_{HG} . However, such suitable type graph restrictions do not exist for every hypergraph grammar.

Thirdly, to exclude further computed critical pairs from the subsequent analysis, we apply additional invariants to our translated hypergraph grammar. These invariants can be stated e.g. using nested graph conditions [135] and invariant verification techniques

⁵ For the AGG type graph given in Figure 50, the multiplicity 0..1 near to the graph node of the type *Node* expresses that in the context of our example the graph nodes of the type *Edge* are allowed to have 0 or 1 incoming edges while the multiplicity 1..2 means that the graph nodes of the type *Edge* can have 1 or 2 outgoing edges. Similarly, the multiplicity 0..1 near to the graph node of the type *Edge* expresses that the graph nodes of the type *Node* are allowed to have 0 or 1 outgoing edges while the multiplicity 0..5 means that the graph nodes of the type *Node* can have 0 to 5 incoming edges.



first \ second	1	2	3	4	5	6	7	8	9	10
1 $\mathcal{F}_{HG}(\text{enterServer})$	3	2	2	1	3	1	3	1	2	0
2 $\mathcal{F}_{HG}(\text{leaveServer})$	2	3	0	2	0	3	0	3	3	0
3 $\mathcal{F}_{HG}(\text{crossC})$	2	0	3	2	0	3	0	3	0	0
4 $\mathcal{F}_{HG}(\text{backC})$	1	2	2	3	3	1	3	1	2	0
5 $\mathcal{F}_{HG}(\text{crossC}_1)$	3	0	0	3	1	7	1	7	0	0
6 $\mathcal{F}_{HG}(\text{backC}_1)$	2	3	3	2	7	4	7	4	3	0
7 $\mathcal{F}_{HG}(\text{crossC}_2)$	3	0	0	3	1	7	1	7	0	0
8 $\mathcal{F}_{HG}(\text{backC}_2)$	2	3	3	2	7	4	7	4	3	0
9 $\mathcal{F}_{HG}(\text{runP})$	2	3	0	2	0	3	0	3	3	0
10 $\mathcal{F}_{HG}(\text{removeR})$	0	0	0	0	0	0	0	0	0	3

Figure 51: Computed critical pairs for the Mobile Processes system considering the AGG type graph with multiplicities

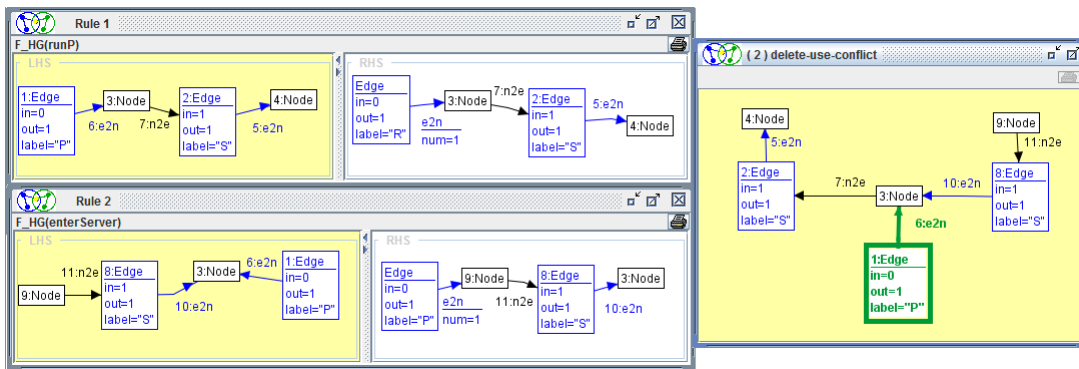


Figure 52: A critical pair computed by AGG for the rule pair $(\mathcal{F}_{HG}(\text{runP}), \mathcal{F}_{HG}(\text{enterServer}))$ non-reachable from the translated start hypergraph by rule application

such as [66, 67] may be used to check whether the translated transformation system together with the translated start hypergraph satisfy these invariants. Then, based on the stated invariants, we can detect (for example using tools such as AutoGraph [278]) critical pairs whose overlapping graphs cannot be contained in a graph that is reachable from the start graph of the translated transformation system by rule application. Moreover, for full automation of the invariant checking procedure we can simply attempt to verify that a given overlapping graph of a critical pair is a pattern that is invariantly not contained in any graph that is reachable from the translated start hypergraph and exclude this critical pair from further analysis if the invariant check confirms this conjecture. However, we believe that using small subgraphs of overlapping graphs for the invariants may be more appropriate to reduce computation costs for the invariant check and may also allow to exclude multiple critical pairs at once. An example for such an excluded critical pair is given in Figure 52. Therein the rules $\mathcal{F}_{HG}(\text{runP})$ and $\mathcal{F}_{HG}(\text{enterServer})$ are shown on the left and the corresponding overlapping graph is shown on the right. The overlapping graph depicts the situation where a process is located between two different servers, which is not possible according to the network given in Figure 11 and also cannot be reached by rule application.

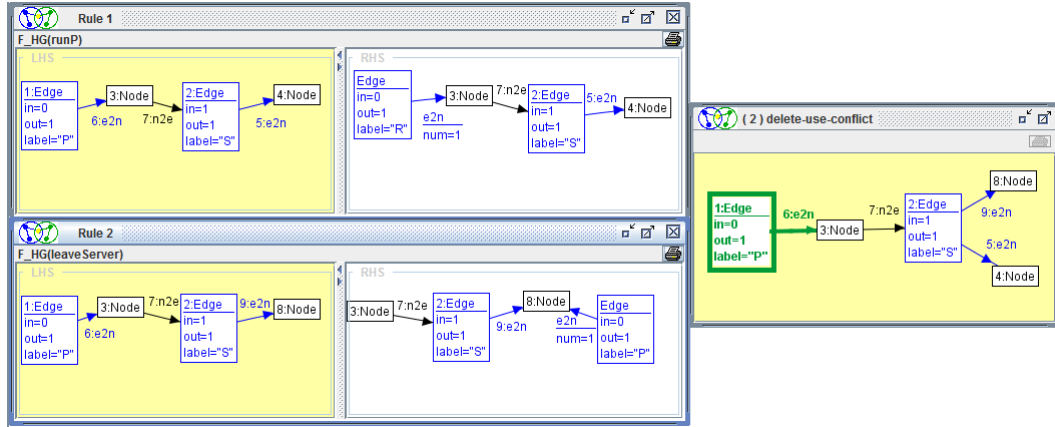


Figure 53: A non- \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{runP}), \mathcal{F}_{\text{HG}}(\text{leaveServer}))$

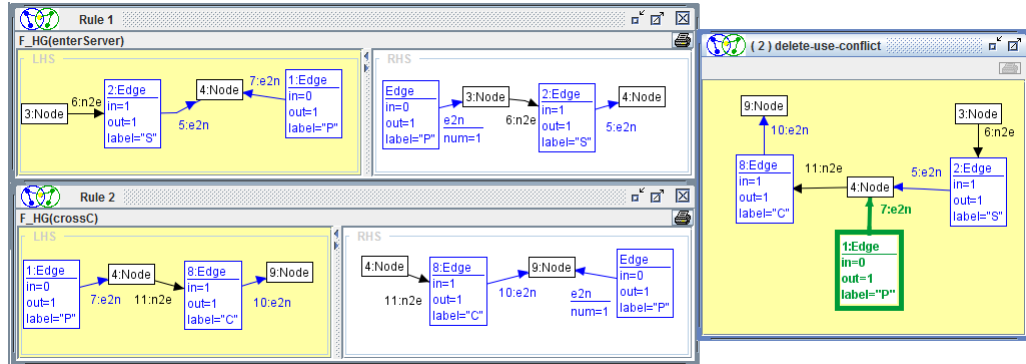


Figure 54: An \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{enterServer}), \mathcal{F}_{\text{HG}}(\text{crossC}))$

Fourthly, we exclude from the further analysis those computed critical pairs that have overlapping graphs that are not \mathcal{F}_{HG} -reachable. This step is of great importance because critical pairs that are not \mathcal{F}_{HG} -reachable do not need to be strictly confluent according to Theorem 17. An example for a non- \mathcal{F}_{HG} -reachable critical pair that is constructed for the rule pair $(\mathcal{F}_{\text{HG}}(\text{runP}), \mathcal{F}_{\text{HG}}(\text{leaveServer}))$ is shown in Figure 53. The depicted delete-use-conflict⁶ shows a non- \mathcal{F}_{HG} -reachable overlapping graph where a Server hyperedge 2 has two outgoing edges 5 and 9 connecting it to two different nodes. For this situation, an original hypergraph does not exist since the attributes $\text{in}=1$ and $\text{out}=1$ mean that the considered Server hyperedge of an original hypergraph must have exactly one incoming and one outgoing edge.

However, the critical pairs for the rules $\mathcal{F}_{\text{HG}}(\text{enterServer})$ and $\mathcal{F}_{\text{HG}}(\text{crossC})$ as well as $\mathcal{F}_{\text{HG}}(\text{runP})$ and $\mathcal{F}_{\text{HG}}(\text{leaveServer})$ given in AGG-notation in Figure 54 and Figure 55, respectively, are examples for computed critical pairs that are \mathcal{F}_{HG} -reachable (and, in the case of the corresponding translated hypergraph grammar, also compatible with the invariants). The conflicts are delete-use-conflicts, which are obviously caused when a process can “choose” between two possibilities to proceed.

⁶ We have a delete-use-conflict if one rule application deletes graph elements that are included in the match morphism of another rule application.

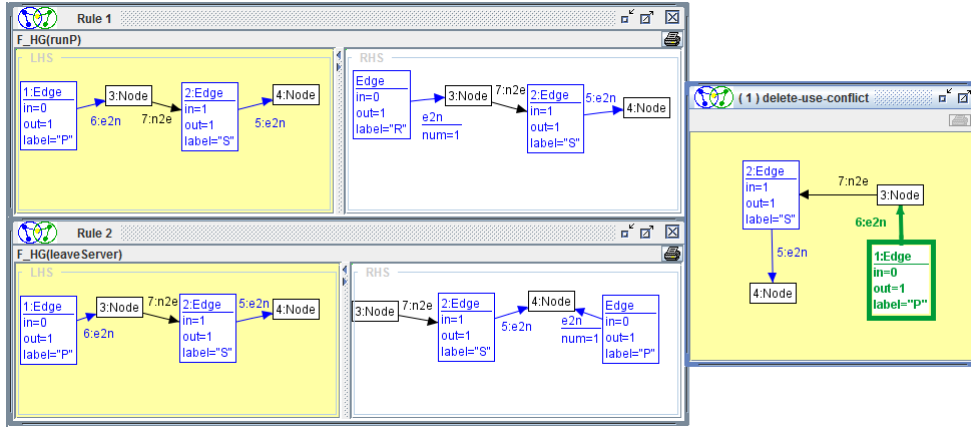


Figure 55: An \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{runP}), \mathcal{F}_{\text{HG}}(\text{leaveServer}))$

Finally, following our workflow, we verify the strict confluence of the remaining critical pairs to obtain that the considered hypergraph transformation system (or the corresponding hypergraph grammar) is locally confluent by applying Theorem 17. Obviously, rules for process traveling via connections and rules for passing a server can be executed in any order. Furthermore, a step modeling the leaving of a server and a step modeling the execution of a process can be joined again by performing entering a server and then executing the process. The corresponding strictness conditions for all \mathcal{F}_{HG} -reachable critical pairs (additionally compatible with invariants for the case of the hypergraph grammar) can then be shown by constructing the corresponding strictness diagrams according to Definition 14. Thus, our considered hypergraph transformation system (as well as the corresponding hypergraph grammar) *without* nested application conditions is locally confluent according to Theorem 17.

To give an example for the verification of strict confluence for a concrete \mathcal{F}_{HG} -reachable critical pair, let us consider the \mathcal{F}_{HG} -reachable critical pair of rules $\mathcal{F}_{\text{HG}}(\text{enterServer})$ and $\mathcal{F}_{\text{HG}}(\text{crossC})$ (see Figure 56 depicting the corresponding \mathcal{F}_{HG} -reachable critical pair diagram according to Definition 52). This \mathcal{F}_{HG} -reachable critical pair is strictly confluent because after applying either rule we can reverse the effect by applying the corresponding inverse rule and hence have at least one graph that can be derived to join the different results. The application of the rules in the mentioned order is depicted in Figure 57, which additionally shows that the corresponding strictness condition is satisfied. The spans $\mathcal{F}_{\text{HG}}(P_1) \xleftarrow{\mathcal{F}_{\text{HG}}(w_1)} \mathcal{F}_{\text{HG}}(N_1) \xrightarrow{\mathcal{F}_{\text{HG}}(v_1)} \mathcal{F}_{\text{HG}}(K)$ and $\mathcal{F}_{\text{HG}}(K) \xleftarrow{\mathcal{F}_{\text{HG}}(v_2)} \mathcal{F}_{\text{HG}}(N_2) \xrightarrow{\mathcal{F}_{\text{HG}}(w_2)} \mathcal{F}_{\text{HG}}(P_2)$ in the upper half of the diagram represent the two conflicting rule applications of rules $\mathcal{F}_{\text{HG}}(\text{enterServer})$ and $\mathcal{F}_{\text{HG}}(\text{crossC})$. The graph $\mathcal{F}_{\text{HG}}(N)$ in the center of the diagram is the largest subgraph of the computed overlapping graph $\mathcal{F}_{\text{HG}}(K)$ that is preserved by the critical pair. In the lower half of the diagram, we can see how the two transformation steps $\mathcal{F}_{\text{HG}}(K) \xrightarrow{\mathcal{F}_{\text{HG}}(\text{enterServer}), \mathcal{F}_{\text{HG}}(o_1)} \mathcal{F}_{\text{HG}}(P_1)$ and $\mathcal{F}_{\text{HG}}(K) \xrightarrow{\mathcal{F}_{\text{HG}}(\text{crossC}), \mathcal{F}_{\text{HG}}(o_2)} \mathcal{F}_{\text{HG}}(P_2)$ can be merged by applying the rule $\mathcal{F}_{\text{HG}}(\text{leaveServer})$ at an adequate match to the graph $\mathcal{F}_{\text{HG}}(P_1)$ on the left and the rule $\mathcal{F}_{\text{HG}}(\text{backC})$ at an adequate match to the graph $\mathcal{F}_{\text{HG}}(P_2)$ on the right. The strictness condition is satisfied because $\mathcal{F}_{\text{HG}}(N)$ is preserved by the merging transformation steps $\mathcal{F}_{\text{HG}}(P_1) \Rightarrow \mathcal{F}_{\text{HG}}(K')$ and $\mathcal{F}_{\text{HG}}(P_2) \Rightarrow \mathcal{F}_{\text{HG}}(K')$.

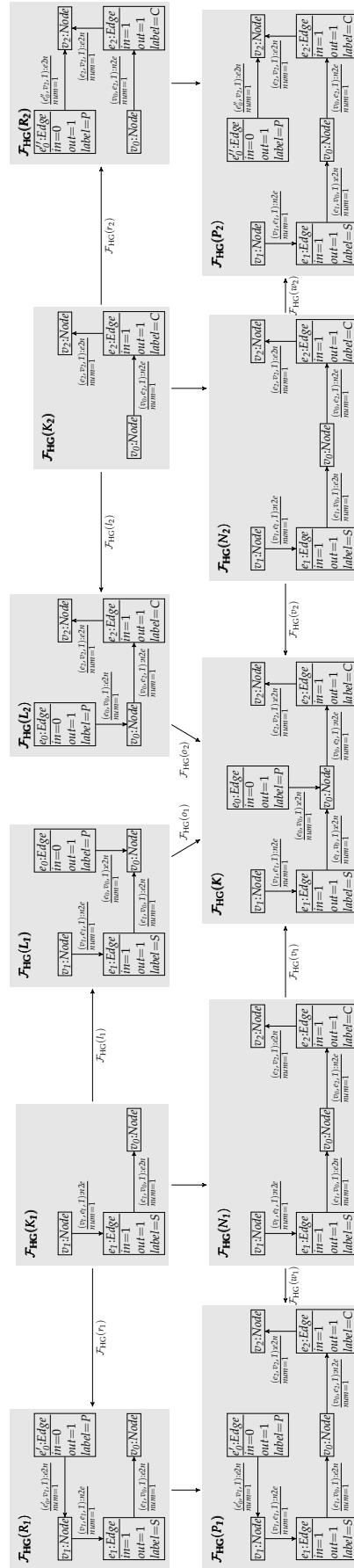
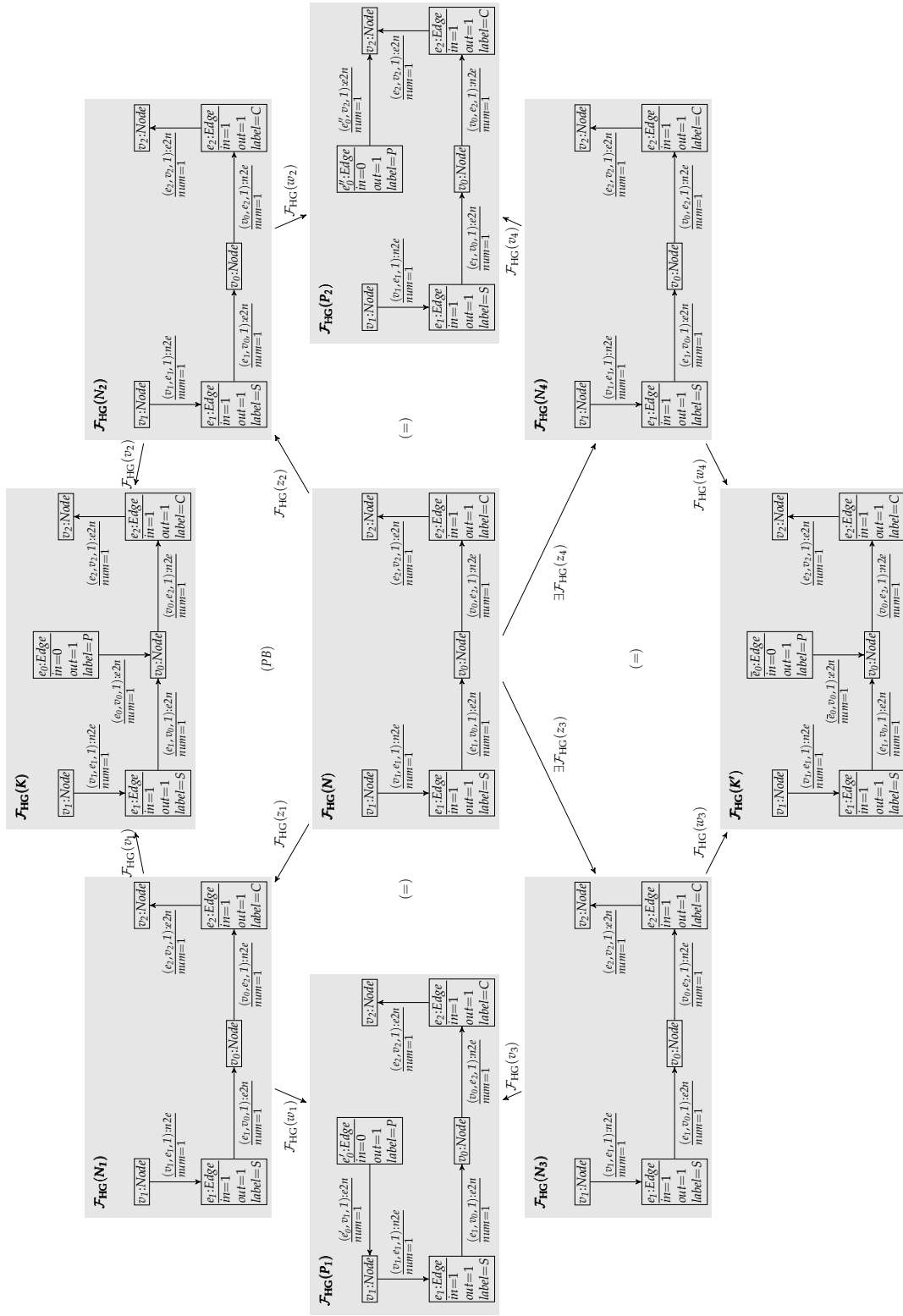


Figure 56: A critical pair diagram depicting the \mathcal{F}_{HG} -reachable critical pair for the rule pair $(\mathcal{F}_{\text{HG}}(\text{enterServer}), \mathcal{F}_{\text{HG}}(\text{crossC}))$

Figure 57: Strictness diagram for the rule pair $(\mathcal{F}_{\text{HG}}(\text{enterServer}), \mathcal{F}_{\text{HG}}(\text{crossC}))$

first \ second	1	2	3	4	5	6	7	8	9	10
1 F_HG(enterServer)	24	4	2	2	3	3	3	3	3	0
2 F_HG(leaveServer)	12	8	2	2	3	3	3	3	5	0
3 F_HG(crossC)	6	2	8	4	3	3	3	3	2	0
4 F_HG(backC)	6	2	4	8	3	3	3	3	2	0
5 F_HG(crossC_1)	9	3	3	3	31	16	31	16	2	0
6 F_HG(backC_1)	9	3	3	3	16	31	16	16	2	0
7 F_HG(crossC_2)	9	3	3	3	31	16	31	16	2	0
8 F_HG(backC_2)	9	3	3	3	16	16	16	31	2	0
9 F_HG(runP)	1	1	1	1	1	1	1	1	1	0
10 F_HG(removeR)	0	0	0	0	0	0	0	0	0	1

Figure 58: Computed critical pairs for the extended Mobile Processes system without considering any additional invariants

The corresponding workflow (see [Figure 48](#)) for local confluence analysis of a hypergraph transformation system (or a hypergraph grammar) *with* nested application conditions is very similar to that of the plain case described before. The main difference consists in the fact that we have to show in the last step of the workflow, in addition to the strict confluence of all computed \mathcal{F}_{HG} -reachable critical pairs (that are assumed to be additionally compatible with invariants for the case of a translated hypergraph grammar), also the $\text{AC}(\mathcal{F}_{\text{HG}})$ -compatibility of the corresponding extended AC-disregarding transformations according to [Definition 60](#). All other analysis steps from the workflow introduced before remain unchanged. If all computed \mathcal{F}_{HG} -reachable critical pairs (that are additionally compatible with invariants for the case of a translated hypergraph grammar) are strictly $\text{AC}(\mathcal{F}_{\text{HG}})$ -confluent, then we obtain local confluence of the considered hypergraph transformation system (or of the considered hypergraph grammar) *with* nested application conditions by application of [Theorem 187](#).

In the following, we apply our workflow to the extended Mobile Processes scenario where the rules of the corresponding hypergraph transformation system are equipped with NACs and PACs as given in [Figure 13](#) in [Example 7](#) (the start hypergraph from [Figure 11](#) remains unchanged).

Firstly, as for the plain case, we translate the transformation rules *with* NACs and PACs according to [Example 14](#) (see [Figure 39](#) there) and, for the case of the corresponding hypergraph grammar, also the start hypergraph given in [Figure 11](#) using the \mathcal{M} -functor \mathcal{F}_{HG} .

Secondly, we compute all critical pairs using AGG. The resulting table in [Figure 58](#) shows the number of critical pairs computed by AGG for each pair of translated transformation rules. Considering again the corresponding translated hypergraph grammar, we can decrease the number of computed critical pairs by using the AGG internal type graph with multiplicities (given in [Figure 50](#)) as can be seen in the table in [Figure 59](#).

Thirdly, trying to exclude further computed critical pairs from the subsequent analysis, we state additional invariants and check, similarly to the plain case, whether the considered translated hypergraph grammar *with* NACs and PACs satisfies them. Using

⁷ Note that when considering a hypergraph grammar we expect that a result similar to [Theorem 18](#) is also valid.

first \ second	1	2	3	4	5	6	7	8	9	10
1 $\mathcal{F}_{HG}(\text{enterServer})$	9	2	2	1	3	1	3	1	2	0
2 $\mathcal{F}_{HG}(\text{leaveServer})$	4	3	0	2	0	3	0	3	3	0
3 $\mathcal{F}_{HG}(\text{crossC})$	4	0	3	2	0	3	0	3	0	0
4 $\mathcal{F}_{HG}(\text{backC})$	1	2	2	3	3	1	3	1	2	0
5 $\mathcal{F}_{HG}(\text{crossC}_1)$	7	0	0	3	1	7	1	7	0	0
6 $\mathcal{F}_{HG}(\text{backC}_1)$	2	3	3	2	7	4	7	3	2	0
7 $\mathcal{F}_{HG}(\text{crossC}_2)$	7	0	0	3	1	7	1	7	0	0
8 $\mathcal{F}_{HG}(\text{backC}_2)$	2	3	3	2	7	4	7	4	2	0
9 $\mathcal{F}_{HG}(\text{runP})$	1	1	0	1	0	1	0	1	1	0
10 $\mathcal{F}_{HG}(\text{removeR})$	0	0	0	0	0	0	0	0	0	1

Figure 59: Computed critical pairs for the extended Mobile Processes system considering the AGG type graph with multiplicities

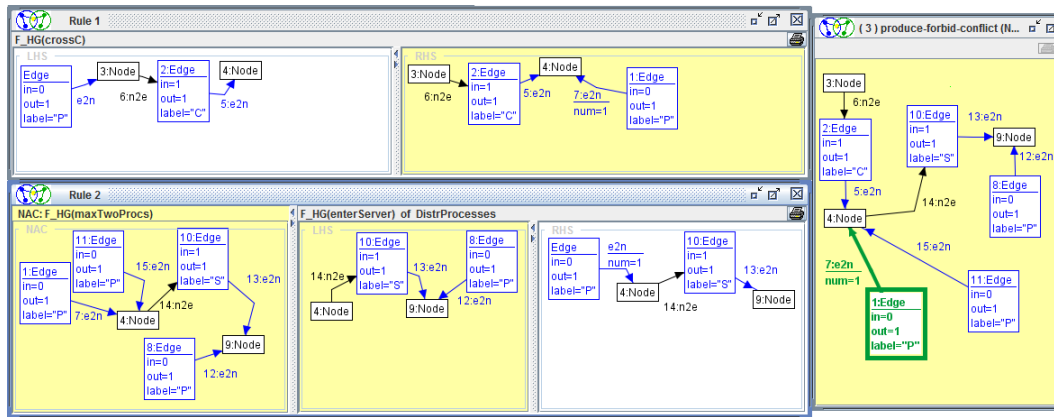


Figure 60: A critical pair computed by AGG for the rule pair $(\mathcal{F}_{HG}(\text{crossC}), \mathcal{F}_{HG}(\text{enterServer}))$ with application conditions non-reachable from the translated start hypergraph by rule application

then tools as e.g. AutoGraph [278], we can identify computed critical pairs whose overlapping graphs cannot be contained in a graph that is reachable from the start graph of the translated transformation system by rule application. An example for such a critical pair is given in Figure 60. The depicted produce-forbid-conflict arises if the application of one rule (here $\mathcal{F}_{HG}(\text{crossC})$) generates graph elements (here the Process hyperedge 1 marked in bold together with the edge 7) in a way that a graph structure would occur that is prohibited by a NAC of another rule (here the rule $\mathcal{F}_{HG}(\text{enterServer})$) that would not be applicable anymore since two Process hyperedges 1 and 11 are already connected to the node 4). The overlapping graph given in Figure 60 on the right includes a sub-graph (containing the Connection hyperedge 2, the node 4, the Server hyperedge 10 as well as the edges 5 and 14 between them) that is not a part of the start hypergraph from Figure 11 translated by \mathcal{F}_{HG} and that can also not be reached from this translated start hypergraph by rule application.

Fourthly, we select those computed critical pairs that are \mathcal{F}_{HG} -reachable, i.e., we only retain those critical pairs whose overlapping graphs correspond to valid hypergraphs translated by \mathcal{F}_{HG} . Non- \mathcal{F}_{HG} -reachable critical pairs, as e.g. a critical pair constructed for

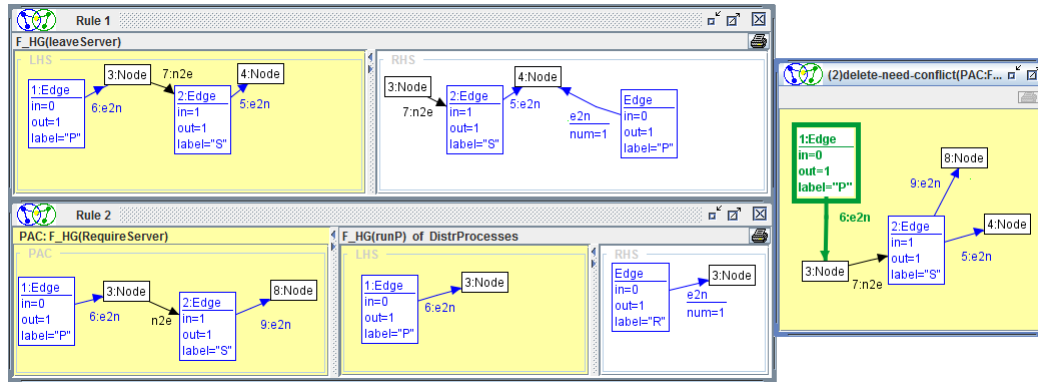


Figure 61: A non- \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{leaveServer}), \mathcal{F}_{\text{HG}}(\text{runP}))$ with application conditions

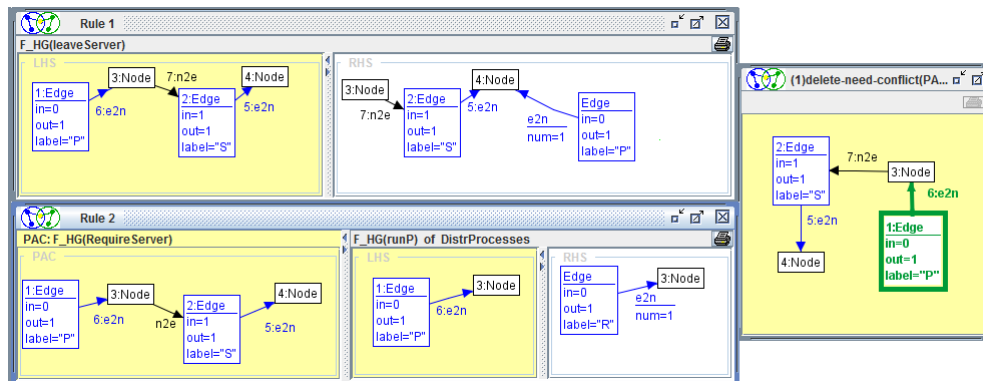


Figure 62: An \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{leaveServer}), \mathcal{F}_{\text{HG}}(\text{runP}))$ with application conditions

the rule pair $(\mathcal{F}_{\text{HG}}(\text{leaveServer}), \mathcal{F}_{\text{HG}}(\text{runP}))$ given in Figure 61, must be excluded from the further strict $\text{AC}(\mathcal{F}_{\text{HG}})$ -confluence analysis according to Theorem 18. The illustrated delete-need-conflict⁸ for this critical pair shows, similarly to the plain case, a non- \mathcal{F}_{HG} -reachable overlapping graph where a Server hyperedge 2 has two outgoing edges 5 and 9 connecting it to two different nodes. For this situation, an original hypergraph does not exist because of the explanations for the plain case.

The critical pairs for the application of rules $\mathcal{F}_{\text{HG}}(\text{leaveServer})$ and $\mathcal{F}_{\text{HG}}(\text{runP})$ as well as $\mathcal{F}_{\text{HG}}(\text{enterServer})$ and $\mathcal{F}_{\text{HG}}(\text{backC1})$, which are given in AGG-notation in Figure 62 and Figure 63, respectively, are examples of computed critical pairs that are \mathcal{F}_{HG} -reachable (and, in the case of the corresponding translated hypergraph grammar, also compatible with the invariants). The first conflict given in Figure 62 is a delete-need-conflict, which arises when the application of one rule (here $\mathcal{F}_{\text{HG}}(\text{leaveServer})$) deletes graph elements (here the Process hyperedge 1 marked in bold together with the edge 6) in a way that a PAC of another rule (here the PAC of $\mathcal{F}_{\text{HG}}(\text{runP})$) cannot be matched into the remaining graph structure anymore. The second conflict given in Figure 63 is a delete-use-conflict, which is caused when a process can “choose” between entering a server or passing a branched connection backwards.

⁸ We have a delete-need-conflict if one rule application deletes graph elements in a way that a PAC of another rule cannot be matched into the remaining graph structure anymore.

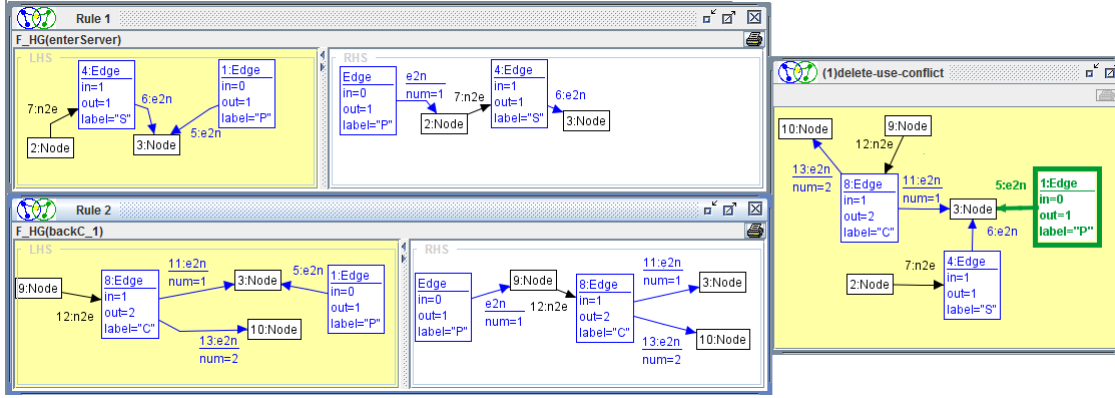


Figure 63: An \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{enterServer}), \mathcal{F}_{\text{HG}}(\text{backC1}))$ with application conditions

Finally, by showing the strict $\text{AC}(\mathcal{F}_{\text{HG}})$ -confluence (i.e., plain strict confluence and $\text{AC}(\mathcal{F}_{\text{HG}})$ -compatibility) of all remaining critical pairs (which are \mathcal{F}_{HG} -reachable and, in the case of the corresponding translated hypergraph grammar, also compatible with the invariants) we conclude also for the extended Mobile Processes system that it is locally confluent by applying Theorem 18. We obtain the plain strict confluence of the remaining critical pairs similarly to the case *without* nested application conditions as explained before. Thereby we adopt the strict confluence analysis results from the plain case for all remaining critical pairs of rules containing no application conditions and analyze in a similar way the plain strict confluence of all critical pairs for the additional delete-need- and produce-forbid-conflicts. The $\text{AC}(\mathcal{F}_{\text{HG}})$ -compatibility of the remaining critical pairs can then be shown similarly to Example 16. For presentation purposes we avoid giving more detailed explanations and pictures concerning the check of the strictness condition and the $\text{AC}(\mathcal{F}_{\text{HG}})$ -compatibility of the remaining critical pairs.

7.4 TERMINATION, CONFLUENCE, AND FUNCTIONAL BEHAVIOR OF HYPERGRAPH TRANSFORMATION SYSTEMS

In this section, we apply termination, confluence, and functional behavior results from Section 4.3 to the \mathcal{M} -functor \mathcal{F}_{HG} between the categories of hypergraphs and typed attributed graphs. Furthermore, we discuss for our Mobile Processes systems introduced in Example 7 whether the mentioned semantical properties hold.

According to Theorems 12, 13 and Remarks 14, 15, the analysis results for termination, confluence, and functional behavior of hypergraph transformation systems without or with nested application conditions can be directly derived using the requirements that were already shown to hold for the \mathcal{M} -functor \mathcal{F}_{HG} when we considered the functorial transfer of behavior and local confluence for hypergraph transformation systems. We formulate and show the mentioned results in the following two theorems.

Theorem 19 (\mathcal{F}_{HG} -Transfer of Termination).

Consider a hypergraph transformation system $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P))$ where P are rules without or with nested application conditions, and the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Then $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$ is terminating iff $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P))$ is \mathcal{F}_{HG} -terminating.

Proof.

In order to use the results from Theorem 12 and Remark 14 for the case of transformations without or with nested application conditions, we have to show that \mathcal{F}_{HG} translates and creates (direct) transformations without or with nested application conditions according to Theorems 1 and 3, i.e., we have to show that $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ has initial pushouts, \mathcal{F}_{HG} is an \mathcal{M} -functor according to Definition 41 as well as that \mathcal{F}_{HG} creates injective and general morphisms and preserves initial pushouts. In fact, we have that $\mathcal{F}_{\text{HG}}(\mathcal{M}_1) \subseteq \mathcal{M}_2$, i.e., \mathcal{F}_{HG} preserves injectivity of morphisms by Lemma 27 and \mathcal{F}_{HG} preserves pushouts along injective morphisms according to Lemma 28. Moreover, according to Lemma 7 the category $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ has initial pushouts. Finally, creation of general morphisms by \mathcal{F}_{HG} is shown in Lemma 30, creation of injective morphisms by \mathcal{F}_{HG} is verified in Lemma 31, and preservation of initial pushouts follows from Lemma 34. \square

Theorem 20 (\mathcal{F}_{HG} -Transfer of Confluence and Functional Behavior).

Consider a hypergraph transformation system $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P))$ where P are rules without or with nested application conditions, and the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Then $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$ is locally confluent and terminating iff $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P))$ is locally confluent for all translated transformation spans and \mathcal{F}_{HG} -terminating. Moreover, $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$ is confluent and has functional behavior if $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(P))$ is locally confluent for all translated transformation spans and \mathcal{F}_{HG} -terminating.

Proof.

In order to use the results from Theorem 13 and Remark 15 for the case of transformations without or with nested application conditions, we have again to show that \mathcal{F}_{HG} translates and creates (direct) transformations without or with nested application conditions according to Theorems 1 and 3. This can be done equivalently to the proof of Theorem 19. \square

After we have shown that the general results of our approach concerning termination, confluence, and functional behavior are applicable to the \mathcal{M} -functor \mathcal{F}_{HG} , we now want to analyze, using these results, whether the mentioned semantical properties hold for our Mobile Processes systems from Example 7.

As already discussed at the end of the previous section, both hypergraph transformation systems from our running example (as well as the corresponding hypergraph grammars, i.e., hypergraph transformation systems with the start hypergraph at hand given in Figure 11) are locally confluent.

Considering the sets of translated transformation rules given in Figure 38 and Figure 39 for the case of transformation systems without application conditions or with NACs and PACs, respectively, (and additionally considering the start hypergraph translated by \mathcal{F}_{HG} for the case of the corresponding hypergraph grammars) we can easily see that the translated hypergraph transformation systems (as well as the corresponding translated hypergraph grammars) are not \mathcal{F}_{HG} -terminating because the rules for moving a process over a connection or over a server and back can be applied alternately unlimited often, which implies that also the original hypergraph transformation systems (as well as the corresponding hypergraph grammars) are not terminating according to Theorem 19⁹.

Hence, while we believe that the hypergraph transformation systems from our running example are confluent but have no functional behavior, this cannot be obtained using Theorem 20. The considered hypergraph transformation systems are confluent because, on the one hand, inverse rules can be applied for moving processes through the network and, on the other hand, the rules for process running and removal applied in one of the two transformation sequences can then also be applied equivalently in the other transformation sequence, which implies altogether that a common hypergraph for joining two transformation sequences starting with the same hypergraph is always derivable. Moreover, both hypergraph transformation systems have no functional behavior because, when considering e.g. a network without servers but with a process attached, the process can permanently move through the network but is never removed (as this requires a server) implying that there is no finite transformation sequence to a unique hypergraph, to which no transformation rules are applicable anymore.

However, when considering for both hypergraph transformation systems the corresponding hypergraph grammars, we can easily see that, in addition to the confluence that holds according to the explanations above, the hypergraph grammars turn out also to have functional behavior because a unique hypergraph can be obtained, to which no transformation rules are applicable anymore, by removing the processes (at some selection of servers) resulting in the hypergraph that only contains the network structure. Though, this analysis result still cannot be obtained using Theorem 20⁹.

For our Mobile Processes example we do not use AGG for the automated verification of the termination property because AGG is only capable to analyze termination for typed graph transformation systems with injective rules, injective match morphisms, and injective NACs.

⁹ Note that when considering a hypergraph grammar, we expect that the results similar to Theorems 19 and 20 are also valid.

Part IV

APPLICATION TO PTI NET TRANSFORMATION SYSTEMS

RESTRICTED FUNCTOR BETWEEN CATEGORIES OF PTI NETS AND TYPED ATTRIBUTED GRAPHS

This chapter is structured similar to Chapter 5. We show that we can apply the general approach from Chapters 3 and 4 to the \mathcal{M} -adhesive categories of Petri nets with individual tokens and typed attributed graphs introduced in Subsections 2.4.3 and 2.4.1, respectively. In Section 8.1 we consider the subcategory of typed attributed graphs over the specific typed attributed graph $PNTG$ allowing us to represent PTI nets as typed attributed graphs. We use subsequently this subcategory in Section 8.2 as the target category for the construction of a restricted functor where we define the translation of objects and morphisms from the category of PTI nets \mathbf{PTINet} into the corresponding objects and morphisms of the category of typed attributed graphs over the specific attributed type graph $PNTG$ $\mathbf{AGraphs}_{PNTG}$. We can only construct a functor $\mathcal{F}_{PTI} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{PNTG}|_{\mathcal{M}_2}$ between the categories restricted to \mathcal{M} -morphisms (with the class of all injective \mathbf{PTINet} -morphisms \mathcal{M}_1 and the class of all injective $\mathbf{AGraphs}_{PNTG}$ -morphisms \mathcal{M}_2) but not a functor $\mathcal{F}_{PTI} : (\mathbf{PTINet}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{PNTG}, \mathcal{M}_2)$ because \mathcal{F}_{PTI} is not well-defined on non-injective morphisms (see the counterexample in Figure 67 where $\mathcal{F}_{PTI}(f)$ does not preserve attributes in and w_{pre}).

Note that we do not use Petri nets with “classical initial markings”, known as Petri net systems [267], because the corresponding \mathcal{M} -adhesive category requires a class \mathcal{M} leading to Petri net rules that are marking preserving, which is not adequate to model firing steps as direct transformations.

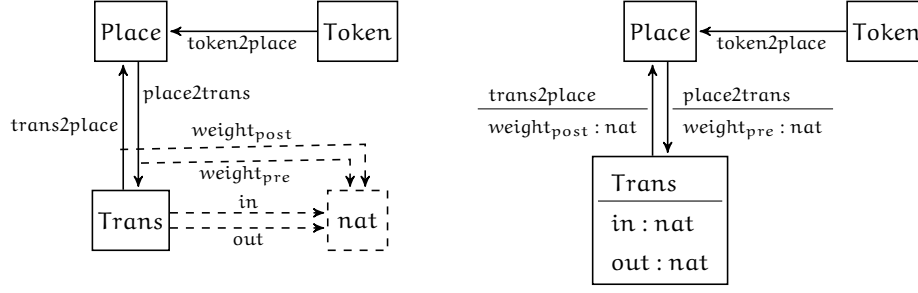
8.1 TYPED ATTRIBUTED GRAPHS OVER THE PTI NET TYPE GRAPH $PNTG$

In this section, we introduce the category of typed attributed graphs, which we call $\mathbf{AGraphs}_{PNTG}$, constructed over the specific typed attributed graph $PNTG$ (see Figure 64) allowing for the representation of Petri nets with individual tokens as E-graphs. Thereby nodes and edges correspond to the graph nodes (V_G) and graph edges (E_G), respectively, dashed nodes are the data nodes (V_D), and dashed edges represent the edges for node and edge attribution. The E-graph of $PNTG$ is shown in Figure 64 to the left and its attributed notation to the right.

For the construction of our restricted functor we use the subcategory $\mathbf{AGraphs}_{PNTG}^{\mathbb{N}}$ of $\mathbf{AGraphs}_{PNTG}$ where all objects are restricted to the data type NAT , $V_D^G = \mathbb{N}$, and all data type morphisms as well as V_D -components of the E-graph morphisms are restricted to identities. Morphisms in $\mathbf{AGraphs}_{PNTG}^{\mathbb{N}}$ are defined componentwise and are *type-compatible*.

The construction of pushouts and pullbacks in $\mathbf{AGraphs}_{PNTG}^{\mathbb{N}}$ can be obtained componentwise, as in $\mathbf{AGraphs}_{PNTG}$, with the identical data type component.

As already mentioned in Subsection 2.4.1, the category $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ is \mathcal{M} -adhesive for each type graph ATG where \mathcal{M} -morphisms are injective with the isomorphic data type part. Hence also the special case of $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ with $ATG = PNTG$ and

Figure 64: Attributed type graph $PNTG$

the subclass $\mathcal{M} = \mathcal{M}_2$ of all injective typed attributed graph morphisms is \mathcal{M} -adhesive. Moreover, the fact that $(\mathbf{AGraphs}_{PNTG}, \mathcal{M}_2)$ is an \mathcal{M} -adhesive category implies that also $(\mathbf{AGraphs}_{PNTG}^{\mathbb{N}}, \mathcal{M}_2^{\mathbb{N}})$ is an \mathcal{M} -adhesive category with the class \mathcal{M}_2 restricted to $\mathcal{M}_2^{\mathbb{N}}$ with the identical data type component.

The initial object $\emptyset^{\mathbb{N}}$ in $\mathbf{AGraphs}_{PNTG}^{\mathbb{N}}$ is empty excepting the data type part and $\emptyset^{\mathbb{N}} = \mathcal{F}_{PTI}(\emptyset)$ where \emptyset is an initial PTI net. In the following, we use the short notation $(\mathbf{AGraphs}_{PNTG}, \mathcal{M}_2)$ for the \mathcal{M} -adhesive category of typed attributed graphs over $PNTG$ instead of the long notation $(\mathbf{AGraphs}_{PNTG}^{\mathbb{N}}, \mathcal{M}_2^{\mathbb{N}})$.

All graphs over $PNTG$ are considered as a graph representation of PTI nets. Thus, all possible components of PTI nets have to be included in $PNTG$. The meaning of every depicted element of $PNTG$ is as follows. The nodes *Place*, *Trans*, and *Token* represent the elements of PTI nets that can be depicted as nodes in typed attributed graphs and hence give the possible node typing. The edges *place2trans*, *trans2place*, and *token2place* represent all kinds of edges that are valid in PTI nets and describe therefore the possible edge typing. The edges *place2trans* and *trans2place* have the attributes *weight_{pre}* and *weight_{post}* that contain weights for edges in the pre- and postdomain of transitions in the corresponding PTI net. The node *Trans* has the attributes *in* and *out* that describe a number of edges in the pre- and postdomain of transitions in the corresponding PTI net. The node *nat* is a data node that defines the type for the attributes and corresponds to the single sort symbol in the signature $\Sigma - nat$ (see Figure 7).

8.2 TRANSLATION OF PTI NETS INTO TYPED ATTRIBUTED GRAPHS

We begin this section with the definition of the restricted functor $\mathcal{F}_{PTI} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{PNTG}|_{\mathcal{M}_2}$ on objects and restricted morphisms. This functor definition shows how PTI nets can be translated properly into the corresponding typed attributed graphs over the attributed type graph $PNTG$.

Definition 64 (Restricted Functor \mathcal{F}_{PTI} [207]).

Consider PTI nets $NI = (P, T, pre, post, I, m)$ and $NI_i = (P_i, T_i, pre_i, post_i, I_i, m_i)$ for $i \in \{1, 2\}$.

- **Translation of objects:**

We define the object $\mathcal{F}_{PTI}(NI) = ((G, NAT), type)^1$ in $\mathbf{AGraphs}_{PNTG}$ with the morphism $type : (G, NAT) \rightarrow (PNTG, D_{fin})$ and the E-graph $G = (V_G^G, V_D^G = \mathbb{N}, E_G^G, E_{NA}^G, E_{EA}^G, (s_j^G, t_j^G)_{j \in \{G, NA, EA\}})$ as follows where we use the following abbreviations:

¹ In the following, we also use the short notation $\mathcal{F}_{PTI}(NI) = G$.

$token2place \triangleq to2p, place2trans \triangleq p2t, trans2place \triangleq t2p, weight_{pre} \triangleq w_{pre},$
 $weight_{post} \triangleq w_{post}, pre(t)(p) = n_p \in \mathbb{N}$ for $pre(t) = \sum_{p \in P} n_p \cdot p \in P^\oplus$, and
 $post(t)(p) = n_p \in \mathbb{N}$ for $post(t) = \sum_{p \in P} n_p \cdot p \in P^\oplus$:

$$V_G^G = P \uplus T \uplus I \text{ (graph nodes),}$$

$$E_G^G = E_{to2p}^G \uplus E_{p2t}^G \uplus E_{t2p}^G \text{ (graph edges) with}$$

$$E_{to2p}^G = \{(x, p) \in I \times P \mid m(x) = p\},$$

$$E_{p2t}^G = \{(p, t) \in P \times T \mid pre(t)(p) > 0\},$$

$$E_{t2p}^G = \{(t, p) \in T \times P \mid post(t)(p) > 0\},$$

$$E_{NA}^G = E_{in}^G \uplus E_{out}^G \text{ (node attribute edges) with}$$

$$E_{in}^G = \{(t, n, in) \mid (t, n) \in T \times \mathbb{N} \wedge |\bullet t| = n\},$$

$$E_{out}^G = \{(t, n, out) \mid (t, n) \in T \times \mathbb{N} \wedge |t \bullet| = n\}$$

where $\bullet t$ and $t \bullet$ are the pre- and postdomains of $t \in T$ with cardinalities
 $|\bullet t|$ and $|t \bullet|$, respectively,

$$E_{EA}^G = E_{w_{pre}}^G \uplus E_{w_{post}}^G \text{ (edge attribute edges) with}$$

$$E_{w_{pre}}^G = \{((p, t), n) \in E_{p2t}^G \times \mathbb{N} \mid pre(t)(p) = n\},$$

$$E_{w_{post}}^G = \{((t, p), n) \in E_{t2p}^G \times \mathbb{N} \mid post(t)(p) = n\},$$

(and the corresponding source and target functions)

$$s_G^G, t_G^G : E_G^G \rightarrow V_G^G \text{ defined by } s_G^G(a, b) = a, t_G^G(a, b) = b,$$

$$s_{NA}^G : E_{NA}^G \rightarrow V_G^G \text{ defined by } s_{NA}^G(t, n, x) = t,$$

$$t_{NA}^G : E_{NA}^G \rightarrow \mathbb{N} \text{ defined by } t_{NA}^G(t, n, x) = n,$$

$$s_{EA}^G : E_{EA}^G \rightarrow E_G^G \text{ defined by } s_{EA}^G((x, y), n) = (x, y),$$

$$t_{EA}^G : E_{EA}^G \rightarrow \mathbb{N} \text{ defined by } t_{EA}^G((x, y), n) = n.$$

To simplify the notation, we flatten nested tuples most of the time, i.e., the tuple $((x, y), z)$ is written (x, y, z) .

The **AGraphs**_{PNTG}-morphism type $(G, NAT) \rightarrow (PNTG, D_{fin})$ is given by the final morphism of the data types from NAT to the final algebra D_{fin} and $type^G : G \rightarrow PNTG$ is given by the E-graph morphism $type^G = (type_{V_G}, type_{V_D}, type_{E_G}, type_{E_{NA}}, type_{E_{EA}})$ where

$$type_{V_G} : V_G^G \rightarrow V_G^{PNTG} \text{ with } x \mapsto \text{Place (if } x \in P), x \mapsto \text{Trans (if } x \in T),$$

$$x \mapsto \text{Token (if } x \in I),$$

$$type_{V_D} : \mathbb{N} \rightarrow D_{fin, nat} \text{ with } x \mapsto \text{nat (if } x \in \mathbb{N}),$$

$$type_{E_G} : E_G^G \rightarrow E_G^{PNTG} \text{ with } x \mapsto y \text{ for } x \in E_y^G \text{ and } y \in \{to2p, p2t, t2p\},$$

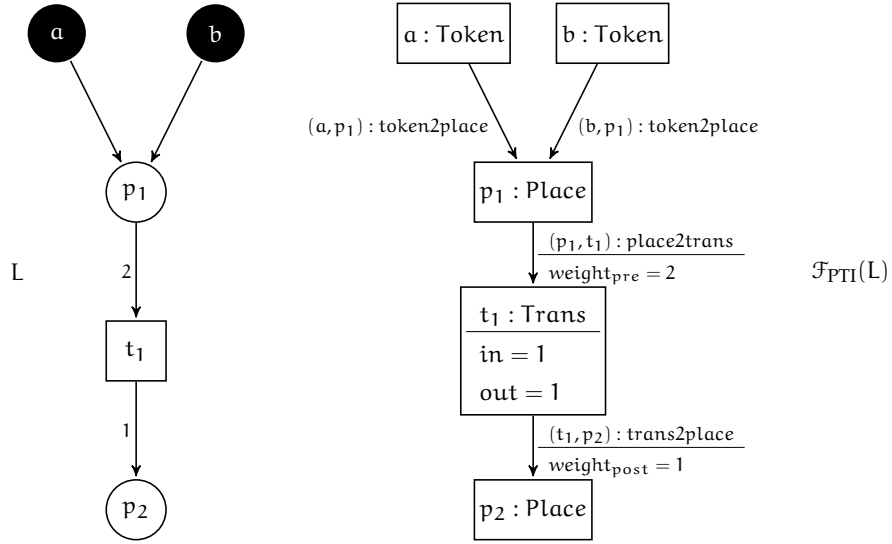
$$type_{E_{NA}} : E_{NA}^G \rightarrow E_{NA}^{PNTG} \text{ with } x \mapsto y \text{ for } x \in E_y^G \text{ and } y \in \{in, out\},$$

$$type_{E_{EA}} : E_{EA}^G \rightarrow E_{EA}^{PNTG} \text{ with } x \mapsto y \text{ for } x \in E_y^G \text{ and } y \in \{w_{pre}, w_{post}\}.$$

- **Translation of morphisms:**

For an arbitrary PTI net morphism $f : NI_1 \rightarrow NI_2$ with $f = (f_P : P_1 \rightarrow P_2, f_T : T_1 \rightarrow$

² $pre(t)(p)$ resp. $post(t)(p)$ denotes the weight n_p of an edge in the predomain resp. postdomain of a transition connecting a place p with a transition t . $pre(t) = \sum_{p \in P} n_p \cdot p \in P^\oplus$ resp. $post(t) = \sum_{p \in P} n_p \cdot p \in P^\oplus$ is then a free commutative monoid giving a sum of weights of edges in the predomain resp. postdomain of a transition t .

Figure 65: PTI net L and its corresponding typed attributed graph $\mathcal{F}_{PTII}(L)$

$T_2, f_I : I_1 \rightarrow I_2) \in \mathcal{M}_1$, i.e., f_P, f_T, f_I are injective, we define $\mathcal{F}_{PTII}(f) : \mathcal{F}_{PTII}(NI_1) \rightarrow \mathcal{F}_{PTII}(NI_2)$ with $\mathcal{F}_{PTII}(NI_i) = (V_{iG}, \mathbb{N}, E_{iG}, E_{iNA}, E_{iEA}, (s_{ij}, t_{ij})_{j \in \{G, NA, EA\}})$ for $i \in \{1, 2\}$ by $\mathcal{F}_{PTII}(f) = f' = (f'_{V_G}, f'_{V_D} = id_{\mathbb{N}}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}})$ where

$f'_{V_G} : V_{1G} \rightarrow V_{2G}$ with $V_{iG} = P_i \uplus T_i \uplus I_i$ for $i \in \{1, 2\}$ is given by

$$f'_{V_G} = f_P \uplus f_T \uplus f_I,$$

$f'_{E_G} : E_{1G} \rightarrow E_{2G}$ with $E_{iG} = E_{i_{to2p}} \uplus E_{i_{p2t}} \uplus E_{i_{t2p}}$ for $i \in \{1, 2\}$ is given by

$$f'_{E_G}(x, p) = (f_I(x), f_P(p)) \text{ for } (x, p) \in E_{1_{to2p}},$$

$$f'_{E_G}(p, t) = (f_P(p), f_T(t)) \text{ for } (p, t) \in E_{1_{p2t}},$$

$$f'_{E_G}(t, p) = (f_T(t), f_P(p)) \text{ for } (t, p) \in E_{1_{t2p}},$$

$f'_{E_{NA}} : E_{1NA} \rightarrow E_{2NA}$ with $E_{iNA} = E_{i_{in}} \uplus E_{i_{out}}$ for $i \in \{1, 2\}$ is given by

$$f'_{E_{NA}}(t, n, x) = (f_T(t), n, x) \text{ for } (t, n, x) \in E_{1_{in}} \uplus E_{1_{out}} \text{ and } x \in \{in, out\},$$

$f'_{E_{EA}} : E_{1EA} \rightarrow E_{2EA}$ with $E_{iEA} = E_{i_{w_{pre}}} \uplus E_{i_{w_{post}}}$ for $i \in \{1, 2\}$ is given by

$$f'_{E_{EA}}(p, t, n) = (f_P(p), f_T(t), n) \text{ for } (p, t, n) \in E_{1_{w_{pre}}} \text{ and } n \in \mathbb{N},$$

$$f'_{E_{EA}}(t, p, n) = (f_T(t), f_P(p), n) \text{ for } (t, p, n) \in E_{1_{w_{post}}} \text{ and } n \in \mathbb{N}.$$

An example for using the restricted functor \mathcal{F}_{PTII} on objects is shown in Figure 65 where the typed attributed graph $\mathcal{F}_{PTII}(L)$ to the right is the translation of the corresponding PTI net L to the left. Another example for the translation of objects is given in Figure 68 where four depicted typed attributed graphs result from the translation of the corresponding PTI nets in Figure 19.

An example for the *type*-morphism components $type_{V_G}$ and $type_{E_G}$ (depicted by dashed arrows) is given in Figure 66. We do not show the other three *type*-morphism components $type_{V_D}$, $type_{E_{NA}}$, and $type_{E_{EA}}$ in order to improve the clarity of the illustration.

As already mentioned before, we can construct the functor $\mathcal{F}_{PTII} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\mathbf{PTNG}}|_{\mathcal{M}_2}$ between the categories restricted to \mathcal{M} -morphisms, but not a functor

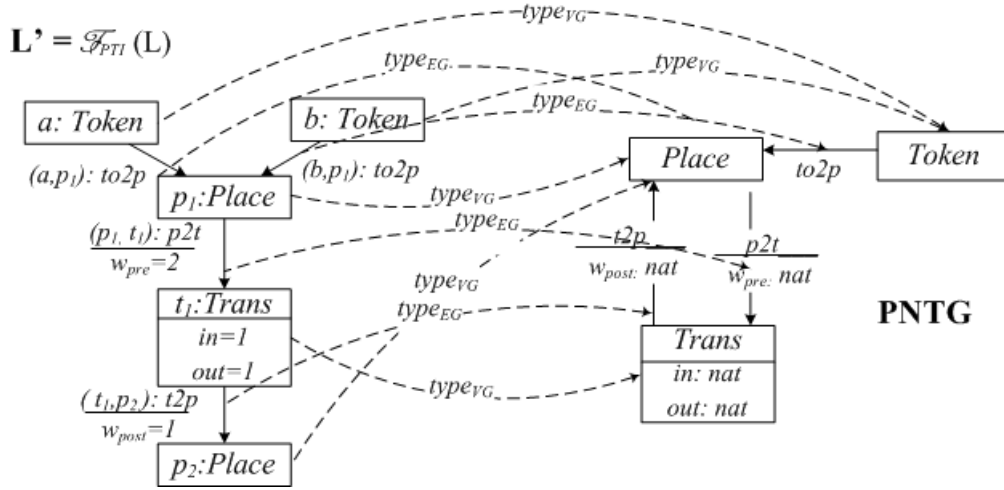
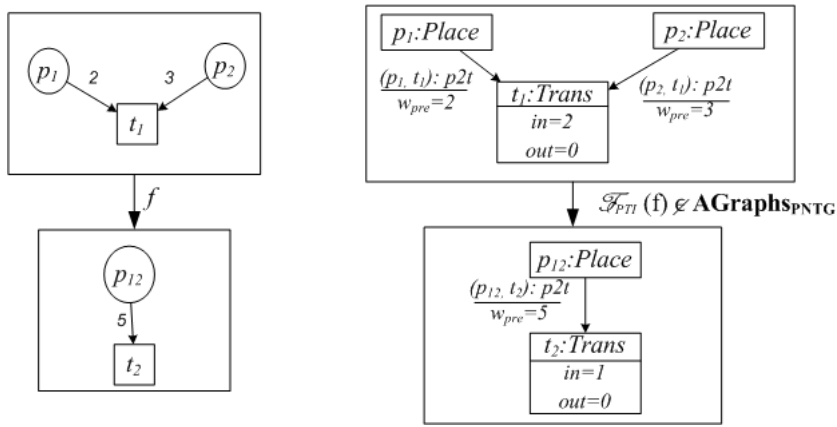
Figure 66: Example for the $\mathbf{AGraphs}_{\text{PNTG}}$ -morphism type_{VG} and type_{EG} 

Figure 67: Counterexample for translation of general (non-injective) morphisms

$\mathcal{F}_{\text{PTI}} : (\mathbf{PTINet}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, because \mathcal{F}_{PTI} is not well-defined on non-injective morphisms. A counterexample for the translation of non-injective morphisms is given in Figure 67 where on the graph side the attributes in and w_{pre} are not preserved by the morphism $\mathcal{F}_{\text{PTI}}(f)$.

Examples for injective PTI net morphisms are shown in Figure 19, while the corresponding morphisms translated by \mathcal{F}_{PTI} are given in Figure 68.

It is important to ensure that the translation of \mathcal{M} -morphisms, defined in the way described before, is well-defined for all PTI net morphisms $f : \text{NI}_1 \rightarrow \text{NI}_2$. We summarize the sufficient properties for the well-definedness of the morphism translation in the following lemma.

Lemma 42 (Well-Definedness of PTI Net Morphism Translation [207]).

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, and the restricted functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then for each PTI net morphism $f : \text{NI}_1 \rightarrow \text{NI}_2$ in \mathcal{M}_1 the corresponding morphism $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(\text{NI}_1) \rightarrow \mathcal{F}_{\text{PTI}}(\text{NI}_2)$ is well-defined in $\mathbf{AGraphs}_{\text{PNTG}}$ with $\mathcal{F}_{\text{PTI}}(f) \in \mathcal{M}_2$. Moreover, \mathcal{F}_{PTI} preserves compositionality, injectivity of morphisms, inclusions, and identities.

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 361](#) showing the following steps:

1. The components of $\mathcal{F}_{\text{PTI}}(f)$ are well-defined w.r.t. the codomain.
2. The components of $\mathcal{F}_{\text{PTI}}(f)$ are compatible with the source and target functions.
3. The components of $\mathcal{F}_{\text{PTI}}(f)$ are compatible with the typing morphisms.
4. Compositionality axiom holds for \mathcal{F}_{PTI} .
5. $f \in \mathcal{M}_1$ (inclusion, identity) implies that $\mathcal{F}_{\text{PTI}}(f) \in \mathcal{M}_2$ (inclusion, identity).

□

Using Lemma [42](#) we obtain that \mathcal{F}_{PTI} is a functor, because \mathcal{F}_{PTI} is well-defined and preserves identities and composition.

Similarly to Chapter 3, we show in this chapter that we can apply the theoretical results for translation and creation of rule applicability without or with nested application conditions, (direct) transformations, parallel and sequential independence of transformations as well as \mathcal{F}_R -bisimilarity and \mathcal{F}_R -transfer of bisimilarity to the concrete restricted functor \mathcal{F}_{PTI} between the categories of Petri nets with individual tokens and typed attributed graphs. Therefore, we show in Section 9.1 that \mathcal{F}_{PTI} is a restricted \mathcal{M} -functor as well as in Sections 9.2 and 9.3 that \mathcal{F}_{PTI} satisfies the remaining adapted requirements of the theoretical results from Chapter 3, which altogether allows for the instantiation of the mentioned theoretical results for PTI net transformation systems. Note that, as for the functor \mathcal{F}_{HG} , using this instantiation, we obtain the behavioral equivalence of the \mathcal{F}_{PTI} -related parts of the source and the target transformation systems.

9.1 TRANSLATION OF PTI NET TRANSFORMATIONS

As the first step we show in this section that the restricted functor \mathcal{F}_{PTI} preserves pushouts of injective PTI net morphisms. Using this result together with the preservation of injective morphisms by \mathcal{F}_{PTI} , which was already shown in Lemma 42 in the previous chapter, we obtain that \mathcal{F}_{PTI} is a restricted \mathcal{M} -functor and thus we can show that \mathcal{F}_{PTI} translates applicability of PTI net rules without or with nested application conditions, (direct) PTI net transformations as well as parallel and sequential independence of PTI net transformations by application of the general theory.

The results of our general theory are applicable to \mathcal{F}_{PTI} only if \mathcal{F}_{PTI} is a restricted \mathcal{M} -functor. The requirement that remains to be shown for \mathcal{F}_{PTI} to become a restricted \mathcal{M} -functor, is the preservation of pushouts of injective morphisms. This technical property intuitively means that if we have a pushout of injective PTI net morphisms in the category **PTINet** then, applying \mathcal{F}_{PTI} to this diagram, we obtain a pushout of injective typed attributed graph morphisms in the category **AGraphs_{PNTG}**.

Lemma 43 (\mathcal{F}_{PTI} Preserves Pushouts of Injective Morphisms [207]).

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$ and $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, restricted functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ introduced in Definition 64, PTI nets NI_i with PTI net morphisms $f_i = (f_{i_P}, f_{i_T}, f_{i_I})$, and typed attributed graphs $\mathcal{F}_{\text{PTI}}(\text{NI}_i)$ with typed attributed graph morphisms $\mathcal{F}_{\text{PTI}}(f_i) = f'_i = (f'_{i_{V_G}}, f'_{i_{V_D}}, f'_{i_{E_G}}, f'_{i_{E_{NA}}}, f'_{i_{E_{EA}}})$ for $i \in \{0, 1, 2, 3\}$. If (1) is a pushout in **PTINet** with $f_i \in \mathcal{M}_1$ then we have that (2) is a pushout in **AGraphs_{PNTG}** with $\mathcal{F}_{\text{PTI}}(f_i) \in \mathcal{M}_2$ for $i \in \{0, 1, 2, 3\}$.

$$\begin{array}{ccc}
 \text{NI}_0 & \xrightarrow{f_1} & \text{NI}_1 \\
 f_2 \downarrow & (1) & \downarrow f_4 \\
 \text{NI}_2 & \xrightarrow{f_3} & \text{NI}_3
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathcal{F}_{\text{PTI}}(\text{NI}_0) & \xrightarrow{\mathcal{F}_{\text{PTI}}(f_1) = f'_1} & \mathcal{F}_{\text{PTI}}(\text{NI}_1) \\
 \mathcal{F}_{\text{PTI}}(f_2) = f'_2 \downarrow & (2) & \downarrow \mathcal{F}_{\text{PTI}}(f_4) = f'_4 \\
 \mathcal{F}_{\text{PTI}}(\text{NI}_2) & \xrightarrow{\mathcal{F}_{\text{PTI}}(f_3) = f'_3} & \mathcal{F}_{\text{PTI}}(\text{NI}_3)
 \end{array}$$

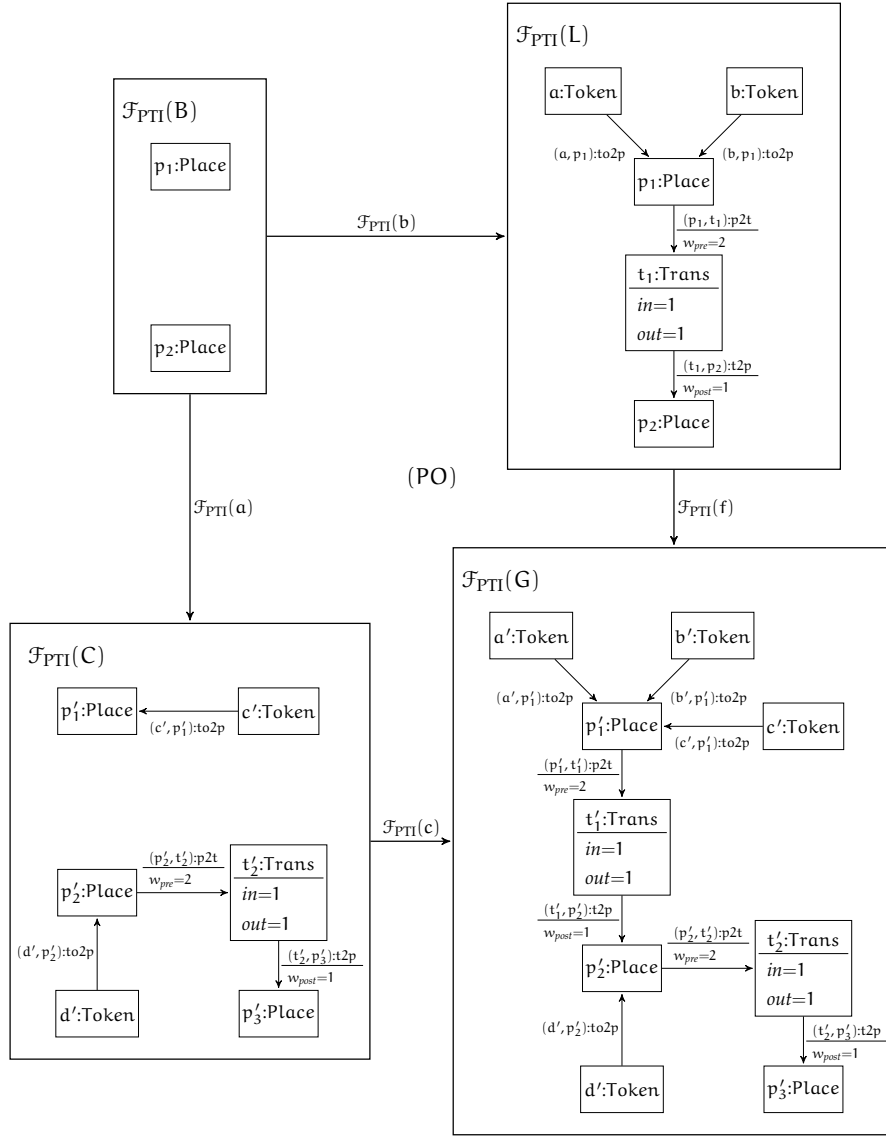


Figure 68: Translated pushout in $\mathbf{AGraphs}_{\mathbf{PNTG}}$ with injective typed attributed graph morphisms $\mathcal{F}_{\text{PTI}}(a)$, $\mathcal{F}_{\text{PTI}}(b)$, $\mathcal{F}_{\text{PTI}}(c)$, and $\mathcal{F}_{\text{PTI}}(f)$

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 366](#). □

An example for the translation of pushouts of injective morphisms is given in [Figure 68](#) where the pushout in [Figure 68](#) is the corresponding translation of the pushout in \mathbf{PTINet} given in [Figure 19](#).

After the sufficient property for the translation of applicability of Petri net rules, (direct) Petri net transformations, and parallel (sequential) independence is shown for \mathcal{F}_{PTI} , we can now formulate and prove our first important result concerning the PTI net application. The following theorem states intuitively that transformation steps, which are possible in a PTI net transformation system, are also possible in the corresponding typed attributed graph transformation system using our concrete restricted functor \mathcal{F}_{PTI} . Note that the following theorem is applicable to PTI net transformation systems containing rules without or with nested application conditions.

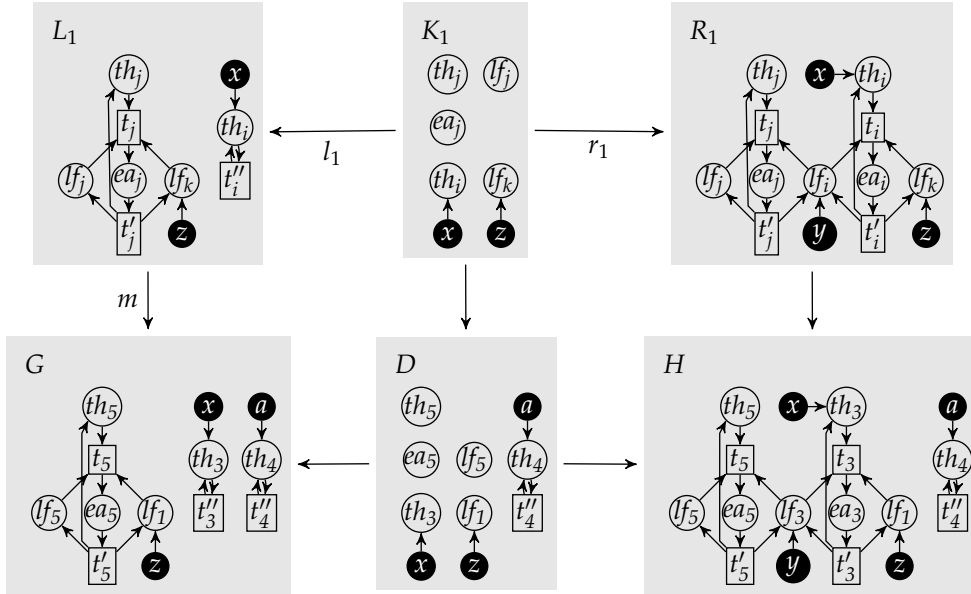


Figure 69: PTI net transformation step for the application of the rule JoinTable

Theorem 21 (Translation of PTI Net Transformations into Typed Attributed Graph Transformations [207]).

Consider \mathcal{M} -adhesive transformation systems $(\mathbf{PTINet}, \mathcal{M}_1, P)$ and $(\mathbf{AGraphs}_{\mathbf{PTNG}}, \mathcal{M}_2, \mathcal{F}_{\mathbf{PTI}}(P))$. The restricted functor $\mathcal{F}_{\mathbf{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\mathbf{PTNG}}|_{\mathcal{M}_2}$ introduced in Definition 64 translates applicability of PTI net rules without or with nested application conditions, construction of (direct) PTI net transformations as well as parallel and sequential independence of PTI net transformations.

Proof.

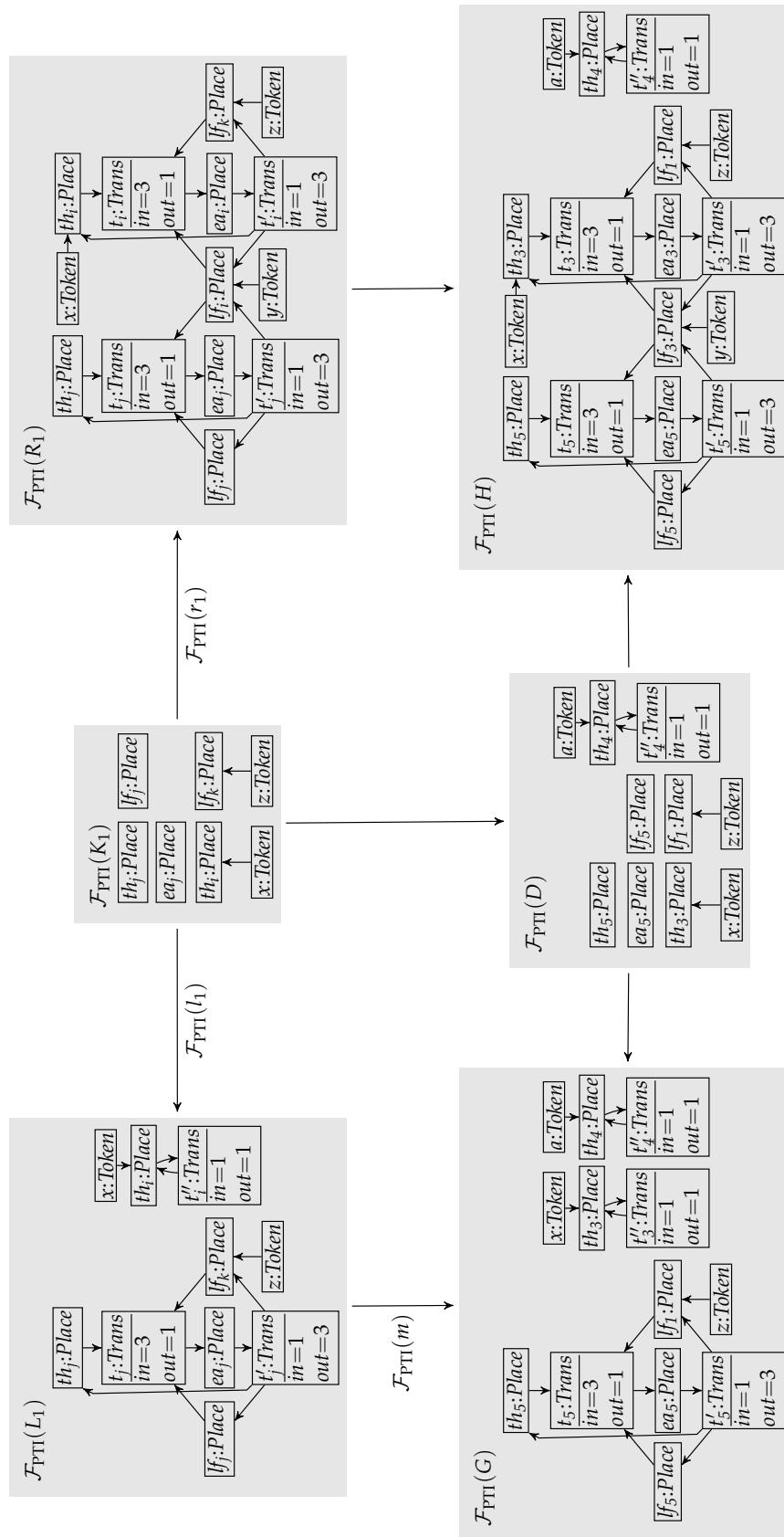
PTI net transformations are (direct) transformations with injective match morphisms only. In order to use the results from Theorems 2 and 7 for the case of transformations without nested application conditions as well as Remark 12 and Theorem 10 for the case of transformations with nested application conditions, we only have to show that $\mathcal{F}_{\mathbf{PTI}}$ is a restricted \mathcal{M} -functor according to Definition 44. In fact, we have $\mathcal{F}_{\mathbf{PTI}}(\mathcal{M}_1) \subseteq \mathcal{M}_2$, i.e., $\mathcal{F}_{\mathbf{PTI}}$ preserves injectivity of morphisms by Lemma 42 and $\mathcal{F}_{\mathbf{PTI}}$ preserves pushouts of injective morphisms according to Lemma 43. \square

Now we show for our running example how PTI net transformation rules can be translated into the corresponding typed attributed graph rules using the restricted \mathcal{M} -functor $\mathcal{F}_{\mathbf{PTI}}$ as well as give an example for a translated transformation step.

Example 17 (PTI Net Transformations Translated by $\mathcal{F}_{\mathbf{PTI}}$).

In this example, we use the PTI net transformation system introduced in Example 12 describing the Mobile Dining Philosophers system.

In Figure 71 and Figure 72 the transformation rules from Example 12 translated by $\mathcal{F}_{\mathbf{PTI}}$ are given for the PTI net transformation system without and with nested application conditions, respectively. Note that for better readability we avoid to indicate the attributes and the typing of the edges. Furthermore, Figure 70 shows a translated transformation step for the application of the translated rule $\mathcal{F}_{\mathbf{PTI}}(\text{JoinTable})$, while the corresponding PTI net transformation step for the application of a PTI net transformation rule JoinTable is given in Figure 69.

Figure 70: Translated transformation step for the application of the rule $\mathcal{F}_{PTII}(\text{JoinTable})$

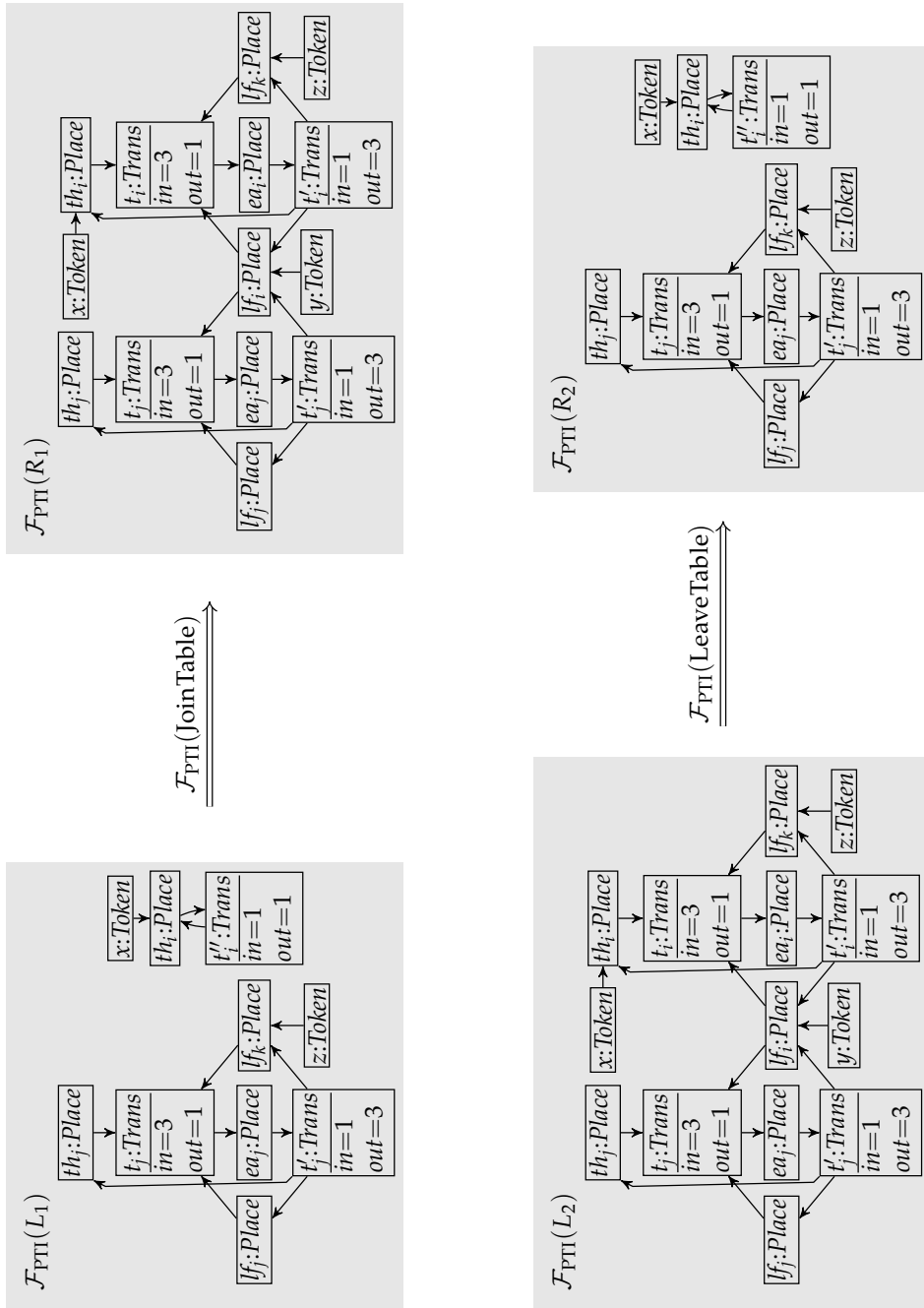


Figure 71: Translated transformation rules without nested application conditions of the Mobile Dining Philosophers system

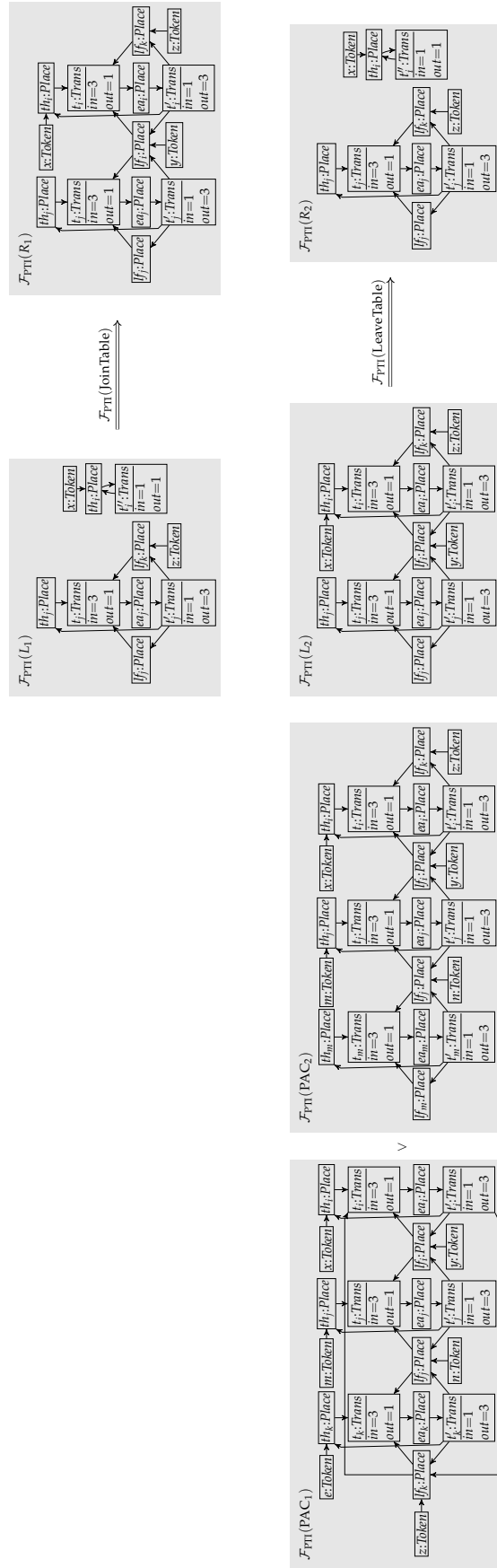


Figure 72: Translated transformation rules with PACs of the extended Mobile Dining Philosophers system

9.2 CREATION OF PTI NET TRANSFORMATIONS

In this section, we are going to verify sufficient properties, which allow for application of the creation part of Theorem 2 and Remark 10 to the restricted \mathcal{M} -functor \mathcal{F}_{PTI} introduced in Definition 64. As the first step we show that \mathcal{F}_{PTI} creates injective PTI net morphisms. As the second step we verify that \mathcal{F}_{PTI} preserves initial pushouts over injective morphisms. Finally, we conclude this section with our next important result concerning the PTI net application stating that \mathcal{F}_{PTI} creates applicability of PTI net rules without or with nested application conditions, (direct) PTI net transformations as well as parallel and sequential independence of PTI net transformations.

As the first step, we show in this section that \mathcal{F}_{PTI} creates injective morphisms. The proof for the uniqueness of morphism creation is based on the additional technical property introduced and shown in the following lemma stating that each typed attributed graph morphism $f' : \mathcal{F}_{\text{PTI}}(\text{NI}_1) \rightarrow \mathcal{F}_{\text{PTI}}(\text{NI}_2)$ is uniquely determined by its V_G -component $f'_{V_G} : V_{1G} \rightarrow V_{2G}$.

Lemma 44 (Uniquely Determined \mathcal{F}_{PTI} -Images).

Consider \mathcal{M} -adhesive categories $(\text{PTINet}, \mathcal{M}_1)$, $(\text{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \text{PTINet}|_{\mathcal{M}_1} \rightarrow \text{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64, typed attributed graphs $\mathcal{F}_{\text{PTI}}(\text{NI}_1)$, $\mathcal{F}_{\text{PTI}}(\text{NI}_2)$, and a morphism $f' : \mathcal{F}_{\text{PTI}}(\text{NI}_1) \rightarrow \mathcal{F}_{\text{PTI}}(\text{NI}_2)$ with $f'_{V_D} = \text{id}_N$. Then we have that f' is uniquely determined by the V_G -component $f'_{V_G} : V_{1G} \rightarrow V_{2G}$ with $V_{iG} = P_i \uplus T_i \uplus I_i$ for $i \in \{1, 2\}$.

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on page 370. □

In the next lemma we show that the restricted \mathcal{M} -functor \mathcal{F}_{PTI} creates injective morphisms, i.e., \mathcal{F}_{PTI} creates injective morphisms uniquely and created morphisms are well-defined.

Lemma 45 (\mathcal{F}_{PTI} Creates Injective Morphisms [207]).

Consider \mathcal{M} -adhesive categories $(\text{PTINet}, \mathcal{M}_1)$, $(\text{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, typed attributed graphs $\mathcal{F}_{\text{PTI}}(\text{NI}_1)$, $\mathcal{F}_{\text{PTI}}(\text{NI}_2)$, and an injective typed attributed graph morphism $f' : \mathcal{F}_{\text{PTI}}(\text{NI}_1) \rightarrow \mathcal{F}_{\text{PTI}}(\text{NI}_2)$ that is compatible with typing morphisms. Then the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \text{PTINet}|_{\mathcal{M}_1} \rightarrow \text{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ creates a unique injective PTI net morphism $f : \text{NI}_1 \rightarrow \text{NI}_2$ such that $\mathcal{F}_{\text{PTI}}(f) = f'$ or formally written:

$$\exists! f : \text{NI}_1 \rightarrow \text{NI}_2 \text{ in } \mathcal{M}_1. \mathcal{F}_{\text{PTI}}(f) = f'.$$

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on page 372. □

To be able to show later on that \mathcal{F}_{PTI} preserves initial pushouts, we have to construct two special boundary objects $\mathcal{F}_{\text{PTI}}(B)$ and B' in the category of typed attributed graphs. $\mathcal{F}_{\text{PTI}}(B)$ is an \mathcal{F}_{PTI} -translation of a boundary object in the category of PTI nets, while B' is a boundary object in the category of typed attributed graphs over the PTI net specific type graph PNTG constructed similar to Fact 6. In the following Lemmas 46 - 48, we introduce the mentioned boundary object constructions considering in Lemma 46 a boundary object in $(\text{PTINet}, \mathcal{M}_1)$ translated by \mathcal{F}_{PTI} , in Lemma 47 a boundary object in

(**AGraphs**_{PNTG}, \mathcal{M}_2) constructed over an injective typed attributed graph morphism, and in Lemma 48 a boundary object in (**AGraphs**_{PNTG}, \mathcal{M}_2) constructed over an \mathcal{F}_{PTI} -image of an injective PTI net morphism.

Lemma 46 (*Application of \mathcal{F}_{PTI} to a PTI Net Boundary Object over an Injective Morphism $f : L \rightarrow G$*).

Consider \mathcal{M} -adhesive categories (**PTINet**, \mathcal{M}_1), (**AGraphs**_{PNTG}, \mathcal{M}_2), a boundary object $B = (P^B, T^B, \text{pre}^B, \text{post}^B, I^B, m^B)$ over an injective morphism $f : L \rightarrow G$ in (**PTINet**, \mathcal{M}_1) constructed according to Remark 9, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow$

AGraphs_{PNTG} $|_{\mathcal{M}_2}$ from Definition 64. Then the application of \mathcal{F}_{PTI} to the boundary object B results in the following typed attributed graph:

$$\begin{aligned} \mathcal{F}_{\text{PTI}}(B) &= ((B_0, \text{NAT}), \text{type}^{\mathcal{F}_{\text{PTI}}(B)}) \text{ with} \\ B_0 &= (V_G^{B_0}, V_D^{B_0} = \mathbb{N}, E_G^{B_0}, E_{\text{NA}}^{B_0}, E_{\text{EA}}^{B_0}, (s_j^{B_0}, t_j^{B_0})_{j \in \{G, \text{NA}, \text{EA}\}}) \text{ where} \\ V_G^{B_0} &= P^B \uplus T^B \uplus I^B = \text{DP}_T \cup \text{DP}_I \text{ with} \\ \text{DP}_T &= \{p \in P^L \mid \exists t \in T^G \setminus f_T(T^L). f_P(p) \in (\bullet t \cup t \bullet)\}, \\ \text{DP}_I &= \{p \in P^L \mid \exists x \in I^G \setminus f_I(I^L). f_P(p) = m^G(x)\}, \\ E_G^{B_0} &= E_{\text{to}2p}^{B_0} \uplus E_{\text{t}2p}^{B_0} \uplus E_{\text{p}2t}^{B_0} = \emptyset \text{ since} \\ E_{\text{to}2p}^{B_0} &= \{(x, p) \in I^B \times P^B \mid m^B(x) = p\} = \emptyset \text{ using } I^B = \emptyset, \\ E_{\text{t}2p}^{B_0} &= \emptyset \text{ using } T^B = \emptyset, \\ E_{\text{p}2t}^{B_0} &= \emptyset \text{ using } T^B = \emptyset, \\ E_{\text{NA}}^{B_0} &= E_{\text{in}}^{B_0} \uplus E_{\text{out}}^{B_0} = \emptyset \text{ using } T^B = \emptyset, \\ E_{\text{EA}}^{B_0} &= E_{\text{wpre}}^{B_0} \uplus E_{\text{wpost}}^{B_0} = \emptyset \text{ since} \\ E_{\text{wpre}}^{B_0} &= \emptyset \text{ using } E_{\text{p}2t}^{B_0} = \emptyset, \\ E_{\text{wpost}}^{B_0} &= \emptyset \text{ using } E_{\text{t}2p}^{B_0} = \emptyset, \end{aligned}$$

and $\mathcal{F}_{\text{PTI}}(b) : \mathcal{F}_{\text{PTI}}(B) \rightarrow \mathcal{F}_{\text{PTI}}(L)$ is an inclusion.

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on page 376. □

Lemma 47 (*Boundary Object in (**AGraphs**_{PNTG}, \mathcal{M}_2) over an Injective Morphism $f' : L' \rightarrow G'$*).

Consider an injective typed attributed graph morphism $f' : L' \rightarrow G'$. Then the boundary object B' is given by $B' = ((B'_0, \text{NAT}), \text{type}^{B'})$ with the boundary points B'_0 that correspond to the dangling points¹ and are defined as follows:

$$\begin{aligned} B'_0 &= (V_G^{B'_0}, V_D^{B'_0} = \mathbb{N}, E_G^{B'_0}, E_{\text{NA}}^{B'_0}, E_{\text{EA}}^{B'_0}, (s_j^{B'_0}, t_j^{B'_0})_{j \in \{G, \text{NA}, \text{EA}\}}) \text{ given by} \\ B'_0 &= \bigcap \{B'' \subseteq L' \mid V_D^{L'} = V_D^{B''} \wedge V_G^{B''} \subseteq V_G^{B''} \wedge E_G^{B''} \subseteq E_G^{B''} \wedge E_{\text{NA}}^{B''} \subseteq E_{\text{NA}}^{B''} \wedge E_{\text{EA}}^{B''} \subseteq E_{\text{EA}}^{B''}\}. \end{aligned}$$

¹ We do not insist on the additional injectivity conditions for $V_G^{B''}$ and $E_G^{B''}$ since we already concern injective morphisms.

The components of $B_0'' = (V_G^{B_0''}, V_D^{B_0''} = \mathbb{N}, E_G^{B_0''}, E_{NA}^{B_0''}, E_{EA}^{B_0''}, (s_j^{B_0''}, t_j^{B_0''})_{j \in \{G, NA, EA\}})$ are defined as given below:

$$E_{NA}^{B_0''} = E_{EA}^{B_0''} = \emptyset,$$

$$V_G^{B_0''} = \{a \in V_G^{L'} = P^L \uplus T^L \uplus I^L \mid$$

$$[\exists a' \in E_{NA}^{G'} \setminus f'_{E_{NA}}(E_{NA}^{L'}) = (E_{in}^{G'} \uplus E_{out}^{G'}) \setminus f'_{E_{NA}}(E_{in}^{L'} \uplus E_{out}^{L'}) \cdot f'_{V_G}(a) = s_{NA}^{G'}(a')]$$

$$\vee [\exists a' \in E_G^{G'} \setminus f'_{E_G}(E_G^{L'}) = (E_{to2p}^{G'} \uplus E_{p2t}^{G'} \uplus E_{t2p}^{G'}) \setminus f'_{E_G}(E_{to2p}^{L'} \uplus E_{p2t}^{L'} \uplus E_{t2p}^{L'}) \cdot$$

$$f'_{V_G}(a) = s_G^{G'}(a') \vee f'_{V_G}(a) = t_G^{G'}(a')],$$

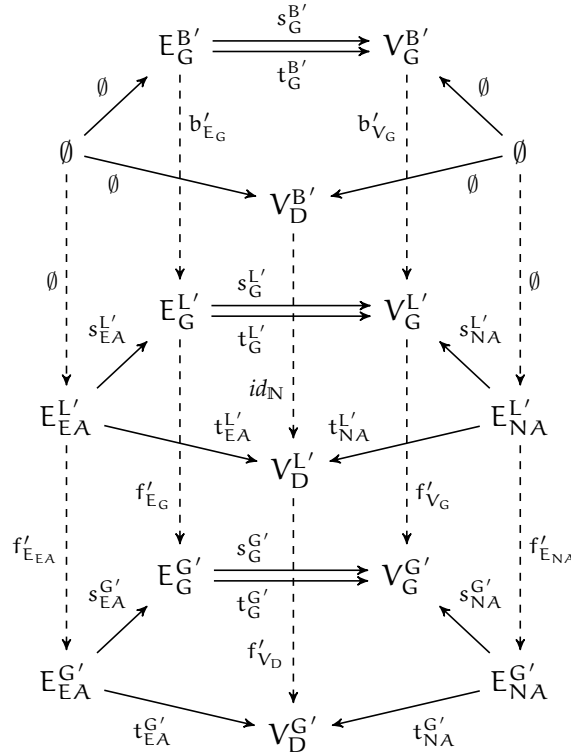
$$E_G^{B_0''} = \{a \in E_G^{L'} = E_{to2p}^{L'} \uplus E_{t2p}^{L'} \uplus E_{p2t}^{L'} \mid$$

$$\exists a' \in E_{EA}^{G'} \setminus f'_{E_{EA}}(E_{EA}^{L'}) = (E_{wpre}^{G'} \uplus E_{wpost}^{G'}) \setminus f'_{E_{EA}}(E_{wpre}^{L'} \uplus E_{wpost}^{L'}) \cdot$$

$$f'_{E_G}(a) = s_{EA}^{G'}(a')],$$

$$s_G^{B_0''}, t_G^{B_0''} : E_G^{B_0''} \rightarrow V_G^{B_0''} \text{ are restrictions of } s_G^{L'}, t_G^{L'} : E_G^{L'} \rightarrow V_G^{L'},$$

and $b' : B' \rightarrow L'$ is an inclusion.



Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 376](#). □

Lemma 48 (Boundary Object in $(\mathbf{AGraphs}_{\text{PTI}}, \mathcal{M}_2)$ over a Morphism $f' = \mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ for an Injective PTI Net Morphism $f : L \rightarrow G$).

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{PTI}}, \mathcal{M}_2)$, an injective PTI net morphism $f : L \rightarrow G$, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PTI}}|_{\mathcal{M}_2}$ from

Definition 64. Then the boundary object B' of the initial pushout over a translated morphism $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ is constructed in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ as follows:

$B' = ((B'_0, \text{NAT}), \text{type}^{B'})$ is essentially given by the boundary points

$$B'_0 = (V_G^{B'_0}, V_D^{B'_0} = \mathbb{N}, E_G^{B'_0}, E_{\text{NA}}^{B'_0}, E_{\text{EA}}^{B'_0}, (s_j^{B'_0}, t_j^{B'_0})_{j \in \{G, \text{NA}, \text{EA}\}}) \text{ with}$$

$$E_G^{B'_0} = E_{\text{NA}}^{B'_0} = E_{\text{EA}}^{B'_0} = \emptyset,$$

$$V_G^{B'_0} = \{a \in V_G^{L'} = P^L \uplus T^L \uplus I^L \mid$$

$$[\exists a' \in E_{\text{NA}}^{G'} \setminus f'_{\text{EA}}(E_{\text{NA}}^{L'}) = (E_{\text{in}}^{G'} \uplus E_{\text{out}}^{G'}) \setminus f'_{\text{EA}}(E_{\text{in}}^{L'} \uplus E_{\text{out}}^{L'})]$$

$$f'_{V_G}(a) = s_{\text{NA}}^{G'}(a')]$$

$$\vee [\exists a' \in E_G^{G'} \setminus f'_{E_G}(E_G^{L'}) = (E_{\text{to2p}}^{G'} \uplus E_{\text{p2t}}^{G'} \uplus E_{\text{t2p}}^{G'}) \setminus f'_{E_G}(E_{\text{to2p}}^{L'} \uplus E_{\text{p2t}}^{L'} \uplus E_{\text{t2p}}^{L'})]$$

$$f'_{V_G}(a) = s_G^{G'}(a') \vee f'_{V_G}(a) = t_G^{G'}(a')]$$

and $b'' : B' \rightarrow \mathcal{F}_{\text{PTI}}(L)$ is an inclusion.

$$\begin{array}{ccc} B' & \xrightarrow{b''} & \mathcal{F}_{\text{PTI}}(L) = L' \\ & & \downarrow \mathcal{F}_{\text{PTI}}(f) = f' \\ & & \mathcal{F}_{\text{PTI}}(G) = G' \end{array}$$

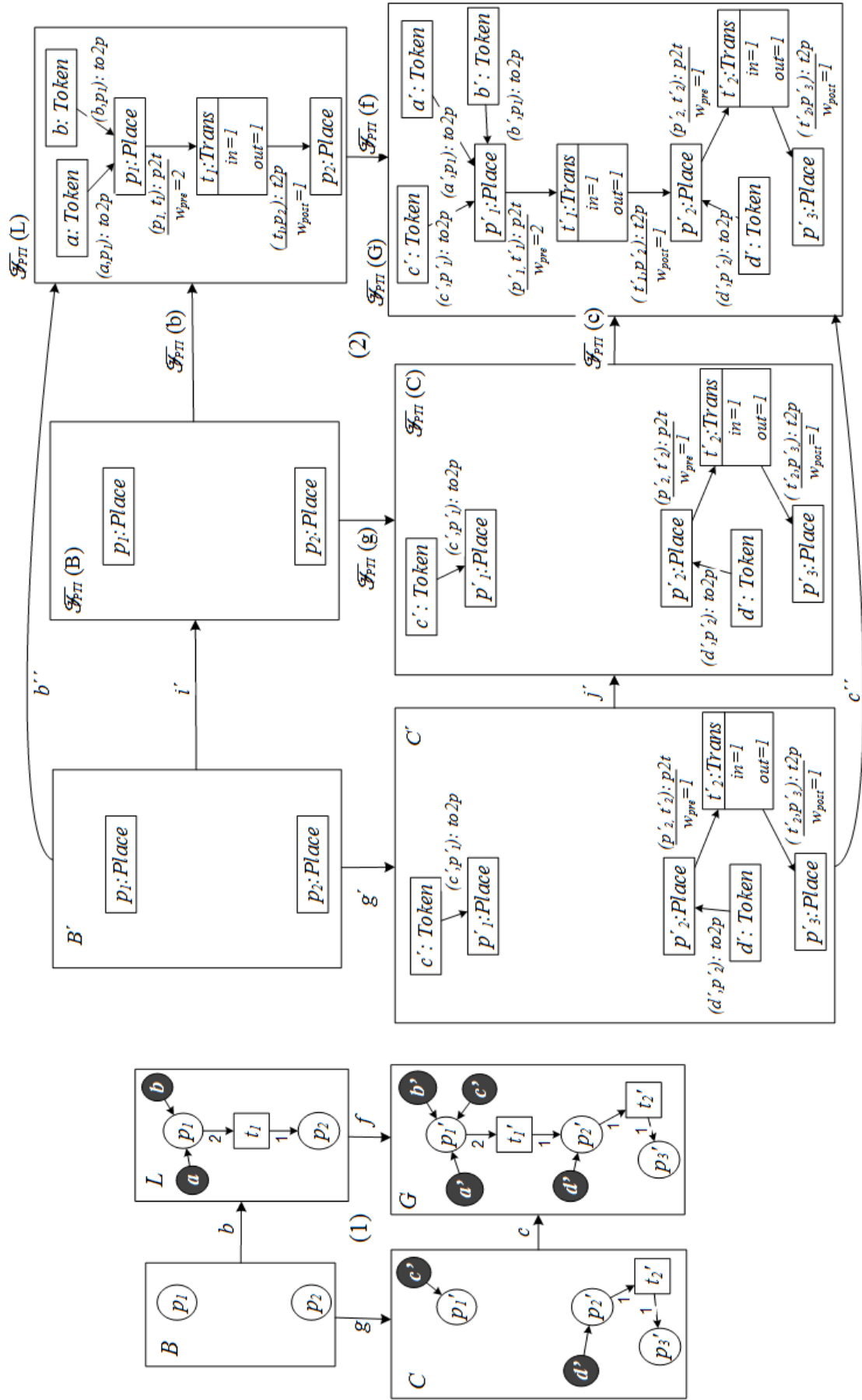
Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 377](#). □

Note that we do not provide alternative constructions for the context object C' in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, because such constructions would not be simplified by assuming that a morphism $f' : L' \rightarrow G'$ is injective or is an \mathcal{F}_{PTI} -image.

In [Figure 73](#) we can see, on the one hand, an example for the application of \mathcal{F}_{PTI} to the PTI net boundary object B according to [Lemma 46](#) given by $V_G^{B_0} = \{p_1, p_2\}$ and, on the other hand, an example for the construction of the boundary object B' in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ over a morphism $f' = \mathcal{F}_{\text{PTI}}(f)$ for an injective PTI net morphism $f : L \rightarrow G$ according to [Lemma 48](#) given by $V_G^{B'_0} = \{p_1, p_2\}$. [Figure 73](#) shows furthermore an example for the construction of the context object C' in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ according to [Fact 7](#) (concerning the boundary object B' constructed according to [Lemma 48](#)) with $V_G^{C'_0} = \{p'_1, p'_2, p'_3, t'_2, c', d'\}$, $E_{\text{NA}}^{C'_0} = \{(t'_2, 1, \text{in}), (t'_2, 1, \text{out})\}$, $E_{\text{EA}}^{C'_0} = \{(p'_2, t'_2, 1), (t'_2, p'_3, 1)\}$, $E_G^{C'_0} = \{(c', p'_1), (d', p'_2), (p'_2, t'_2), (t'_2, p'_3)\}$, and the corresponding source and target functions.

As the next step we need to ensure that the diagram consisting of the boundary object B' constructed according to [Lemma 48](#), the context object C' constructed according to [Fact 7](#), and the objects $\mathcal{F}_{\text{PTI}}(L)$, $\mathcal{F}_{\text{PTI}}(G)$ of the translated injective morphism $f' = \mathcal{F}_{\text{PTI}}(f)$ is in fact an initial pushout in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$.

Figure 73: Preservation of initial pushouts in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$

Lemma 49 (Initial Pushout in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ over a Morphism $f' = \mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ for an Injective PTI Net Morphism $f : L \rightarrow G$).

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, an injective PTI net morphism $f : L \rightarrow G$, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then the diagram (1) given below with the boundary object B' constructed according to Lemma 48, the context object C' constructed according to Fact 7, inclusions $b' : B' \rightarrow L'$, $c' : C' \rightarrow G'$, and the morphism $g' : B' \rightarrow C'$ given by $g'_j(x) = (f'_j \circ b'_j)(x)$ for $j \in \{V_G, V_D, E_G, E_{NA}, E_{EA}\}$ is an initial pushout in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ over $f' = \mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$.

$$\begin{array}{ccc} B' & \xrightarrow{b'} & L' \\ g' \downarrow & (1) & \downarrow f' = \mathcal{F}_{\text{PTI}}(f) \\ C' & \xrightarrow{c'} & G' \end{array}$$

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 378](#). \square

After the sufficient boundary object constructions are introduced in Lemmas 46, 48 and the construction of initial pushouts over injective morphisms that are \mathcal{F}_{PTI} -images is discussed in Lemma 49, we can verify that \mathcal{F}_{PTI} preserves initial pushouts over injective morphisms.

Lemma 50 (\mathcal{F}_{PTI} Preserves Initial Pushouts over Injective Morphisms [207]).

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64, and let (1) be an initial pushout over an injective morphism $f : L \rightarrow G$ in $(\mathbf{PTINet}, \mathcal{M}_1)$. Then (2) is an initial pushout over an injective morphism $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$.

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ \downarrow & (1) & \downarrow f \\ C & \longrightarrow & G \end{array} \quad \Rightarrow \quad \begin{array}{ccc} \mathcal{F}_{\text{PTI}}(B) & \xrightarrow{\mathcal{F}_{\text{PTI}}(b)} & \mathcal{F}_{\text{PTI}}(L) \\ \downarrow & (2) & \downarrow \mathcal{F}_{\text{PTI}}(f) \\ \mathcal{F}_{\text{PTI}}(C) & \longrightarrow & \mathcal{F}_{\text{PTI}}(G) \end{array}$$

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 379](#). \square

An example for the preservation of initial pushouts by \mathcal{F}_{PTI} is given in [Figure 73](#) where (1) is an initial pushout over an injective morphism $f : L \rightarrow G$ in $(\mathbf{PTINet}, \mathcal{M}_1)$, (2) is the induced pushout over the injective morphism $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$, and the initial pushout over $\mathcal{F}_{\text{PTI}}(f)$ in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ is given by the outer diagram with corners B' , C' , $\mathcal{F}_{\text{PTI}}(L)$, and $\mathcal{F}_{\text{PTI}}(G)$. Since $i' : B' \rightarrow \mathcal{F}_{\text{PTI}}(B)$ and $j' : C' \rightarrow \mathcal{F}_{\text{PTI}}(C)$ are isomorphisms, the diagram (2) is an initial pushout over the injective morphism $\mathcal{F}_{\text{PTI}}(f)$.

After all of the sufficient properties for the creation of rule applicability, (direct) PTI net transformations as well as parallel and sequential independence of PTI net transformations are shown to hold for the restricted \mathcal{M} -functor \mathcal{F}_{PTI} , we now formulate and

prove one of our next important theorems for the PTI net application. This theorem states intuitively that transformation steps, which are possible in a transformation system translated by \mathcal{F}_{PTI} , are also possible in the original PTI net transformation system. Note that the following theorem is applicable to PTI net transformation systems containing rules without or with nested application conditions.

Theorem 22 (Creation of PTI Net Transformations from Typed Attributed Graph Transformations [207]).

Consider \mathcal{M} -adhesive transformation systems $(\mathbf{PTINet}, \mathcal{M}_1, P)$, $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$. The restricted functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ given in Definition 64 creates applicability of PTI net rules, (direct) PTI net transformations as well as parallel and sequential independence of PTI net transformations.

Proof.

PTI net transformations are (direct) transformations with injective match morphisms only. In order to use the results from Theorems 2 and 7 for the case of transformations without nested application conditions as well as Remark 12 and Theorem 10 for the case of transformations with nested application conditions, we have to show that \mathcal{F}_{PTI} is a restricted \mathcal{M} -functor according to Definition 44, \mathcal{F}_{PTI} creates injective morphisms, and \mathcal{F}_{PTI} preserves initial pushouts over injective morphisms. Furthermore, it is required that the category $(\mathbf{PTINet}, \mathcal{M}_1)$ has initial pushouts, which is satisfied according to Fact 11 in Subsection 2.4.3. In fact, we have that $\mathcal{F}_{\text{PTI}}(\mathcal{M}_1) \subseteq \mathcal{M}_2$, i.e., \mathcal{F}_{PTI} preserves injectivity of morphisms by Lemma 42 and \mathcal{F}_{PTI} preserves pushouts of injective morphisms according to Lemma 43. Moreover, the creation of injective morphisms by \mathcal{F}_{PTI} is shown in Lemma 45 and the preservation of initial pushouts over injective morphisms by \mathcal{F}_{PTI} is shown in Lemma 50. \square

For an example of the creation of PTI net transformation steps consider again Figure 69 and Figure 70 from the last section. For the translated transformation step given in Figure 70, the result of the creation is the PTI net transformation step depicted in Figure 69.

9.3 \mathcal{F}_{PTI} -TRANSFER OF BISIMILARITY FOR PTI NET TRANSFORMATION SYSTEMS

Similarly to Section 6.3, we instantiate in this section our general theoretical results on bisimulation between \mathcal{M} -adhesive transformation systems from Section 3.2 to our second concrete application considering PTI net transformation systems. Basically, we ensure that the adapted requirements from Theorems 4 and 5 are satisfied by the restricted \mathcal{M} -functor \mathcal{F}_{PTI} and the categories of PTI nets and typed attributed graphs. The proof of the theorem depends on the various lemmas introduced in earlier sections.

Theorem 23 (\mathcal{F}_{PTI} -Bisimilarity and \mathcal{F}_{PTI} -Transfer of R-Bisimilarity).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{PTINet}, \mathcal{M}_1, P_1)$, $AS_2 = (\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P_1))$, $AS_3 = (\mathbf{PTINet}, \mathcal{M}_1, P_2)$, $AS_4 = (\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P_2))$, a rule relation $R \subseteq P_1 \times P_2$, and the restricted functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ introduced in Definition 64. Then the following two statements hold:

- Let G be a PTI net from AS_1 . Then G is \mathcal{F}_{PTI} -bisimilar to the corresponding typed attributed graph $\mathcal{F}_{\text{PTI}}(G)$ in AS_2 , written $G \sim_{\mathcal{F}_{\text{PTI}}} \mathcal{F}_{\text{PTI}}(G)$.

- Let G be a PTI net from AS_1 and G' be a PTI net from AS_3 . Then G is R -bisimilar to G' iff $\mathcal{F}_{PTI}(G)$ is $\mathcal{F}_{PTI}(R)$ -bisimilar to $\mathcal{F}_{PTI}(G')$, written $(G \sim_R G') \Leftrightarrow (\mathcal{F}_{PTI}(G) \sim_{\mathcal{F}_{PTI}(R)} \mathcal{F}_{PTI}(G'))$.

Proof.

PTI net transformations are (direct) transformations with injective match morphisms only. In order to use the results from Theorems 4 and 5 for the case of transformation steps without or with nested application conditions, we have to show that \mathcal{F}_{PTI} is a restricted \mathcal{M} -functor according to Definition 44, \mathcal{F}_{PTI} creates injective morphisms, and \mathcal{F}_{PTI} preserves initial pushouts over injective morphisms. Furthermore, it is required that the category $(\mathbf{PTINet}, \mathcal{M}_1)$ has initial pushouts, which is satisfied according to Fact 11 in Subsection 2.4.3. In fact, we have that $\mathcal{F}_{PTI}(\mathcal{M}_1) \subseteq \mathcal{M}_2$, i.e., \mathcal{F}_{PTI} preserves injectivity of morphisms by Lemma 42 and \mathcal{F}_{PTI} preserves pushouts of injective morphisms according to Lemma 43. Moreover, the creation of injective morphisms by \mathcal{F}_{PTI} is shown in Lemma 45 and the preservation of initial pushouts over injective morphisms by \mathcal{F}_{PTI} is shown in Lemma 50. \square

For our running example on the Mobile Dining Philosophers problem, it is obvious that for each PTI net derivable by rule application in the original PTI net transformation system, there is an \mathcal{F}_{PTI} -bisimilar typed attributed graph in the corresponding typed attributed graph transformation system implying altogether the behavioral equivalence of the \mathcal{F}_{PTI} -related parts of the involved source and target transformation systems.

Note that we do not consider in this section a concrete example for \mathcal{F}_{PTI} -transfer of R -bisimilarity but refer the reader to the corresponding example for hypergraph transformation systems in Example 15.

CONFLUENCE ANALYSIS OF PTI NET TRANSFORMATION SYSTEMS

In this chapter, we consider similarly to the case of hypergraphs how we can apply theoretical results from Sections 4.1 and 4.2 to our concrete restricted \mathcal{M} -functor \mathcal{F}_{PTI} between categories of PTI nets and typed attributed graphs to obtain local confluence analysis results for PTI net transformation systems. In Section 10.1 we focus on local confluence analysis of PTI net transformation systems without nested application conditions, while in Section 10.2 we consider local confluence analysis of PTI net transformation systems containing rules with nested application conditions. Afterwards in Section 10.3, we verify local confluence of a PTI net transformation system introduced in Example 12 using the AGG-tool and our theoretical results from Sections 10.1 and 10.2. Finally, in Section 10.4 we consider termination, confluence, and functional behavior analysis for PTI net transformation systems without or with nested application conditions.

10.1 LOCAL CONFLUENCE OF PTI NET TRANSFORMATION SYSTEMS WITHOUT NESTED APPLICATION CONDITIONS

In this section, we consider in detail how we can analyze local confluence of PTI net transformation systems using our framework of restricted \mathcal{M} -functors and the corresponding theoretical results concerning the functorial transfer of local confluence for transformations without nested application conditions introduced in Section 4.1.

In the following we formulate and prove several technical basics and results, which we need to be able to prove the main applicational result of this section concerning the local confluence analysis of PTI net transformation systems. The most important technical requirement, which we need for this reason, is given in Lemma 54 where we show that the restricted \mathcal{M} -functor \mathcal{F}_{PTI} is compatible with pair factorization. According to Definition 51, this means that the source and the target categories of \mathcal{F}_{PTI} have $\mathcal{E}' - \mathcal{M}$ pair factorizations and \mathcal{F}_{PTI} translates an $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization in **PTINet** into an $\mathcal{E}'_2 - \mathcal{M}_2$ pair factorization in **AGraphs_{PNTG}**.

Since for the source and the target categories of \mathcal{F}_{PTI} holds that $\mathcal{M}_i = \mathcal{M}'_i$ for $i \in \{1, 2\}$, we have similarly to the case of hypergraphs that the $\mathcal{M} - \mathcal{M}'$ pushout-pullback decomposition property introduced in Definition 72 (see Appendix A) and needed for the completeness of critical pairs (see Fact 2) is fulfilled and we do not need to assume the satisfaction of this property in the subsequent results. Furthermore, we use here also the notion of $\mathcal{E}' - \mathcal{M}$ pair factorization instead of the common $\mathcal{E}' - \mathcal{M}'$ pair factorization as already described in Remark 1.

First we want to show that the source and the target categories of the restricted \mathcal{M} -functor \mathcal{F}_{PTI} have the corresponding $\mathcal{E}' - \mathcal{M}$ pair factorizations. As we already know from Remark 5.26 in [88, p. 122], there is a general construction for an $\mathcal{E}' - \mathcal{M}'$ pair factorization based on coproducts and $\mathcal{E} - \mathcal{M}$ -factorization, which can be applied to PTI nets and typed attributed graphs. For the considered case this requires that the categories **PTINet**

and $\mathbf{AGraphs}_{\text{PNTG}}$ have coproducts and $\mathcal{E} - \mathcal{M}$ -factorizations. Since the empty PTI net \emptyset resp. almost empty typed attributed graph $\mathcal{F}_{\text{PTI}}(\emptyset)$ (see more details in Section 8.1) are initial in the categories \mathbf{PTINet} resp. $\mathbf{AGraphs}_{\text{PNTG}}$ and we have pushouts in both categories, we also have coproducts in both categories, which are constructed component-wise as disjoint union. Furthermore, according to [88], the category $\mathbf{AGraphs}_{\text{ATG}}$ and hence also $\mathbf{AGraphs}_{\text{PNTG}}$ has $\mathcal{E}_2 - \mathcal{M}_2$ -factorization where \mathcal{M}_2 is the class of all injective typed attributed graph morphisms and \mathcal{E}_2 is the class of all surjective typed attributed graph morphisms. Note that for typed attributed graphs all morphisms are identical in the V_D -component and the data type \mathbb{N} . The corresponding $\mathcal{E}_1 - \mathcal{M}_1$ -factorization for PTI nets can be constructed as given in the following lemma.

Lemma 51 ($\mathcal{E} - \mathcal{M}$ -Factorization in \mathbf{PTINet}).

The \mathcal{M} -adhesive category $(\mathbf{PTINet}, \mathcal{M}_1)$ has an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization according to Definition 10 where \mathcal{M}_1 is the class of all injective PTI net morphisms and \mathcal{E}_1 is the class of all surjective PTI net morphisms.

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 380](#). \square

Similar to Section 7.1, we have to show in the next step that \mathcal{F}_{PTI} preserves coproducts and $\mathcal{E} - \mathcal{M}$ -factorizations. The first of these two properties is shown to hold in the lemma below.

Lemma 52 (\mathcal{F}_{PTI} Preserves Coproducts).

Consider a PTI net A , a family of PTI nets $(A_j)_{j \in I}$, a family of PTI net morphisms $(i_j : A_j \rightarrow A)_{j \in I}$, a coproduct $(A, (i_j)_{j \in I})$ of $(A_j)_{j \in I}$ in \mathbf{PTINet} , and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then $(\mathcal{F}_{\text{PTI}}(A), (\mathcal{F}_{\text{PTI}}(i_j))_{j \in I})$ is a coproduct of $(\mathcal{F}_{\text{PTI}}(A_j))_{j \in I}$ in $\mathbf{AGraphs}_{\text{PNTG}}$.

$$\begin{array}{ccc}
 A_j & \xrightarrow{i_j} & A \\
 & \searrow f_j & \downarrow f \\
 & & B
 \end{array}
 \xRightarrow{\mathcal{F}_{\text{PTI}}}
 \begin{array}{ccc}
 \mathcal{F}_{\text{PTI}}(A_j) & \xrightarrow{\mathcal{F}_{\text{PTI}}(i_j)} & \mathcal{F}_{\text{PTI}}(A) \\
 & \searrow \mathcal{F}_{\text{PTI}}(f_j) & \downarrow \mathcal{F}_{\text{PTI}}(f) \\
 & & \mathcal{F}_{\text{PTI}}(B)
 \end{array}$$

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 383](#). \square

For the case of PTI net transformations, we are interested only in the preservation of binary coproducts as given in [Figure 74](#) to be able to construct the corresponding $\mathcal{E}' - \mathcal{M}$ pair factorizations for the source and the target categories of the restricted \mathcal{M} -functor \mathcal{F}_{PTI} .

The second property, namely the preservation of $\mathcal{E} - \mathcal{M}$ -factorizations by \mathcal{F}_{PTI} , is fulfilled if \mathcal{F}_{PTI} preserves injective and surjective morphisms. Since \mathcal{F}_{PTI} is a restricted \mathcal{M} -functor, we already have that \mathcal{F}_{PTI} preserves injective morphisms by Lemma 42. It remains to show that \mathcal{F}_{PTI} preserves surjective morphisms.

Lemma 53 (\mathcal{F}_{PTI} Preserves Surjective Morphisms).

Consider two PTI nets $\text{NI}_1 = (P_1, T_1, \text{pre}_1, \text{post}_1, I_1, m_1)$, $\text{NI}_2 = (P_2, T_2, \text{pre}_2, \text{post}_2, I_2, m_2)$, a surjective PTI net morphism $f : \text{NI}_1 \rightarrow \text{NI}_2$ with $f = (f_P : P_1 \rightarrow P_2, f_T : T_1 \rightarrow T_2, f_I : I_1 \rightarrow I_2)$, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$. Then

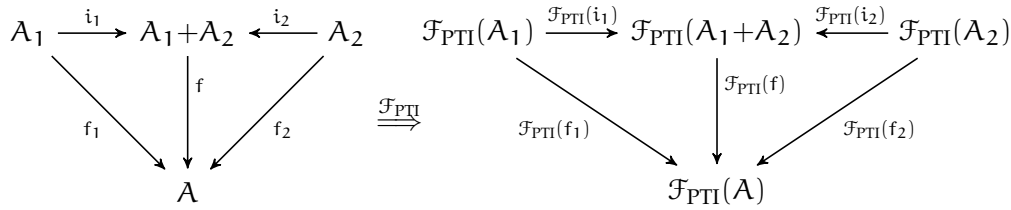


Figure 74: Preservation of binary coproducts by the restricted \mathcal{M} -functor \mathcal{F}_{PTI}

the corresponding typed attributed graph morphism $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(\text{NI}_1) \rightarrow \mathcal{F}_{\text{PTI}}(\text{NI}_2)$ with $\mathcal{F}_{\text{PTI}}(f) = f' = (f'_{V_G} : V_{1G} \rightarrow V_{2G}, f'_{V_D} : \mathbb{N} \rightarrow \mathbb{N}, f'_{E_G} : E_{1G} \rightarrow E_{2G}, f'_{E_{NA}} : E_{1NA} \rightarrow E_{2NA}, f'_{E_{EA}} : E_{1EA} \rightarrow E_{2EA})$ is also surjective.

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 383](#). \square

The following lemma summarizes the results from the previous three lemmas and show that the restricted \mathcal{M} -functor \mathcal{F}_{PTI} is compatible with pair factorization. This property is required to obtain local confluence of PTI net transformation systems based on \mathcal{F}_{PTI} -reachable critical pairs.

Lemma 54 (\mathcal{F}_{PTI} is Compatible with Pair Factorization).

Consider a PTI net transformation system $(\text{PTINet}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\text{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \text{PTINet}|_{\mathcal{M}_1} \rightarrow \text{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from [Definition 64](#). Then \mathcal{F}_{PTI} is compatible with pair factorization.

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 385](#). \square

Now we can apply the result concerning the creation of local confluence based on \mathcal{F} -reachable critical pairs from [Theorem 8](#) to the restricted \mathcal{M} -functor \mathcal{F}_{PTI} . This is one of our main applicational results for PTI net transformation systems allowing us to verify local confluence by analyzing the corresponding typed attributed graph transformation systems using the AGG-tool.

Theorem 24 (Local Confluence of PTI Net Transformation Systems without Nested Application Conditions [209]).

Consider a PTI net transformation system $(\text{PTINet}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\text{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \text{PTINet}|_{\mathcal{M}_1} \rightarrow \text{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from [Definition 64](#). Then $(\text{PTINet}, \mathcal{M}_1, P)$ is locally confluent for all transformation spans $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ if all \mathcal{F}_{PTI} -reachable critical pairs of $\mathcal{F}_{\text{PTI}}(\rho_1)$ and $\mathcal{F}_{\text{PTI}}(\rho_2)$ are strictly confluent.

Proof.

In [Lemma 45](#) resp. [Theorem 22](#) we have already shown that \mathcal{F}_{PTI} creates injective morphisms¹ resp. (direct) transformations. Furthermore, according to [Lemma 54](#) \mathcal{F}_{PTI} is compatible with pair factorization, which altogether allows us to apply [Theorem 8](#) with $\mathcal{F}_R = \mathcal{F}_{\text{PTI}}$. \square

¹ Since \mathcal{F}_{PTI} is an \mathcal{M} -functor restricted to injective morphisms only, it suffices to show in this special case the creation of injective morphisms by \mathcal{F}_{PTI} instead of the creation of general morphisms by \mathcal{F}_{PTI} to be able to apply [Theorem 8](#).

10.2 LOCAL CONFLUENCE OF PTI NET TRANSFORMATION SYSTEMS WITH NESTED APPLICATION CONDITIONS

This section considers the verification of local confluence for PTI net transformation systems with nested application conditions. Here we verify sufficient properties like translation and creation of jointly surjective morphisms as well as preservation of pullbacks of injective morphisms allowing us to apply Theorem 11 concerning the creation of local confluence based on \mathcal{F} -reachable critical pairs for rules with nested application conditions to the restricted \mathcal{M} -functor \mathcal{F}_{PTI} from Definition 64.

As the first step we introduce two lemmas, in which we show that the restricted \mathcal{M} -functor \mathcal{F}_{PTI} between the categories of PTI nets and typed attributed graphs preserves and creates \mathcal{E}' -instances or, more precisely for the case of PTI net and typed attributed graph transformations, jointly surjective morphisms. We need these two technical properties to obtain the compatibility of \mathcal{F}_{PTI} with **Shift**-transformation, which is one of the requirements of Theorem 11.

Lemma 55 (\mathcal{F}_{PTI} *Translates Jointly Surjective Morphisms*).

Consider \mathcal{E}'_1 - \mathcal{M}_1 pair factorization in $(\mathbf{PTINet}, \mathcal{M}_1)$ and \mathcal{E}'_2 - \mathcal{M}_2 pair factorization in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$. Then the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64 translates jointly surjective PTI net morphisms (α', β') into the corresponding jointly surjective typed attributed graph morphisms (α'', β'') with $\alpha'' = \mathcal{F}_{\text{PTI}}(\alpha')$ and $\beta'' = \mathcal{F}_{\text{PTI}}(\beta')$.

$$\begin{array}{ccc}
 & P' & \\
 & \downarrow \alpha' & \\
 C & \xrightarrow{\beta'} & C'
 \end{array}
 \qquad
 \begin{array}{ccc}
 & \mathcal{F}_{\text{PTI}}(P') & \\
 & \downarrow \alpha'' = \mathcal{F}_{\text{PTI}}(\alpha') & \\
 \mathcal{F}_{\text{PTI}}(C) & \xrightarrow{\beta'' = \mathcal{F}_{\text{PTI}}(\beta')} & C'' = \mathcal{F}_{\text{PTI}}(C')
 \end{array}$$

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 386](#). □

Lemma 56 (\mathcal{F}_{PTI} *Creates Jointly Surjective Morphisms*).

Consider \mathcal{E}'_1 - \mathcal{M}_1 pair factorization in $(\mathbf{PTINet}, \mathcal{M}_1)$, \mathcal{E}'_2 - \mathcal{M}_2 pair factorization in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, and PTI net morphisms $\alpha : P \rightarrow C$, $\beta : P \rightarrow P'$. Then the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64 creates jointly surjective PTI net morphisms (α', β') from the corresponding typed attributed graph morphisms (α'', β'') in $\bar{\mathcal{E}}'_2 = \mathcal{F}_{\text{PTI}}(\mathcal{E}'_1)$ if for all spans $(C \xleftarrow{\alpha} P \xrightarrow{\beta} P')$ holds that the diagram (1) below commutes, $\mathcal{F}_{\text{PTI}}(\alpha') = \alpha''$, $\mathcal{F}_{\text{PTI}}(\beta') = \beta''$, and the injectivity of β'' implies the injectivity of β' .

$$\begin{array}{ccc}
 P & \xrightarrow{\beta} & P' \\
 \alpha \downarrow & (1) & \downarrow \alpha' \\
 C & \xrightarrow{\beta'} & C'
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathcal{F}_{\text{PTI}}(P) & \xrightarrow{\mathcal{F}_{\text{PTI}}(\beta)} & \mathcal{F}_{\text{PTI}}(P') \\
 \mathcal{F}_{\text{PTI}}(\alpha) \downarrow & (2) & \downarrow \alpha'' = \mathcal{F}_{\text{PTI}}(\alpha') \\
 \mathcal{F}_{\text{PTI}}(C) & \xrightarrow{\beta'' = \mathcal{F}_{\text{PTI}}(\beta')} & C'' = \mathcal{F}_{\text{PTI}}(C')
 \end{array}$$

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 387](#). □

Another technical property, which must hold for the application of our theoretical results to the restricted \mathcal{M} -functor \mathcal{F}_{PTI} , is the preservation of PTI net pullbacks of injective morphisms. This property has to be satisfied to be able to deal with translated derived rules as introduced in Definition 55.

Lemma 57 (\mathcal{F}_{PTI} Preserves Pullbacks of Injective Morphisms).

Consider a PTI net transformation system $(\mathbf{PTINet}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$, the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$, PTI nets NI_i for $i \in \{0, 1, 2, 3\}$ with PTI net morphisms $\mathbf{b} = (b_P, b_T, b_I)$, $\mathbf{c} = (c_P, c_T, c_I)$, $\mathbf{g} = (g_P, g_T, g_I)$, $\mathbf{h} = (h_P, h_T, h_I)$, and typed attributed graphs $\mathcal{F}_{\text{PTI}}(\text{NI}_i)$ for $i \in \{0, 1, 2, 3\}$ with typed attributed graph morphisms $\mathcal{F}_{\text{PTI}}(\mathbf{b}) = \mathbf{b}' = (b'_{V_G}, b'_{V_D}, b'_{E_G}, b'_{E_{NA}}, b'_{E_{EA}})$, $\mathcal{F}_{\text{PTI}}(\mathbf{c}) = \mathbf{c}' = (c'_{V_G}, c'_{V_D}, c'_{E_G}, c'_{E_{NA}}, c'_{E_{EA}})$, $\mathcal{F}_{\text{PTI}}(\mathbf{g}) = \mathbf{g}' = (g'_{V_G}, g'_{V_D}, g'_{E_G}, g'_{E_{NA}}, g'_{E_{EA}})$, $\mathcal{F}_{\text{PTI}}(\mathbf{h}) = \mathbf{h}' = (h'_{V_G}, h'_{V_D}, h'_{E_G}, h'_{E_{NA}}, h'_{E_{EA}})$. If (1) is a pullback in \mathbf{PTINet} with $\mathbf{g}, \mathbf{h} \in \mathcal{M}_1$ then we have that (2) is a pullback in $\mathbf{AGraphs}_{\text{PNTG}}$ with $\mathcal{F}_{\text{PTI}}(\mathbf{g}), \mathcal{F}_{\text{PTI}}(\mathbf{h}) \in \mathcal{M}_2$.

$$\begin{array}{ccc}
 \text{NI}_0 & \xrightarrow{\mathbf{b}} & \text{NI}_1 \\
 \mathbf{c} \downarrow & (1) & \downarrow \mathbf{g} \\
 \text{NI}_2 & \xrightarrow{\mathbf{h}} & \text{NI}_3
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathcal{F}_{\text{PTI}}(\text{NI}_0) & \xrightarrow{\mathcal{F}_{\text{PTI}}(\mathbf{b})=\mathbf{b}'} & \mathcal{F}_{\text{PTI}}(\text{NI}_1) \\
 \mathcal{F}_{\text{PTI}}(\mathbf{c})=\mathbf{c}' \downarrow & (2) & \downarrow \mathcal{F}_{\text{PTI}}(\mathbf{g})=\mathbf{g}' \\
 \mathcal{F}_{\text{PTI}}(\text{NI}_2) & \xrightarrow{\mathcal{F}_{\text{PTI}}(\mathbf{h})=\mathbf{h}'} & \mathcal{F}_{\text{PTI}}(\text{NI}_3)
 \end{array}$$

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on [page 387](#). □

After all needed requirements are shown to be satisfied, we can now apply the result from Theorem 11 to the restricted \mathcal{M} -functor \mathcal{F}_{PTI} . This result extends the corresponding result from the previous section and allows us to analyze local confluence of PTI net transformation systems containing rules with nested application conditions.

Theorem 25 (Local Confluence of PTI Net Transformation Systems with Nested Application Conditions).

Consider a PTI net transformation system $(\mathbf{PTINet}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$ where P are rules with nested application conditions, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then $(\mathbf{PTINet}, \mathcal{M}_1, P)$ is locally confluent for all transformation spans $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ with nested application conditions if all \mathcal{F}_{PTI} -reachable critical pairs with nested application conditions of $\mathcal{F}_{\text{PTI}}(\rho_1)$ and $\mathcal{F}_{\text{PTI}}(\rho_2)$ are strictly $\text{AC}(\mathcal{F}_{\text{PTI}})$ -confluent.

Proof.

In Lemma 45 resp. Theorem 22, we have shown that \mathcal{F}_{PTI} creates injective morphisms² resp. (direct) transformations and rule applicability. According to Lemma 54, we have that \mathcal{F}_{PTI} is compatible with pair factorization. Moreover, by Theorem 21 resp. Lemma 57 it holds that \mathcal{F}_{PTI} translates rule applicability resp. preserves pullbacks of injective morphisms. Finally, in Lemma 55 resp. Lemma 56 we have shown that \mathcal{F}_{PTI} translates resp. creates jointly surjective morphisms. This altogether allows us to apply Theorem 11 with $\mathcal{F}_R = \mathcal{F}_{\text{PTI}}$. □

² Since \mathcal{F}_{PTI} is an \mathcal{M} -functor restricted to injective morphisms only, it suffices to show in this special case the creation of injective morphisms by \mathcal{F}_{PTI} instead of the creation of general morphisms by \mathcal{F}_{PTI} to be able to apply Theorem 11.

In the following example we show how we can construct a concrete \mathcal{F}_{PTI} -reachable critical pair and how we can check whether this \mathcal{F}_{PTI} -reachable critical pair is strictly $\text{AC}(\mathcal{F}_{\text{PTI}})$ -confluent.

Example 18 (Strict $\text{AC}(\mathcal{F}_{\text{PTI}})$ -Confluence of an \mathcal{F}_{PTI} -Reachable Critical Pair in the Context of Mobile Dining Philosophers Scenario).

Consider an \mathcal{F} -reachable weak critical pair $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ for the weak critical pair $P_1 \xleftarrow{\text{LeaveTable}, o_1} K \xrightarrow{\text{LeaveTable}, o_2} P_2$ from Example 13 with $\text{ac}_{\mathcal{F}(K)} = \mathcal{F}(\text{ac}_K)$, $\text{ac}_{\mathcal{F}(K)}^* = \mathcal{F}(\text{ac}_K^*)$, and $\mathcal{F} = \mathcal{F}_{\text{PTI}}$. Then we have the following translated extension and conflict-inducing application conditions:

- $\text{ac}_{\mathcal{F}(K)} = \mathcal{F}(\text{ac}_K) = \mathcal{F}(\text{ac}_{K_1} \wedge \text{ac}_{K_2}) \stackrel{\text{Def. 47}}{=} \mathcal{F}(\text{ac}_{K_1}) \wedge \mathcal{F}(\text{ac}_{K_2})$
- $\text{ac}_{\mathcal{F}(K)}^* = \mathcal{F}(\text{ac}_K^*) = \mathcal{F}(\text{true}) \stackrel{\text{Def. 47}}{=} \text{true}$

Similar to Example 13, $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ can be extended by the embedding morphism $\mathcal{F}(m) : \mathcal{F}(K) \rightarrow \mathcal{F}(G)$ with $\mathcal{F}(m) \models \text{ac}_{\mathcal{F}(K)} \wedge \text{ac}_{\mathcal{F}(K)}^*$ and a translated pair of AC-regarding transformations $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(m_2)} \mathcal{F}(H_2)$. Thus, we have that $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ is an \mathcal{F} -reachable critical pair.

$$\begin{array}{ccccc} \mathcal{F}(P_1) & \xleftarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_1)} & \mathcal{F}(K) & \xrightarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_2)} & \mathcal{F}(P_2) \\ \downarrow & & \downarrow \mathcal{F}(m) & & \downarrow \\ \mathcal{F}(H_1) & \xleftarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(m_1)} & \mathcal{F}(G) & \xrightarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(m_2)} & \mathcal{F}(H_2) \end{array}$$

Now we want to show that $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ is strictly $\text{AC}(\mathcal{F})$ -confluent according to Definition 60. We have the plain strict confluence similarly to Example 13 as given in the diagram below

$$\begin{array}{ccc} & \mathcal{F}(K) & \\ \swarrow \mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_1) & & \searrow \mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_2) \\ \mathcal{F}(P_1) = \mathcal{F}(P_2) & & \mathcal{F}(P_2) = \mathcal{F}(P_1) \\ \swarrow \text{Id}, \mathcal{F}(o'_1) & & \searrow \text{Id}, \mathcal{F}(o'_2) \\ & \mathcal{F}(K') \cong (\mathcal{F}(P_1) = \mathcal{F}(P_2)) & \end{array}$$

and the $\text{AC}(\mathcal{F})$ -compatibility, i.e., $(\text{ac}_{\mathcal{F}(K)} \wedge \text{ac}_{\mathcal{F}(K)}^*) \Rightarrow (\text{ac}(\mathcal{F}(t_1)) \wedge \text{ac}(\mathcal{F}(t_2)))$ since $\text{ac}_{\mathcal{F}(K)} \wedge \text{ac}_{\mathcal{F}(K)}^* = \mathcal{F}(\text{ac}_{K_1}) \wedge \mathcal{F}(\text{ac}_{K_2})$ (as shown above) implies $\text{ac}(\mathcal{F}(t_1)) \wedge \text{ac}(\mathcal{F}(t_2)) = \mathcal{F}(\text{ac}_{K_1}) \wedge \mathcal{F}(\text{ac}_{K_2})$ as shown below.

$$\begin{aligned} \text{ac}(\mathcal{F}(t_1)) &= \text{ac}(\mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_1)} \mathcal{F}(P_1) \xrightarrow{\text{Id}, \mathcal{F}(o'_1)} \mathcal{F}(K')) \\ &= \mathcal{F}(\text{ac}_{K_1}) \wedge \mathcal{F}(\text{true}), \\ \text{ac}(\mathcal{F}(t_2)) &= \text{ac}(\mathcal{F}(K) \xrightarrow{\mathcal{F}(\text{LeaveTable}), \mathcal{F}(o_2)} \mathcal{F}(P_2) \xrightarrow{\text{Id}, \mathcal{F}(o'_2)} \mathcal{F}(K')) \\ &= \mathcal{F}(\text{ac}_{K_2}) \wedge \mathcal{F}(\text{true}) \text{ and} \\ \text{ac}(\mathcal{F}(t_1)) \wedge \text{ac}(\mathcal{F}(t_2)) &= \mathcal{F}(\text{ac}_{K_1}) \wedge \mathcal{F}(\text{ac}_{K_2}) \end{aligned}$$

10.3 AGG-BASED LOCAL CONFLUENCE ANALYSIS OF PTI NET TRANSFORMATION SYSTEMS

In this section, we discuss, similarly to Section 7.3, the workflow for the analysis of local confluence by applying it to our concrete PTI net transformation systems without and with nested application conditions from Example 12. This workflow, which integrates the application of our theoretical results from Sections 4.1 and 10.1 for the case of rules *without* nested application conditions and the theoretical results from Sections 4.2 and 10.2 for the case of rules *with* nested application conditions, is partially supported by the AGG-tool, which is used for critical pair computation. Note that our approach is tool independent, which means that other tools that are capable to compute and possibly analyze critical pairs (for rules without or with nested application conditions) can be used equivalently. However, in this section, we only exemplify our workflow by using AGG for the critical pair computation since other existing tools that are capable to compute critical pairs like *Henshin* [69, 6], *VERIGRAPH* [305] or *SyGrAV* [53, 290] do not allow the involved rules to make use of NACs and PACs so far.

In the following we consider the two PTI net transformation systems without and with nested application conditions that were already introduced in Example 12. This example is a cutout of the “House of Philosophers” model introduced in [153] where philosophers may leave or join a table. We have already given the rules of the Mobile Dining Philosophers system *without* nested application conditions in Figure 21 and the rules of the Mobile Dining Philosophers system *with* nested application conditions in Figure 22 (where rules make use of PACs).

In fact, we consider in the following four different scenarios for which the corresponding workflows share common steps (see Figure 75). Firstly, we distinguish between PTI net transformation systems without and with nested application conditions. The latter case, in which nested application conditions are used, is separately discussed at the end of this section where we address the differences to the workflow for PTI net transformation systems *without* nested application conditions. Secondly, while the theory introduced in this thesis is tailored to transformation systems, it is beneficial to also consider the case where the transformation system is given with a start structure at hand. In the PTI net context, such combinations of a PTI net transformation system and a start PTI net are called PTI net grammars [88]. Using straightforward adaptations of our theory we can analyze semantical properties also for PTI net grammars. The core idea is then to analyze these properties not for all PTI nets that are derivable within the PTI net transformation system but only for those that are reachable from the given start PTI net by rule application. The discussion of how the possible usage of a start PTI net affects our workflow is included along the way.

Our proposed workflow for the local confluence analysis of a PTI net transformation system *without* nested application conditions consists of the following four steps while the workflow for the analysis of the corresponding PTI net grammar contains one additional intermediate step (see the third step below). Firstly, we apply the restricted \mathcal{M} -functor \mathcal{F}_{PTI} from Definition 64 to the considered PTI net transformation system (or to the considered PTI net grammar) obtaining the corresponding typed attributed graph transformation system (or the corresponding typed attributed graph grammar). Secondly, we compute all critical pairs of the translated transformation rules using the AGG-tool. In the

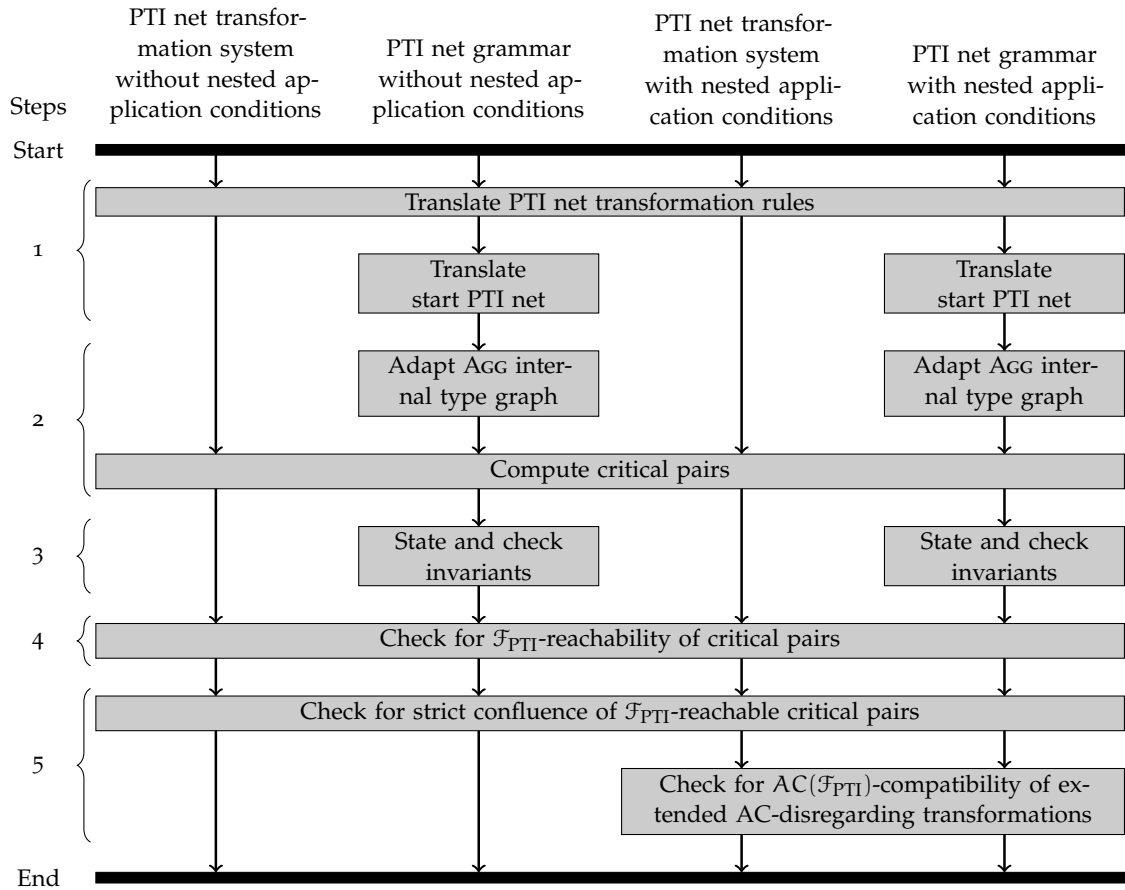


Figure 75: Workflow for local confluence analysis of PTI net transformation systems and PTI net grammars without or with nested application conditions

case of a translated PTI net grammar we apply invariants during the critical pair computation that are given by the restriction of the AGG internal type graph using multiplicities to specify translated PTI nets that are provably not reachable from the translated start PTI net by rule application and that are therefore irrelevant in the context of the analysis of the considered PTI net grammar. Note that the invariants defined by the restriction of the AGG internal type graph are in general not sufficient to ensure that all computed overlapping graphs are relevant for the further analysis. Hence, for translated PTI net grammars the satisfaction of additional invariants is checked in the next step to improve the over-approximation of relevant critical pairs. Thirdly, only in the case of a translated PTI net grammar, we again apply invariants (given e.g. by nested graph conditions [135]) that can be established by invariant verification techniques as e.g. proposed in [66, 67] to exclude further computed critical pairs whose overlapping graphs are provably not reachable from the translated start PTI net by rule application. Fourthly, we select those computed critical pairs that are \mathcal{F}_{PTI} -reachable. This elimination of non- \mathcal{F}_{PTI} -reachable critical pairs is required by our theory (see Theorem 24) where critical pairs must not be considered in the further analysis when their overlapping graphs do not correspond to some translated valid PTI net. Finally, we determine whether all these \mathcal{F}_{PTI} -reachable

critical pairs are strictly confluent, which is also partially supported by AGG³. If all computed \mathcal{F}_{PTI} -reachable critical pairs (that are additionally compatible with invariants for the case of a translated PTI net grammar) are strictly confluent, we obtain local confluence of the considered PTI net transformation system (or of the considered PTI net grammar) by application of Theorem 24⁴.

Note that tools employed, such as AGG in our case, may not succeed in computing the critical pairs for concrete transformation systems when the provided resources are insufficient relative to the size of the left-hand sides of the transformation rules. The restriction then to the corresponding PTI net grammar and the usage of invariants (as discussed in detail above in the second and the third steps of the workflow) can help to successfully execute critical pair computation in some of these cases.

We now apply our workflow to the PTI net transformation system *without* nested application conditions from Example 12 and, by additionally using the start PTI net from Figure 20, to the corresponding PTI net grammar.

Firstly, we translate the PTI net transformation rules given in Figure 21 according to Example 17 (see Figure 71 there) together with the start PTI net from Figure 20 obtaining the corresponding start typed attributed graph.

Secondly, we apply AGG on the obtained set of translated transformation rules to compute all critical pairs of the translated transformation system. However, for our concrete PTI net transformation system AGG was unable to compute all critical pairs since the AGG implementation of the critical pair computation needed more memory than was available on our machine⁵.

Trying still to obtain some verification results for the considered PTI net transformation system, we restricted our analysis to the corresponding translated PTI net grammar and stated, based on the translated transformation rules and the translated start PTI net, some useful invariants. We determined invariants in the form of multiplicities for edges⁶ that we integrated into the AGG internal type graph (see Figure 76). The validity of the invariants stated by these multiplicities can be easily verified by inspection of the translated rules from Figure 71 and the start PTI net from Figure 20 translated by \mathcal{F}_{PTI} . However, such suitable type graph restrictions do not exist for every PTI net grammar. Note that to improve the clarity of AGG illustrations we also used additional String attributes for the graph nodes of types *Place* and *Trans* (see the type graph in Figure 76) representing the names of the respective places and transitions used in Example 12 but

³ Let K be a minimal overlapping of the left-hand sides of two conflicting rules computed by AGG. Let $P_1 \leftarrow K \Rightarrow P_2$ be the corresponding critical pair. Then using AGG we can apply the rules of the considered transformation system to P_1 and P_2 trying to merge them to a unique (up to isomorphism) common result graph. The strictness condition is then to be checked manually.

⁴ Note that when considering a PTI net grammar we expect that a result similar to Theorem 24 is also valid.

⁵ Our machine (CPU: $2 \times \text{E5-2630} @ 2.3\text{GHz} \times 6 \text{ cores} \times 2 \text{ threads}$, RAM: 384GB DDR3), which supports up to 24 threads, was insufficient as the single thread used by AGG required more memory than available.

⁶ For the AGG type graph given in Figure 76, the multiplicity 1..2 associated to the *t2p*-edge expresses that in the context of our example the graph nodes of the type *Place* are allowed to have 1 or 2 incoming *t2p*-edges while the multiplicity 1..3 means that the graph nodes of the type *Trans* are allowed to have 1 to 3 outgoing *t2p*-edges. Similarly, the multiplicity 1..2 associated to the *p2t*-edge expresses that the graph nodes of the type *Place* can have 1 or 2 outgoing *p2t*-edges while the multiplicity 1..3 means that the graph nodes of the type *Trans* can have 1 to 3 incoming *p2t*-edges. Finally, the multiplicity 0..1 associated to the *to2p*-edge expresses that the graph nodes of the type *Place* are allowed to have 0 or 1 incoming *to2p*-edges while the multiplicity 1 means that the graph nodes of the type *Token* must have exactly 1 outgoing edge.

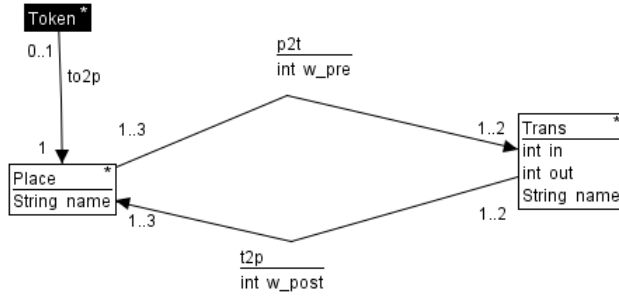


Figure 76: AGG type graph for the Mobile Dining Philosophers system extended by multiplicities

in the following we computed critical pairs without these additional attributes since they would influence the critical pairs computed by AGG.

Unfortunately, the AGG-tool was also not capable to compute all critical pairs when using the adapted type graph due to insufficient memory. On our machine AGG only succeeded in computing the critical pairs for the rule pair $(\mathcal{F}_{PTI}(\text{JoinTable}), \mathcal{F}_{PTI}(\text{JoinTable}))$ and, hence, our subsequent analysis must remain incomplete for the example at hand. We continue our analysis by considering the 25 critical pairs that were computed by AGG in about three days. Note that these 25 critical pairs computed for the translated PTI net grammar are also critical pairs that would be expected to be computed for our translated PTI net transformation system and, hence, in the following we discuss further analysis steps also for the translated PTI net transformation system.

Thirdly, to exclude further computed critical pairs from the subsequent analysis, we apply additional invariants to our translated PTI net grammar. These invariants can be stated e.g. using nested graph conditions [135] and invariant verification techniques such as [66, 67] may be used to check whether the translated transformation system together with the translated start PTI net satisfy these invariants. Then, based on the stated invariants, we can detect (for example using tools such as AutoGraph [278]) critical pairs whose overlapping graphs cannot be contained in a graph that is reachable from the start graph of the translated transformation system by rule application. Moreover, for full automation of the invariant checking procedure we can simply attempt to verify that a given overlapping graph of a critical pair is a pattern that is invariantly not contained in any graph that is reachable from the translated start PTI net and exclude this critical pair from further analysis if the invariant check confirms this conjecture. However, we believe that using small subgraphs of overlapping graphs for the invariants may be more appropriate to reduce computation costs for the invariant check and may also allow to exclude multiple critical pairs at once.

For our considered translated PTI net grammar we determined that 21 of the above mentioned 25 computed critical pairs have overlapping graphs that cannot be contained in a graph that is reachable by rule application from the translated start PTI net. That is, we found out that only four of the computed critical pairs are compatible with the invariants and, thus, have to be evaluated for being \mathcal{F}_{PTI} -reachable in the next step.

An example for such an excluded critical pair is given in Figure 77. Therein the rule $\mathcal{F}_{PTI}(\text{JoinTable})$ is depicted twice on the left and the overlapping of the left-hand sides of the rules is depicted on the right. The overlapping graph that shows two standing philosophers sharing a substructure (namely the place 5, the token 9, and the edge 22 between them) obviously cannot be contained in a graph that is reachable by rule application from the translated start PTI net.

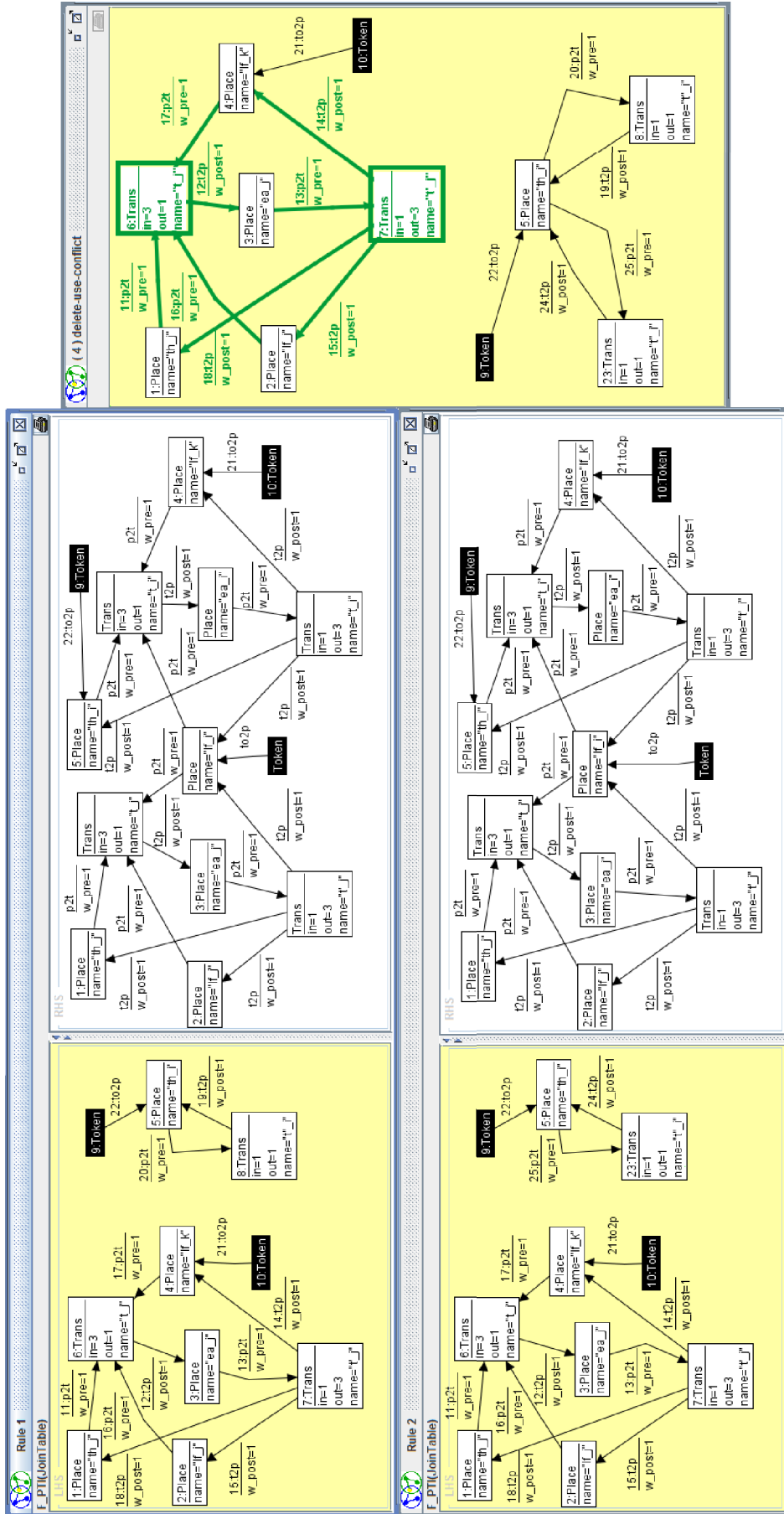


Figure 77: A critical pair computed by AGG for the rule pair $(\mathcal{F}_{PTI}(\text{JoinTable}), \mathcal{F}_{PTI}(\text{JoinTable}))$ non-reachable from the translated start PTI net by rule application

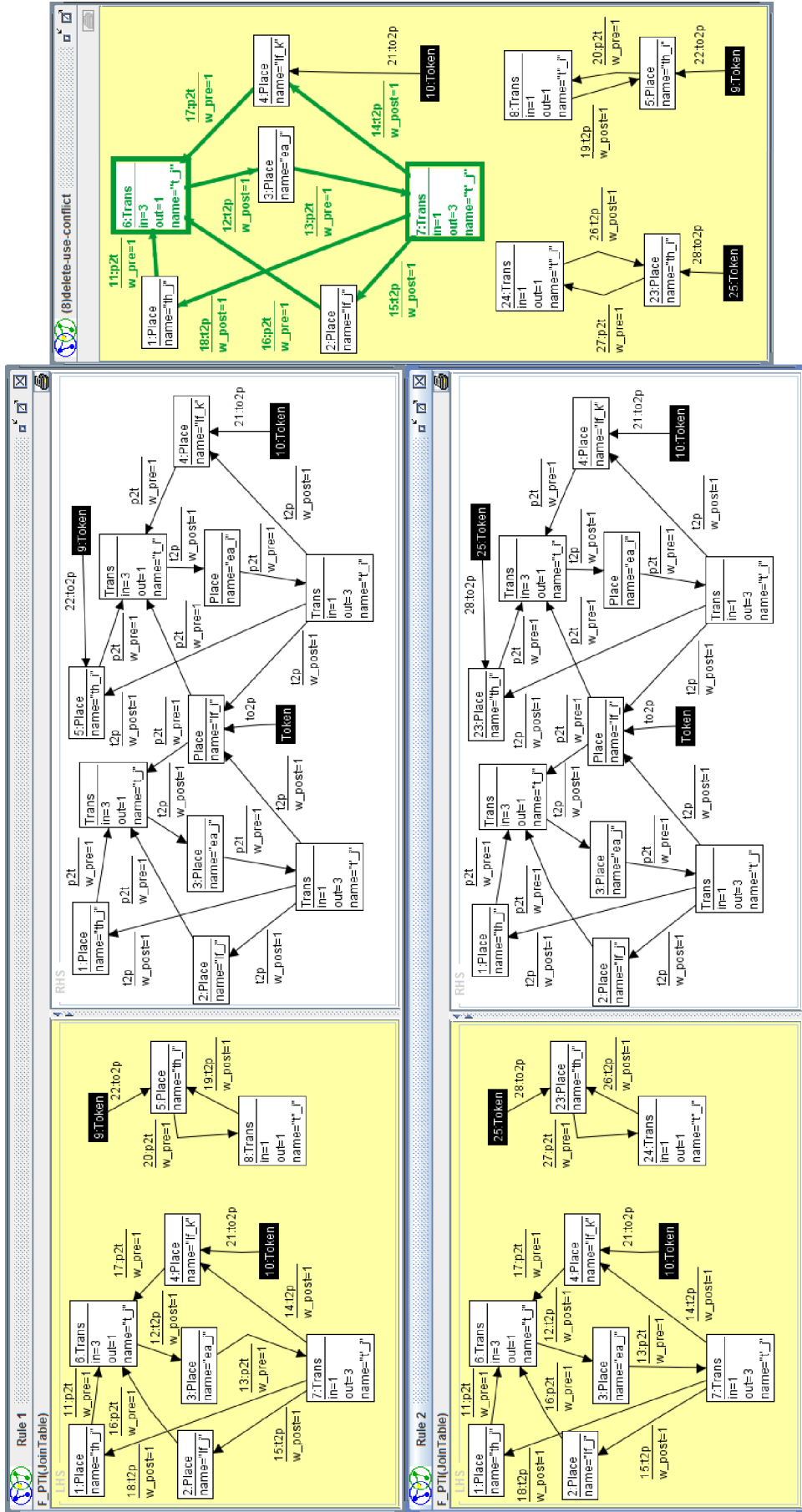


Figure 78: An \mathcal{F}_{PTII} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{PTII}(\text{JoinTable}), \mathcal{F}_{PTII}(\text{JoinTable}))$

Fourthly, we exclude from the further analysis those critical pairs that have overlapping graphs that are not \mathcal{F}_{PTI} -reachable. This step is of great importance because critical pairs that are not \mathcal{F}_{PTI} -reachable do not need to be strictly confluent according to Theorem 24.

Considering our translated PTI net transformation system (or equivalently the corresponding translated PTI net grammar), we can see that non- \mathcal{F}_{PTI} -reachable critical pairs for the pair of transformation rules ($\mathcal{F}_{\text{PTI}}(\text{JoinTable}), \mathcal{F}_{\text{PTI}}(\text{JoinTable})$) occur for similar reasons as in Figure 53 for the case of the translated hypergraph transformation system where the *in* and *out* attributes of the Server hyperedge 2 are not valid w.r.t. the actual number of incoming and outgoing edges in the computed overlapping graph. Therefore, we do not give an example of such a non- \mathcal{F}_{PTI} -reachable critical pair explicitly here.

However, for the rule pair ($\mathcal{F}_{\text{PTI}}(\text{JoinTable}), \mathcal{F}_{\text{PTI}}(\text{JoinTable})$) we obtained only one critical pair that is compatible with the invariants and \mathcal{F}_{PTI} -reachable, i.e., we excluded three additional critical pairs from the further analysis in this step. This single remaining critical pair is given in AGG-notation in Figure 78. The conflict given by this critical pair is a delete-use-conflict, which is obviously caused when two standing philosophers want to take place between the same two philosophers already sitting at the table.

Finally, following our workflow, we verify the strict confluence of the remaining critical pairs to obtain that the considered PTI net transformation system (or the corresponding PTI net grammar) is locally confluent by applying Theorem 24. Considering the translated PTI net transformation system (or the translated PTI net grammar), it is obvious that both rules (for joining and leaving a table) can be reversed by applying the corresponding inverse rule, which implies that a common PTI net for joining two conflicting rule applications is always derivable. Moreover, we expect that the strictness condition can be verified for all \mathcal{F}_{PTI} -reachable critical pairs (additionally compatible with invariants for the case of the PTI net grammar) by constructing the corresponding strictness diagrams according to Definition 14. The construction of strictness diagrams is possible since for every expected overlapping graph each application of one of the rules can be reversed by an application of the corresponding inverse rule.

As an example for such a strict confluence analysis we consider the only remaining critical pair of the rule pair ($\mathcal{F}_{\text{PTI}}(\text{JoinTable}), \mathcal{F}_{\text{PTI}}(\text{JoinTable})$) from above (see Figure 79 depicting the corresponding \mathcal{F}_{PTI} -reachable critical pair diagram⁷ according to Definition 52). This \mathcal{F}_{PTI} -reachable critical pair is strictly confluent since, following another argumentation compared to the one above, after one of the standing philosophers takes place between the two already sitting philosophers, the second standing philosopher can always choose the “right” position at the table to take place such that we obtain at least one graph that can be derived to join the different results. The application of the rules in the mentioned order is depicted in Figure 80, which additionally shows that the corresponding strictness condition is satisfied. The spans $\mathcal{F}_{\text{PTI}}(P_1) \xrightarrow{\mathcal{F}_{\text{PTI}}(w_1)} \mathcal{F}_{\text{PTI}}(N_1) \xrightarrow{\mathcal{F}_{\text{PTI}}(v_1)} \mathcal{F}_{\text{PTI}}(K)$ and $\mathcal{F}_{\text{PTI}}(K) \xrightarrow{\mathcal{F}_{\text{PTI}}(v_2)} \mathcal{F}_{\text{PTI}}(N_2) \xrightarrow{\mathcal{F}_{\text{PTI}}(w_2)} \mathcal{F}_{\text{PTI}}(P_2)$ in the upper half of the diagram represent the two conflicting rule applications of the rule $\mathcal{F}_{\text{PTI}}(\text{JoinTable})$. The graph $\mathcal{F}_{\text{PTI}}(N)$ in the center of the diagram is the largest subgraph of the computed overlapping graph $\mathcal{F}_{\text{PTI}}(K)$ that is preserved by the critical pair. In the lower half of the diagram we can see how the two direct graph transformations $\mathcal{F}_{\text{PTI}}(K) \xrightarrow{\mathcal{F}_{\text{PTI}}(\text{JoinTable}), \mathcal{F}_{\text{PTI}}(o_1)} \mathcal{F}_{\text{PTI}}(P_1)$ and $\mathcal{F}_{\text{PTI}}(K) \xrightarrow{\mathcal{F}_{\text{PTI}}(\text{JoinTable}), \mathcal{F}_{\text{PTI}}(o_2)} \mathcal{F}_{\text{PTI}}(P_2)$ can be mer-

⁷ For better clarity of graph depiction we omit the edge typing (to2p, p2t, t2p) as well as the edge attributes ($w_{\text{pre}}, w_{\text{post}}$).

ged by applying once more the rule $\mathcal{F}_{\text{PTI}}(\text{JoinTable})$ at an adequate match to the graphs $\mathcal{F}_{\text{PTI}}(P_1)$ and $\mathcal{F}_{\text{PTI}}(P_2)$. The strictness condition is satisfied since $\mathcal{F}_{\text{PTI}}(N)$ is preserved by the merging transformation steps $\mathcal{F}_{\text{PTI}}(P_1) \Rightarrow \mathcal{F}_{\text{PTI}}(K'')$ and $\mathcal{F}_{\text{PTI}}(P_2) \Rightarrow \mathcal{F}_{\text{PTI}}(K'')$.

However, while we believe that the considered PTI net transformation system *without* nested application conditions as well as the corresponding PTI net grammar are locally confluent, we cannot conclude this from the presented considerations using Theorem 24 because we only considered the critical pairs computed by AGG for the rule pair $(\mathcal{F}_{\text{PTI}}(\text{JoinTable}), \mathcal{F}_{\text{PTI}}(\text{JoinTable}))$ of the translated PTI net grammar. As soon as the critical pairs are computable by, for example, an enhanced version of AGG or another tool, we can complete the above described inspection of critical pairs for other pairs of transformation rules in the context of our translated PTI net transformation system (or the corresponding PTI net grammar).

The corresponding workflow (see Figure 75) for local confluence analysis of a PTI net transformation system (or a PTI net grammar) *with* nested application conditions is very similar to that of the plain case described before. The main difference consists in the fact that we have to show, in addition to the strict confluence of all computed \mathcal{F}_{PTI} -reachable critical pairs (that are assumed to be additionally compatible with invariants for the case of a translated PTI net grammar), also the $\text{AC}(\mathcal{F}_{\text{PTI}})$ -compatibility of the corresponding extended AC-disregarding transformations according to Definition 60. All other analysis steps from the workflow introduced before remain unchanged. If all computed \mathcal{F}_{PTI} -reachable critical pairs (that are additionally compatible with invariants for the case of a translated PTI net grammar) are strictly $\text{AC}(\mathcal{F}_{\text{PTI}})$ -confluent, then we obtain local confluence of the considered PTI net transformation system (or of the considered PTI net grammar) *with* nested application conditions by application of Theorem 25⁸.

Unfortunately, as already explained before for the case of rules *without* nested application conditions, AGG was not capable to compute the critical pairs for all pairs of transformation rules. Obviously, this problem may become even more severe when additionally incorporating PACs. Indeed, AGG failed to compute critical pairs for any pair of transformation rules (also when considering the corresponding translated PTI net grammar) due to the size of the left-hand sides of the rules and the size of the added PACs, which required too much memory during the computation of the overlapping graphs. Hence, while we again believe that the considered PTI net transformation system *with* PACs as well as the corresponding PTI net grammar are locally confluent, we cannot obtain this using Theorem 25.

The considered example already demonstrates that more adequate tool support is required. This may be achieved by adaptation and optimization of historically grown tools such as AGG or by the development of new tools where the algorithms given by our theoretical results can be implemented in a clean environment using techniques such as multi-threading.

8 Note that when considering a PTI net grammar we expect that a result similar to Theorem 25 is also valid.

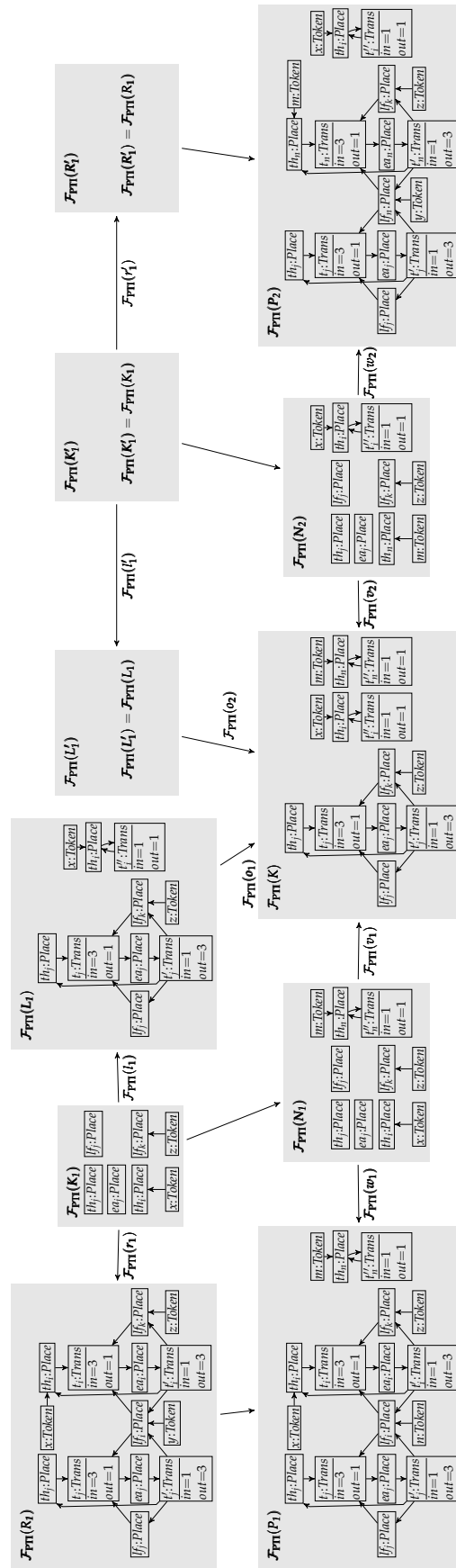
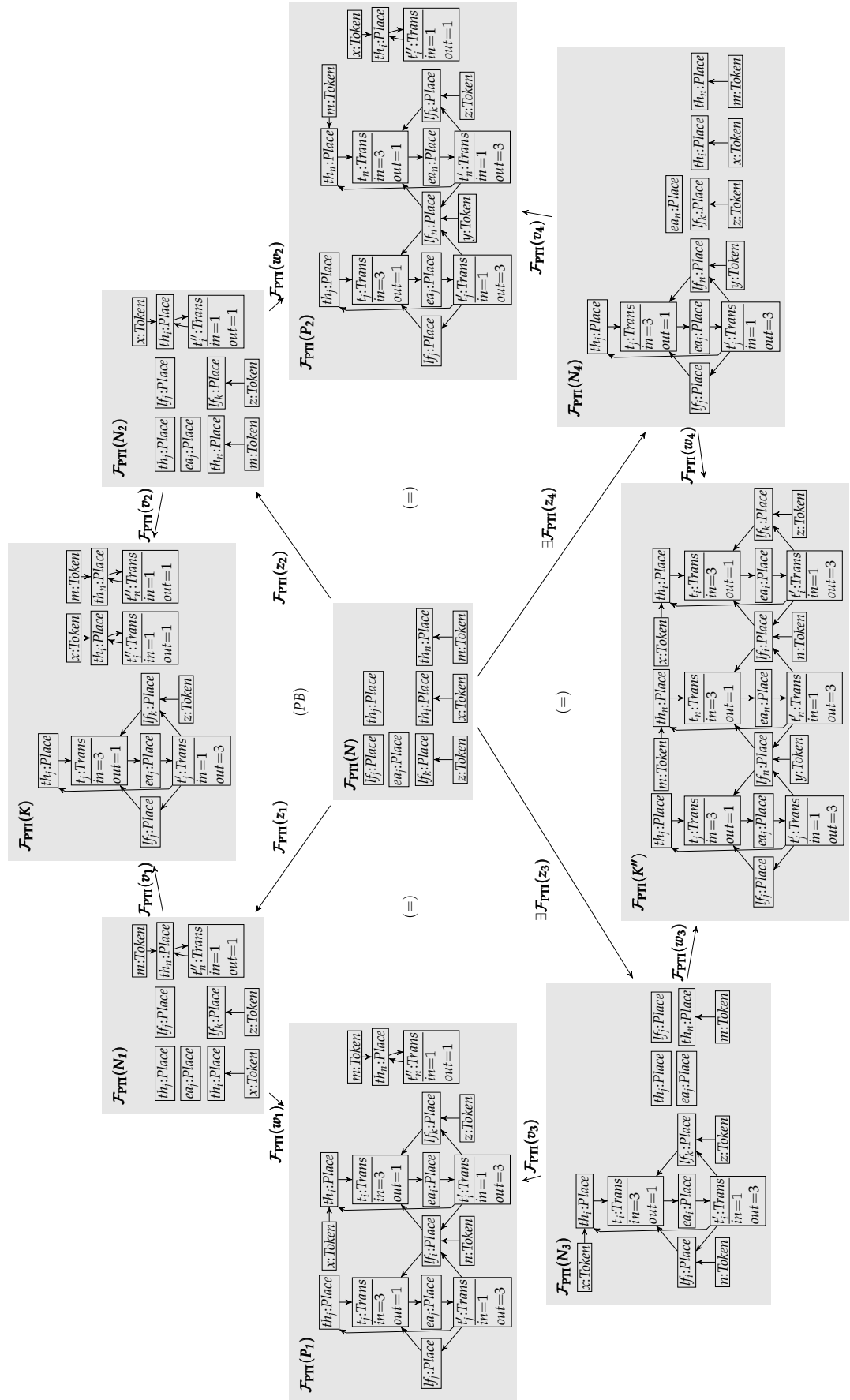


Figure 79: A critical pair diagram depicting the \mathcal{F}_{PTII} -reachable critical pair for the rule pair $(\mathcal{F}_{PTII}(\text{JoinTable}), \mathcal{F}_{PTII}(\text{JoinTable}))$

Figure 80: Strictness diagram for the rule pair $(\mathcal{F}_{PT}(\text{JoinTable}), \mathcal{F}_{PT}(\text{JoinTable}))$

10.4 TERMINATION, CONFLUENCE, AND FUNCTIONAL BEHAVIOR OF PTI NET TRANSFORMATION SYSTEMS

In this section, we show, similarly to Section 7.4, how we can analyze the semantical properties of termination, confluence, and functional behavior for PTI net transformation systems by applying the results from Section 4.3 to the restricted \mathcal{M} -functor \mathcal{F}_{PTI} between the categories of PTI nets and typed attributed graphs. Moreover, we discuss for our Mobile Dining Philosophers systems introduced in Example 12 whether the mentioned semantical properties hold.

According to Theorems 12, 13 and Remarks 14, 15, the analysis results for termination, confluence, and functional behavior of PTI net transformation systems without or with nested application conditions can be directly derived using the requirements that were already shown to hold for the restricted \mathcal{M} -functor \mathcal{F}_{PTI} when we considered the functorial transfer of behavior and local confluence for PTI net transformation systems. We formulate and show the mentioned results in the following two theorems.

Theorem 26 (\mathcal{F}_{PTI} -Transfer of Termination).

Consider a PTI net transformation system $(\mathbf{PTINet}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$ where P are rules without or with nested application conditions, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then $(\mathbf{PTINet}, \mathcal{M}_1, P)$ is terminating iff $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$ is \mathcal{F}_{PTI} -terminating.

Proof.

In order to use the results from Theorem 12 and Remark 14 for the case of transformations without or with nested application conditions, we have to show that \mathcal{F}_{PTI} translates and creates (direct) transformations without or with nested application conditions for the case of \mathcal{M} -match morphisms according to Theorem 2 and Remark 12, i.e., we have to show that $(\mathbf{PTINet}, \mathcal{M}_1)$ has initial pushouts, \mathcal{F}_{PTI} is a restricted \mathcal{M} -functor according to Definition 44 as well as that \mathcal{F}_{PTI} creates injective morphisms and preserves initial pushouts over injective morphisms. In fact, we have that $\mathcal{F}_{\text{PTI}}(\mathcal{M}_1) \subseteq \mathcal{M}_2$, i.e., \mathcal{F}_{PTI} preserves injectivity of morphisms by Lemma 42 and \mathcal{F}_{PTI} preserves pushouts of injective morphisms according to Lemma 43. Moreover, according to Fact 11 the category $(\mathbf{PTINet}, \mathcal{M}_1)$ has initial pushouts over injective morphisms. Finally, creation of injective morphisms by \mathcal{F}_{PTI} is shown in Lemma 45 and preservation of initial pushouts over injective morphisms follows from Lemma 50. \square

Theorem 27 (\mathcal{F}_{PTI} -Transfer of Confluence and Functional Behavior).

Consider a PTI net transformation system $(\mathbf{PTINet}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$ where P are rules without or with nested application conditions, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then $(\mathbf{PTINet}, \mathcal{M}_1, P)$ is locally confluent and terminating iff $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$ is locally confluent for all translated transformation spans and \mathcal{F}_{PTI} -terminating. Moreover, $(\mathbf{PTINet}, \mathcal{M}_1, P)$ is confluent and has functional behavior if $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$ is locally confluent for all translated transformation spans and \mathcal{F}_{PTI} -terminating.

Proof.

In order to use the results from Theorem 13 and Remark 15 for the case of transformations without or with nested application conditions, we have again to show that \mathcal{F}_{PTI} translates and creates (direct) transformations without or with nested application conditions for the case of \mathcal{M} -match

morphisms according to Theorem 2 and Remark 12. This can be done equivalently to the proof of Theorem 26. \square

After we have shown that the general results of our approach concerning termination, confluence, and functional behavior are applicable to the restricted \mathcal{M} -functor \mathcal{F}_{PTI} , we now want to analyze, using these results, whether the mentioned semantical properties hold for our Mobile Dining Philosophers systems from Example 12.

Considering the sets of translated transformation rules given in Figure 71 and Figure 72 for the case of PTI net transformation systems without application conditions or with PACs, respectively, (and additionally considering the start PTI net from Figure 20 translated by \mathcal{F}_{PTI} for the case of the corresponding PTI net grammars) we can easily see that the resulting translated PTI net transformation systems (as well as the corresponding translated PTI net grammars) are not \mathcal{F}_{PTI} -terminating because the rules for joining and leaving a table can be applied alternately unlimited often, which implies that also the original PTI net transformation systems (as well as the corresponding PTI net grammars) are not terminating according to Theorem 26⁹.

Furthermore, as discussed at the end of the previous section, we cannot make any statement about the local confluence of the considered PTI net transformation systems (as well as about the corresponding PTI net grammars) since AGG was not capable in any case to compute critical pairs for all pairs of transformation rules.

Hence, while we believe that the PTI net transformation systems from Example 12 (as well as the corresponding PTI net grammars) are confluent but have no functional behavior, this cannot be obtained using Theorem 27⁹. The confluence property holds for the considered PTI net transformation systems (as well as for the corresponding PTI net grammars) because both rules (for joining and leaving a table) can be reversed by applying the corresponding inverse rule, which implies that a common PTI net for joining two transformation sequences starting with the same PTI net is always derivable. Moreover, the considered PTI net transformation systems (as well as the corresponding PTI net grammars) have no functional behavior because from at least one PTI net (e.g. from the start PTI net given in Figure 20) there is no finite sequence to a unique PTI net, to which no transformation rules are applicable anymore, since it is always possible for a philosopher either to join or to leave the table.

We do not use AGG for the automated verification of the termination property for our running example because, as already mentioned before for the case of hypergraph transformation systems, AGG is only capable to analyze termination for typed graph transformation systems with injective rules, injective match morphisms, and injective NACs.

⁹ Note that when considering a PTI net grammar, we expect that the results similar to Theorems 26 and 27 are also valid.

Part V

FUNCTOR DECOMPOSITION STRATEGY FOR VERIFICATION OF REQUIREMENTS AND ITS APPLICATION

FUNCTOR DECOMPOSITION STRATEGY FOR VERIFICATION OF REQUIREMENTS

The analysis of semantical properties like behavioral equivalence, local confluence, termination, and functional behavior of \mathcal{M} -adhesive transformation systems using (restricted) \mathcal{M} -functors is possible if the source and the target categories of the considered (restricted) \mathcal{M} -functors and the (restricted) \mathcal{M} -functors themselves satisfy all of the requirements discussed in Chapters 3 and 4. To possibly reduce the complexity of proofs that show the satisfaction of these requirements for concrete applications of our theoretical framework, we propose in this chapter a functor decomposition strategy characterizing the applicability of the developed general theory of (restricted) \mathcal{M} -functors. Subsequently, in Chapter 12 we use the introduced functor decomposition strategy to obtain the results enabling the analysis of hypergraph and PTI net transformation systems as before.

When carefully comparing the respective proofs needed in the context of the functor decomposition strategy presented subsequently and the proofs used in the context of the regular strategy for the requirement verification presented in Part ii for the two concrete functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} , it turned out that except for a few properties (such as e.g. the preservation of \mathcal{E}' -instances) the functor decomposition strategy required slightly shorter proofs in total. Hence, we believe that the functor decomposition strategy is usually a good choice for the requirement verification but it introduces some overhead that may not be compensated for concrete functors where proofs are to be carried out at a low level of abstraction or when fundamental properties on the source category of the considered inclusion functor (such as e.g. the existence of pair factorization) have to be verified.

We begin this chapter with the introduction of several categorical notions that are required for the formulation of the functor decomposition strategy. Afterwards, in Section 11.1 we establish the functor decomposition strategy and adapt it subsequently in Section 11.2 to the verification of the sufficient properties that are required by the general theory of *restricted* \mathcal{M} -functors.

As the first step, we recall the notions of functor transformation, functor equivalence, category equivalence, and category isomorphism that are defined similarly to [1]. These notions have a technical nature and are used in the following in the context of the functor decomposition strategy for the requirement verification.

Definition 65 (Functor Transformation [1]).

Let \mathbf{C}, \mathbf{D} be categories and $F, G : \mathbf{C} \rightarrow \mathbf{D}$ be functors. A family of morphisms $(t(A))_{A \in \text{Ob}_{\mathbf{C}}}$ with $t(A) : F(A) \rightarrow G(A)$ in \mathbf{D} is called a functor transformation iff for arbitrary objects $A, B \in \text{Ob}_{\mathbf{C}}$ and a morphism $f : A \rightarrow B$ it holds that $G(f) \circ t(A) = t(B) \circ F(f)$ (see the diagram on the right).

$$\begin{array}{ccc} F(A) & \xrightarrow{t(A)} & G(A) \\ F(f) \downarrow & = & \downarrow G(f) \\ F(B) & \xrightarrow{t(B)} & G(B) \end{array}$$

Definition 66 (Functor Equivalence [1]).

Let \mathbf{C}, \mathbf{D} be categories, $F, G : \mathbf{C} \rightarrow \mathbf{D}$ be functors, and $t = (t(A) : F(A) \rightarrow G(A))_{A \in \text{Ob}_{\mathbf{C}}}$ be a functor transformation. Then $t : F \cong G$ is called a functor equivalence iff for every object $A \in \text{Ob}_{\mathbf{C}}$ it holds that $t(A) : F(A) \rightarrow G(A)$ is an isomorphism in \mathbf{D} .

Definition 67 (Category Equivalence [1]).

Let \mathbf{C} and \mathbf{D} be categories. Then \mathbf{C} and \mathbf{D} are called equivalent iff there are functors $F : \mathbf{C} \rightarrow \mathbf{D}$, $G : \mathbf{D} \rightarrow \mathbf{C}$ and functor equivalences $t_1 : G \circ F \cong \text{Id}_{\mathbf{C}}$, $t_2 : F \circ G \cong \text{Id}_{\mathbf{D}}$ ¹ with the corresponding identity functors $\text{Id}_{\mathbf{C}} : \mathbf{C} \rightarrow \mathbf{C}$ and $\text{Id}_{\mathbf{D}} : \mathbf{D} \rightarrow \mathbf{D}$.

Definition 68 (Category Isomorphism [1]).

Let \mathbf{C} and \mathbf{D} be categories. Then \mathbf{C} and \mathbf{D} are called isomorphic iff there are functors $F : \mathbf{C} \rightarrow \mathbf{D}$ and $G : \mathbf{D} \rightarrow \mathbf{C}$ such that $G \circ F = \text{Id}_{\mathbf{C}}$ and $F \circ G = \text{Id}_{\mathbf{D}}$ with the corresponding identity functors $\text{Id}_{\mathbf{C}} : \mathbf{C} \rightarrow \mathbf{C}$ and $\text{Id}_{\mathbf{D}} : \mathbf{D} \rightarrow \mathbf{D}$.

11.1 VERIFICATION OF REQUIREMENTS FOR THE GENERAL THEORY OF \mathcal{M} -FUNCTORS

The main idea of the functor decomposition strategy that we propose here is to verify the sufficient properties required by the general theory of \mathcal{M} -functors not for the functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ but only for the corresponding inclusion functor from a certain subcategory of the target category of \mathcal{F} into the entire target category of \mathcal{F} .

To obtain the inclusion functor for a functor \mathcal{F} at hand, we first construct a suitable category equivalence between the source category $(\mathbf{C}_1, \mathcal{M}_1)$ and the subcategory $(\mathbf{C}'_2, \mathcal{M}_2^*)$ of the target category of \mathcal{F} (where $\mathcal{M}_2^* \subseteq \mathcal{M}_2$), which contains the \mathcal{F} -images only. The corresponding monomorphism and epimorphism classes \mathcal{M}_2^* and \mathcal{E}_2^* of this subcategory are given by $\mathcal{M}_2^* = \mathcal{F}(\mathcal{M}_1)$ and $\mathcal{E}_2^* = \mathcal{F}(\mathcal{E}_1)$, respectively. Then the functor \mathcal{F} , for which we want to verify the required properties, can be considered as a composition of a functor $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}'_2, \mathcal{M}_2^*)$ from the category equivalence and the corresponding inclusion functor $I : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ from the subcategory of the target category of \mathcal{F} into the entire target category of \mathcal{F} .

Building up our functor decomposition strategy, we first show in this section that the functor $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}'_2, \mathcal{M}_2^*)$ from the category equivalence constructed as described above satisfies the sufficient properties required by the general theory of \mathcal{M} -functors. Afterwards, we prove the fact that the functor composition, consisting of \mathcal{F}_C and the corresponding inclusion functor $I : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$, preserves the satisfaction of the required properties. Finally, we conclude from both shown results that a functor \mathcal{F} given by the composition of functors $I \circ \mathcal{F}_C$ satisfies the sufficient properties required by the general theory of \mathcal{M} -functors.

The following definition summarizes all sufficient technical properties that have to be shown to hold for a given functor \mathcal{F} to enable the applicability of all results of our general theory of \mathcal{M} -functors.

Definition 69 (Collection of Properties for \mathcal{M} -Functors).

Let $\text{AS}_1 = (\mathbf{C}_1, \mathcal{P})$, $\text{AS}_2 = (\mathbf{C}_2, \mathcal{F}(\mathcal{P}))$ be transformation systems with distinguished classes of monomorphisms \mathcal{M}_1 , \mathcal{M}_2 , respectively, and $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ be a functor². For the application of the results of our general theory of \mathcal{M} -functors to \mathcal{F} , the following properties have to be verified:

1. \mathcal{F} preserves monomorphisms, i.e., $\mathcal{F}(\mathcal{M}_1) \subseteq \mathcal{M}_2$,

¹ We also write $F : \mathbf{C} \xrightarrow{\sim} \mathbf{D}$ to indicate that the categories \mathbf{C} and \mathbf{D} are equivalent.

² In general, we do not require that the source and the target categories of the considered functor \mathcal{F} are \mathcal{M} -adhesive.

2. \mathcal{F} preserves pushouts along \mathcal{M} -morphisms,
3. \mathcal{F} creates morphisms,
4. \mathcal{F} preserves initial pushouts,
5. \mathcal{F} preserves epimorphisms, i.e., $\mathcal{F}(\mathcal{E}_1) \subseteq \mathcal{E}_2$,
6. \mathcal{F} preserves coproducts,
7. \mathcal{F} creates \mathcal{M} -morphisms,
8. \mathcal{F} preserves pullbacks of \mathcal{M} -morphisms,
9. \mathcal{F} preserves \mathcal{E}' -instances,
10. \mathcal{F} creates \mathcal{E}' -instances.

Note that, when considering transformation systems without nested application conditions, it is sufficient to verify properties 1 – 6 while for the case of transformation systems with nested application conditions all ten properties have to be verified.

Now we construct a category equivalence between the source category of a considered functor \mathcal{F} and the subcategory of its target category containing \mathcal{F} -images only (for objects and morphisms). Our aim here is to show that the functor \mathcal{F}_C from the constructed category equivalence satisfies the technical properties given in Definition 69.

Lemma 58 (\mathcal{F}_C Satisfies Required Properties).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$ such that $(\mathbf{C}_1, \mathcal{M}_1)$ has \mathcal{E}'_1 - \mathcal{M}_1 pair factorization and initial pushouts, a functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$, and functors $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}'_2, \mathcal{M}_2^*)$, $\mathcal{F}_C^{-1} : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_1, \mathcal{M}_1)$ building a category equivalence $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{C}'_2, \mathcal{M}_2^*)$ where $(\mathbf{C}'_2, \mathcal{M}_2^*)$ is the subcategory of $(\mathbf{C}_2, \mathcal{M}_2)$, in which all non- \mathcal{F} -images have been removed, and where the functor \mathcal{F}_C is the restriction of \mathcal{F} . Then the functor \mathcal{F}_C satisfies the properties listed in Definition 69.

Proof.

The detailed proof for this lemma is given in [Appendix A](#) on [page 300](#). □

The next lemma shows that if each of the two functors, \mathcal{F}_C as considered before and the inclusion functor $I : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ from the subcategory of the target category of \mathcal{F} into the entire target category of \mathcal{F} , satisfies the technical properties given in Definition 69, then also their composition satisfies these properties.

Lemma 59 ($I \circ \mathcal{F}_C$ Satisfies Required Properties).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$ such that the underlying \mathcal{M} -adhesive categories $(\mathbf{C}_i, \mathcal{M}_i)$ have \mathcal{E}'_i - \mathcal{M}_i pair factorization for $i \in \{1, 2\}$ and $(\mathbf{C}_1, \mathcal{M}_1)$ has initial pushouts, a functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$, an inclusion functor $I : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$, and functors $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}'_2, \mathcal{M}_2^*)$, $\mathcal{F}_C^{-1} : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_1, \mathcal{M}_1)$ building a category equivalence $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{C}'_2, \mathcal{M}_2^*)$ where $(\mathbf{C}'_2, \mathcal{M}_2^*)$ is the subcategory of $(\mathbf{C}_2, \mathcal{M}_2)$, in which all non- \mathcal{F} -images have been removed. Then the functor composition $I \circ \mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ satisfies the properties listed in Definition 69 if the functors I and \mathcal{F}_C satisfy these properties.

Proof.

The detailed proof for this lemma is given in [Appendix A](#) on [page 305](#). □

In the following theorem, we combine the results shown in the previous two lemmas by stating that a functor \mathcal{F} that is defined as a composition of a functor \mathcal{F}_C :

$(\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}'_2, \mathcal{M}_2^*)$ from the category equivalence and the corresponding inclusion functor $I : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ satisfies the technical properties given in Definition 69 if the inclusion functor I satisfies these properties.

Theorem 28 ($\mathcal{F} = I \circ \mathcal{F}_C$ Satisfies Required Properties).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$ such that the underlying \mathcal{M} -adhesive categories $(\mathbf{C}_i, \mathcal{M}_i)$ have \mathcal{E}'_i - \mathcal{M}_i pair factorization for $i \in \{1, 2\}$ and $(\mathbf{C}_1, \mathcal{M}_1)$ has initial pushouts, a functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$, functors $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}'_2, \mathcal{M}_2^*)$, $\mathcal{F}_C^{-1} : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_1, \mathcal{M}_1)$ building a category equivalence $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{C}'_2, \mathcal{M}_2^*)$ where $(\mathbf{C}'_2, \mathcal{M}_2^*)$ is the subcategory of $(\mathbf{C}_2, \mathcal{M}_2)$, in which all non- \mathcal{F} -images have been removed, and an inclusion functor $I : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ satisfying the properties listed in Definition 69. Then also $(\mathcal{F} = I \circ \mathcal{F}_C) : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ satisfies the properties listed in Definition 69.

Proof.

According to Lemma 58, it holds that the functor \mathcal{F}_C from the category equivalence satisfies the properties given in Definition 69. Moreover, by assumption we have that the inclusion functor I satisfies these properties as well. Applying now Lemma 59, we obtain that also the composition of both functors $I \circ \mathcal{F}_C$ satisfies the required properties. Furthermore, by assumption we have that $\mathcal{F} = I \circ \mathcal{F}_C$ and \mathcal{F} is a functor, which implies that also \mathcal{F} satisfies the properties given in Definition 69. \square

Now we summarize in the following remark, the stepwise workflow for the proposed functor decomposition strategy enabling the requirement verification for concrete functors at hand. As already pointed out before, we believe that this functor decomposition strategy allows us to often reduce the verification effort and the complexity of the proofs to be carried out for concrete applications of our theoretical framework.

Remark 16 (Functor Decomposition Strategy for Verification of Properties Required by the General Theory of \mathcal{M} -Functors).

Let $(\mathbf{C}_1, \mathcal{M}_1, P)$, $(\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$ be \mathcal{M} -adhesive transformation systems and $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ be a functor. To obtain the analysis results for the source transformation system of the considered functor, execute the following steps.

1. Define a subcategory $(\mathbf{C}'_2, \mathcal{M}_2^*)$ of $(\mathbf{C}_2, \mathcal{M}_2)$ containing precisely the \mathcal{F} -images with $\mathcal{M}_2^* = \mathcal{F}(\mathcal{M}_1)$.
2. Construct an equivalence of categories $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{C}'_2, \mathcal{M}_2^*)$.
3. Define an inclusion functor $I : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$.
4. Show the satisfaction of the properties required for the main results of our general theory of \mathcal{M} -functors (see Definition 69) not for the functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ but only for the inclusion functor $I : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$.

11.2 VERIFICATION OF REQUIREMENTS FOR GENERAL THEORY OF RESTRICTED \mathcal{M} -FUNCTORS

Considering functors restricted to \mathcal{M} -morphisms only, we can similarly formulate and use a functor decomposition strategy for the requirement verification as introduced in the previous section. To possibly reduce the complexity of proofs for concrete restricted

functors, we want to verify in the context of the functor decomposition strategy the properties that are required by the general theory of restricted \mathcal{M} -functors not for the restricted functor \mathcal{F}_R but only for the corresponding restricted inclusion functor from the subcategory of the target category of \mathcal{F}_R into the entire target category of \mathcal{F}_R .

This section follows the same lines as the previous section. As the first step, we list the sufficient properties that have to be satisfied to be able to use the general theoretical results for restricted \mathcal{M} -functors. Afterwards, we adapt the functor decomposition strategy, proposed in the previous section, to the framework of restricted \mathcal{M} -functors. In particular, we firstly construct for a given restricted functor $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$ (that is defined on \mathcal{M} -morphisms only) a suitable category equivalence between the source category $\mathbf{C}_1|_{\mathcal{M}_1}$ and the subcategory $\mathbf{C}'_2|_{\mathcal{M}_2^*}$ of the target category of \mathcal{F}_R (where $\mathcal{M}_2^* \subseteq \mathcal{M}_2$), which precisely contains the \mathcal{F}_R -images. Secondly, we show that the restricted functor $\mathcal{F}_{RC} : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}'_2|_{\mathcal{M}_2^*}$ from the constructed category equivalence satisfies the properties required by the general theory of restricted \mathcal{M} -functors. Thirdly, we verify that the satisfaction of these required properties is preserved by the composition of restricted functors \mathcal{F}_{RC} and I_R (for the restricted inclusion functor I_R from the subcategory of the target category of \mathcal{F}_R into the entire target category of \mathcal{F}_R) if each of the restricted functors from the composition satisfies these required properties. Finally, we show that the composition of restricted functors \mathcal{F}_{RC} and I_R , resulting in the considered restricted functor \mathcal{F}_R , satisfies the required properties if the restricted inclusion functor I_R satisfies these properties.

Our first step, as mentioned before, is to summarize in the following definition the sufficient technical properties that have to be verified for a restricted functor \mathcal{F}_R to enable the applicability of all results of our general theory of restricted \mathcal{M} -functors.

Definition 70 (Collection of Properties for Restricted \mathcal{M} -Functors).

Let $AS_1 = (\mathbf{C}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{F}_R(P))$ be transformation systems with distinguished classes of monomorphisms \mathcal{M}_1 , \mathcal{M}_2 , respectively, and $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$ be a functor between two categories restricted to \mathcal{M} -morphisms only³. For the application of the results of our general theory of restricted \mathcal{M} -functors to \mathcal{F}_R , the following properties have to be verified:

1. \mathcal{F}_R preserves monomorphisms, i.e., $\mathcal{F}_R(\mathcal{M}_1) \subseteq \mathcal{M}_2$,
2. \mathcal{F}_R preserves pushouts of \mathcal{M} -morphisms,
3. \mathcal{F}_R creates \mathcal{M} -morphisms,
4. \mathcal{F}_R preserves initial pushouts over \mathcal{M} -morphisms,
5. \mathcal{F}_R preserves epimorphisms, i.e., $\mathcal{F}_R(\mathcal{E}_1) \subseteq \mathcal{E}_2$,
6. \mathcal{F}_R preserves coproducts of \mathcal{M} -morphisms,
7. \mathcal{F}_R preserves pullbacks of \mathcal{M} -morphisms,
8. \mathcal{F}_R preserves \mathcal{E}' -instances,
9. \mathcal{F}_R creates \mathcal{E}' -instances.

Note that, when considering transformation systems without nested application conditions, it is sufficient to verify properties 1 – 6 while for the case of transformation systems with nested application conditions all nine properties have to be verified.

Similarly to the previous section, we now construct a category equivalence between the source category of a considered restricted functor and the subcategory $\mathbf{C}'_2|_{\mathcal{M}_2^*}$ of its target category containing \mathcal{F}_R -images only (for objects and morphisms) and show that

³ In general, we do not require that the source and the target categories of the considered restricted functor \mathcal{F}_R are \mathcal{M} -adhesive. For example, for the restricted functors \mathcal{F}_{PTIC} and I_{PTI} in Section 12.2, we use the target category of \mathcal{F}_{PTIC} and the source category of I_{PTI} that are not \mathcal{M} -adhesive.

the restricted functor \mathcal{F}_{RC} from this category equivalence satisfies the required properties listed in Definition 70. Note that the distinguished monomorphism class \mathcal{M}_2^* of the subcategory $\mathbf{C}_2'|\mathcal{M}_2^*$ is given by $\mathcal{M}_2^* = \mathcal{F}_R(\mathcal{M}_1)$ and the distinguished epimorphism class \mathcal{E}_2^* of the subcategory is given by $\mathcal{E}_2^* = \mathcal{F}_R(\mathcal{E}_1)$.

Lemma 60 (\mathcal{F}_{RC} Satisfies Required Properties).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}_R(P))$ such that $(\mathbf{C}_1, \mathcal{M}_1)$ has \mathcal{E}_1' - \mathcal{M}_1 pair factorization and initial pushouts, a restricted functor $\mathcal{F}_R : \mathbf{C}_1|\mathcal{M}_1 \rightarrow \mathbf{C}_2|\mathcal{M}_2$, and restricted functors $\mathcal{F}_{RC} : \mathbf{C}_1|\mathcal{M}_1 \rightarrow \mathbf{C}_2'|\mathcal{M}_2^*$, $\mathcal{F}_{RC}^{-1} : \mathbf{C}_2'|\mathcal{M}_2^* \rightarrow \mathbf{C}_1|\mathcal{M}_1$ building a category equivalence $\mathcal{F}_{RC} : \mathbf{C}_1|\mathcal{M}_1 \xrightarrow{\sim} \mathbf{C}_2'|\mathcal{M}_2^*$ such that $\mathbf{C}_2'|\mathcal{M}_2^*$ is the subcategory of $\mathbf{C}_2|\mathcal{M}_2$, in which all non- \mathcal{F}_R -images have been removed, and where the restricted functor \mathcal{F}_{RC} is the restriction of \mathcal{F}_R . Then the restricted functor \mathcal{F}_{RC} satisfies the properties listed in Definition 70.

Proof.

The detailed proof for this lemma is given in [Appendix A](#) on page 308. \square

Now we show for the restricted functors $\mathcal{F}_{RC} : \mathbf{C}_1|\mathcal{M}_1 \rightarrow \mathbf{C}_2'|\mathcal{M}_2^*$ and $I_R : \mathbf{C}_2'|\mathcal{M}_2^* \rightarrow \mathbf{C}_2|\mathcal{M}_2$ (where I_R is a restricted inclusion functor from the subcategory of the target category of \mathcal{F}_R into the entire target category of \mathcal{F}_R) that if both of them satisfy the required properties then so does also their composition.

Lemma 61 ($I_R \circ \mathcal{F}_{RC}$ Satisfies Required Properties).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}_R(P))$ such that the underlying \mathcal{M} -adhesive categories $(\mathbf{C}_i, \mathcal{M}_i)$ have \mathcal{E}_i' - \mathcal{M}_i pair factorization for $i \in \{1, 2\}$ and $(\mathbf{C}_1, \mathcal{M}_1)$ has initial pushouts, a restricted functor $\mathcal{F}_R : \mathbf{C}_1|\mathcal{M}_1 \rightarrow \mathbf{C}_2|\mathcal{M}_2$, a restricted inclusion functor $I_R : \mathbf{C}_2'|\mathcal{M}_2^* \rightarrow \mathbf{C}_2|\mathcal{M}_2$, and restricted functors $\mathcal{F}_{RC} : \mathbf{C}_1|\mathcal{M}_1 \rightarrow \mathbf{C}_2'|\mathcal{M}_2^*$, $\mathcal{F}_{RC}^{-1} : \mathbf{C}_2'|\mathcal{M}_2^* \rightarrow \mathbf{C}_1|\mathcal{M}_1$ building a category equivalence $\mathcal{F}_{RC} : \mathbf{C}_1|\mathcal{M}_1 \xrightarrow{\sim} \mathbf{C}_2'|\mathcal{M}_2^*$ such that $\mathbf{C}_2'|\mathcal{M}_2^*$ is the subcategory of $\mathbf{C}_2|\mathcal{M}_2$, in which all non- \mathcal{F}_R -images have been removed. Then the functor composition $I_R \circ \mathcal{F}_{RC} : \mathbf{C}_1|\mathcal{M}_1 \rightarrow \mathbf{C}_2|\mathcal{M}_2$ satisfies the properties listed in Definition 70 if the restricted functors I_R and \mathcal{F}_{RC} satisfy these properties.

Proof.

The detailed proof for this lemma is given in [Appendix A](#) on page 308. \square

The following theorem states that for a constructed category equivalence the restricted functor \mathcal{F}_R , considered as a composition of the restricted functor \mathcal{F}_{RC} from the category equivalence and the corresponding restricted inclusion functor I_R , satisfies the properties listed in Definition 70 if the restricted inclusion functor I_R satisfies these properties.

Theorem 29 ($\mathcal{F}_R = I_R \circ \mathcal{F}_{RC}$ Satisfies Required Properties).

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}_R(P))$ such that the underlying \mathcal{M} -adhesive categories $(\mathbf{C}_i, \mathcal{M}_i)$ have \mathcal{E}_i' - \mathcal{M}_i pair factorization for $i \in \{1, 2\}$ and $(\mathbf{C}_1, \mathcal{M}_1)$ has initial pushouts, a restricted functor $\mathcal{F}_R : \mathbf{C}_1|\mathcal{M}_1 \rightarrow \mathbf{C}_2|\mathcal{M}_2$, restricted functors $\mathcal{F}_{RC} : \mathbf{C}_1|\mathcal{M}_1 \rightarrow \mathbf{C}_2'|\mathcal{M}_2^*$, $\mathcal{F}_{RC}^{-1} : \mathbf{C}_2'|\mathcal{M}_2^* \rightarrow \mathbf{C}_1|\mathcal{M}_1$ building a category equivalence $\mathcal{F}_{RC} : \mathbf{C}_1|\mathcal{M}_1 \xrightarrow{\sim} \mathbf{C}_2'|\mathcal{M}_2^*$ such that $\mathbf{C}_2'|\mathcal{M}_2^*$ is the subcategory of $\mathbf{C}_2|\mathcal{M}_2$, in which all non- \mathcal{F}_R -images have been removed, and a restricted inclusion functor $I_R : \mathbf{C}_2'|\mathcal{M}_2^* \rightarrow \mathbf{C}_2|\mathcal{M}_2$ satisfying the required properties for restricted functors listed in Definition 70. Then also $(\mathcal{F}_R = I_R \circ \mathcal{F}_{RC}) : \mathbf{C}_1|\mathcal{M}_1 \rightarrow \mathbf{C}_2|\mathcal{M}_2$ satisfies the properties listed in Definition 70.

Proof.

According to Lemma 60, it holds that the restricted functor \mathcal{F}_{RC} from the category equivalence satisfies the properties listed in Definition 70. Moreover, by assumption we have that the restricted inclusion functor I_R satisfies these properties as well. Applying now Lemma 61, we obtain that also the composition of both restricted functors $I_R \circ \mathcal{F}_{RC}$ satisfies the required properties. Furthermore, by assumption we have that $\mathcal{F}_R = I_R \circ \mathcal{F}_{RC}$ and \mathcal{F}_R is a restricted functor, which implies that also \mathcal{F}_R satisfies the properties listed in Definition 70. \square

The workflow of the functor decomposition strategy for the requirement verification, given in the following remark, summarizes the steps that have to be executed for concrete restricted functors at hand.

Remark 17 (Functor Decomposition Strategy for Verification of Properties Required by the General Theory of Restricted \mathcal{M} -Functors).

Let $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}_R(P))$ be \mathcal{M} -adhesive transformation systems and $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$ be a restricted functor. To obtain the analysis results for the source transformation system of the considered restricted functor, execute the following steps.

1. Define a subcategory $\mathbf{C}'_2|_{\mathcal{M}_2^*}$ of $\mathbf{C}_2|_{\mathcal{M}_2}$ containing precisely the \mathcal{F}_R -images with $\mathcal{M}_2^* = \mathcal{F}_R(\mathcal{M}_1)$.
2. Construct an equivalence of categories $\mathcal{F}_{RC} : \mathbf{C}_1|_{\mathcal{M}_1} \xrightarrow{\sim} \mathbf{C}'_2|_{\mathcal{M}_2^*}$.
3. Define a restricted inclusion functor $I_R : \mathbf{C}'_2|_{\mathcal{M}_2^*} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$.
4. Show the satisfaction of the properties required for the main results of our general theory of restricted \mathcal{M} -functors (see Definition 70) not for the restricted functor $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$ but only for the restricted inclusion functor $I_R : \mathbf{C}'_2|_{\mathcal{M}_2^*} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$.

After we have established in this chapter the functor decomposition strategy for verification of properties required by the general theory of (restricted) \mathcal{M} -functors, we apply this proposed strategy in the next chapter to our concrete (restricted) functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} .

APPLICATION OF FUNCTOR DECOMPOSITION STRATEGY FOR REQUIREMENT VERIFICATION TO HYPERGRAPH AND PTI NET TRANSFORMATION SYSTEMS

In this chapter, we apply the workflows for the requirement verification given in Remarks 16 and 17 to the concrete functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} , respectively. As we already know, both functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} are injective by instantiation of Lemmas 9 and 10 but not surjective because not every arbitrary typed attributed graph corresponds to some functor-translated hypergraph or PTI net. This implies that the categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ and $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ as well as $\mathbf{PTINet}|_{\mathcal{M}_1}$ and $\mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ are not equivalent.

12.1 APPLICATION TO HYPERGRAPH TRANSFORMATION SYSTEMS

In this section, we apply the proposed functor decomposition strategy for the requirement verification to the functor $\mathcal{F}_{\text{HG}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ introduced in Definition 63. According to the workflow given in Remark 16, we define first a suitable subcategory $(\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$ of $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ containing all \mathcal{F}_{HG} -images of hypergraphs as objects and all \mathcal{F}_{HG} -translated morphisms between these hypergraphs as morphisms, i.e., $\text{Ob}_{\mathbf{SubAGraphs}_{\text{HGTG}}} = \mathcal{F}_{\text{HG}}(\text{Ob}_{\mathbf{HyperGraphs}})$ and $\text{Mor}_{\mathbf{SubAGraphs}_{\text{HGTG}}}(\mathcal{F}_{\text{HG}}(A), \mathcal{F}_{\text{HG}}(B)) = \mathcal{F}_{\text{HG}}(\text{Mor}_{\mathbf{HyperGraphs}}(A, B))$ for arbitrary $A, B \in \text{Ob}_{\mathbf{HyperGraphs}}$. Note that the class of monomorphisms \mathcal{M}_2^* consists of all injective typed attributed graph morphisms, which are \mathcal{F}_{HG} -images of injective hypergraph morphisms from \mathcal{M}_1 , and the class of epimorphisms \mathcal{E}_2^* consists of all surjective typed attributed graph morphisms, which are \mathcal{F}_{HG} -images of surjective hypergraph morphisms from \mathcal{E}_1 . Obviously, the described subcategory is a well-defined category. Furthermore, as we will show later in the proof of Lemma 63, the subcategory $(\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$ has an $\mathcal{E}_2'' - \mathcal{M}_2^*$ pair factorization with the class of jointly epimorphic typed attributed graph morphisms $\mathcal{E}_2'' = \mathcal{F}_{\text{HG}}(\mathcal{E}_1')$ and $\mathcal{M}_2^* = \mathcal{F}_{\text{HG}}(\mathcal{M}_1)$.

After the suitable subcategory of typed attributed graphs $(\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$ is defined, we construct $\mathcal{F}_{\text{HGC}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$ and show that \mathcal{F}_{HGC} is a category equivalence.

Lemma 62 (\mathcal{F}_{HGC} is a Category Equivalence).

The categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ and $(\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$ are equivalent, i.e., there exists a category equivalence $\mathcal{F}_{\text{HGC}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$.

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on page 351. □

Following our workflow, we define as the next step the corresponding inclusion functor $I_{\text{HG}} : (\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ by $I_{\text{HGOb}} : \text{Ob}_{\mathbf{SubAGraphs}_{\text{HGTG}}} \rightarrow \text{Ob}_{\mathbf{AGraphs}_{\text{HGTG}}}$ with $I_{\text{HGOb}}(A) = A$ and $I_{\text{HGMor}} : \text{Mor}_{\mathbf{SubAGraphs}_{\text{HGTG}}}(A, B) \rightarrow \text{Mor}_{\mathbf{AGraphs}_{\text{HGTG}}}(I_{\text{HGOb}}(A), I_{\text{HGOb}}(B))$ with $I_{\text{HGMor}}(f) = f$. In the subsequent lemma, we show that I_{HG} satisfies the required properties from Definition 69, which is the most challenging part when applying the proposed functor decomposition strategy to a concrete functor.

Lemma 63 (I_{HG} Satisfies Required Properties).

Consider a transformation system $(\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{F}_{\text{HG}}(\mathcal{P}))^1$ with a distinguished class of monomorphisms \mathcal{M}_2^* , an \mathcal{M} -adhesive transformation system $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(\mathcal{P}))$, and a functor $\mathcal{F}_{\text{HG}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$. Then the inclusion functor $I_{\text{HG}} : (\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ satisfies the properties listed in Definition 69.

Proof.

The detailed proof of this lemma is given in [Appendix C](#) on [page 355](#). \square

After all four steps specified by the workflow given in Remark 16 are executed, we show in the following theorem that the proposed functor decomposition strategy is applicable to our concrete functor $\mathcal{F}_{\text{HG}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$.

Theorem 30 (\mathcal{F}_{HG} Satisfies Required Properties).

Consider \mathcal{M} -adhesive transformation systems $(\mathbf{HyperGraphs}, \mathcal{M}_1, \mathcal{P})$ and $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\text{HG}}(\mathcal{P}))$. The functor $\mathcal{F}_{\text{HG}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ from Definition 63 satisfies the properties listed in Definition 69, which enables the application of general theory of \mathcal{M} -functors to \mathcal{F}_{HG} .

Proof.

According to [88], there is a general construction for $\mathcal{E}' - \mathcal{M}$ pair factorizations based on coproducts and $\mathcal{E} - \mathcal{M}$ -factorizations, which can be applied to hypergraphs and typed attributed graphs. Since the empty hypergraph resp. empty typed attributed graph are initial in $\mathbf{HyperGraphs}$ resp. $\mathbf{AGraphs}_{\text{HGTG}}$ and we have pushouts in both categories, we also have coproducts in both categories, which are constructed componentwise as disjoint union. The construction of \mathcal{E} - \mathcal{M} -factorizations for hypergraphs and typed attributed graphs is given in Lemma 35 and in [88], respectively. Thus, $\mathbf{HyperGraphs}$ and $\mathbf{AGraphs}_{\text{HGTG}}$ have the corresponding $\mathcal{E}' - \mathcal{M}$ pair factorizations. Moreover, as we have already shown in Lemma 62, $\mathcal{F}_{\text{HGC}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$ is a category equivalence. Finally, the inclusion functor $I_{\text{HG}} : (\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ satisfies the properties listed in Definition 69 according to Lemma 63 and $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ has initial pushouts according to Lemma 7. Thus, \mathcal{F}_{HG} also satisfies the properties listed in Definition 69 by application of Theorem 28. \square

12.2 APPLICATION TO PTI NET TRANSFORMATION SYSTEMS

In this section, we show, following the lines of the functor decomposition strategy for the requirement verification proposed in Chapter 11, that the restricted functor \mathcal{F}_{PTI} from Definition 64 satisfies the properties required by the general theory of restricted \mathcal{M} -functors.

According to the workflow given in Remark 17, we define as the first step a suitable subcategory $\mathbf{SubAGraphs}_{\text{PNTG}}|_{\mathcal{M}_2^*}$ of $\mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ that contains all \mathcal{F}_{PTI} -images of PTI nets as objects and all \mathcal{F}_{PTI} -translated morphisms between these PTI nets as morphisms, i.e., $\text{Ob}_{\mathbf{SubAGraphs}_{\text{PNTG}}|_{\mathcal{M}_2^*}} = \mathcal{F}_{\text{PTI}}(\text{Ob}_{\mathbf{PTINet}|_{\mathcal{M}_1}})$ and $\text{Mor}_{\mathbf{SubAGraphs}_{\text{PNTG}}|_{\mathcal{M}_2^*}}(\mathcal{F}_{\text{PTI}}(A), \mathcal{F}_{\text{PTI}}(B)) = \mathcal{F}_{\text{PTI}}(\text{Mor}_{\mathbf{PTINet}|_{\mathcal{M}_1}}(A, B))$ for arbitrary $A, B \in \text{Ob}_{\mathbf{PTINet}|_{\mathcal{M}_1}}$. The corresponding classes of monomorphisms \mathcal{M}_2^* and epimorphisms \mathcal{E}_2^* consist of injective typed attributed graph morphisms and surjective typed attributed graph morphisms, which are \mathcal{F}_{PTI} -images of injective PTI net morphisms from \mathcal{M}_1 and surjective PTI net morphisms from \mathcal{E}_1 , respectively. For better readability we abbreviate $\mathbf{SubAGraphs}_{\text{PNTG}}|_{\mathcal{M}_2^*}$ in the follow-

¹ In general, we do not require that the source category of the inclusion functor I_{HG} is \mathcal{M} -adhesive.

ing by **SubAGraphs_{PNTG}**. Again, it is obvious that the subcategory introduced above is a well-defined category. Moreover, as we will show later in the proof of Lemma 65, the subcategory **SubAGraphs_{PNTG}** has an $\mathcal{E}_2'' - \mathcal{M}_2^*$ pair factorization with the class of jointly epimorphic typed attributed graph morphisms $\mathcal{E}_2'' = \mathcal{F}_{\text{PTI}}(\mathcal{E}_1')$ and $\mathcal{M}_2^* = \mathcal{F}_{\text{PTI}}(\mathcal{M}_1)$.

In the following lemma, we construct $\mathcal{F}_{\text{PTIC}} : \mathbf{PTINet}|_{\mathcal{M}_1} \xrightarrow{\sim} \mathbf{SubAGraphs}_{\text{PNTG}}$ and show that $\mathcal{F}_{\text{PTIC}}$ is a category equivalence.

Lemma 64 ($\mathcal{F}_{\text{PTIC}}$ is a Category Equivalence).

The categories $\mathbf{PTINet}|_{\mathcal{M}_1}$ and **SubAGraphs_{PNTG}** are equivalent, i.e., there exists a category equivalence $\mathcal{F}_{\text{PTIC}} : \mathbf{PTINet}|_{\mathcal{M}_1} \xrightarrow{\sim} \mathbf{SubAGraphs}_{\text{PNTG}}$.

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on page 391. \square

As the next step, we define the restricted inclusion functor $I_{\text{PTI}} : \mathbf{SubAGraphs}_{\text{PNTG}} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ by $I_{\text{PTIOb}} : \text{Ob}_{\mathbf{SubAGraphs}_{\text{PNTG}}} \rightarrow \text{Ob}_{\mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}}$ with $I_{\text{PTIOb}}(A) = A$ and $I_{\text{PTIMor}} : \text{Mor}_{\mathbf{SubAGraphs}_{\text{PNTG}}}(A, B) \rightarrow \text{Mor}_{\mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}}(I_{\text{PTIOb}}(A), I_{\text{PTIOb}}(B))$ with $I_{\text{PTIMor}}(f) = f$. The following lemma shows then that I_{PTI} satisfies the properties required by the general theory of restricted \mathcal{M} -functors.

Lemma 65 (I_{PTI} Satisfies Required Properties).

Consider a transformation system $(\mathbf{SubAGraphs}_{\text{PNTG}}, \mathcal{F}_{\text{PTI}}(\mathcal{P}))^2$ with a distinguished class of monomorphisms \mathcal{M}_2^* , an \mathcal{M} -adhesive transformation system $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(\mathcal{P}))$, and a restricted functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$. Then the restricted inclusion functor $I_{\text{PTI}} : \mathbf{SubAGraphs}_{\text{PNTG}} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ satisfies the properties listed in Definition 70.

Proof.

The detailed proof for this lemma is given in [Appendix D](#) on page 395. \square

In the subsequent theorem we show that the functor decomposition strategy for the requirement verification proposed in the previous chapter is applicable to our concrete restricted functor \mathcal{F}_{PTI} .

Theorem 31 (\mathcal{F}_{PTI} Satisfies Required Properties).

Consider \mathcal{M} -adhesive transformation systems $(\mathbf{PTINet}, \mathcal{M}_1, \mathcal{P})$ and $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(\mathcal{P}))$. The restricted functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64 satisfies the properties listed in Definition 70, which enables the application of general theory of restricted \mathcal{M} -functors to \mathcal{F}_{PTI} .

Proof.

According to [88], there is a general construction for $\mathcal{E}' - \mathcal{M}$ pair factorizations based on coproducts and $\mathcal{E} - \mathcal{M}$ -factorizations, which can be applied to PTI nets and typed attributed graphs. Since the empty PTI net resp. empty typed attributed graph are initial in the categories **PTINet** resp. **AGraphs_{PNTG}** and we have pushouts in both categories, we also have coproducts in both categories, which are constructed componentwise as disjoint union. The construction of \mathcal{E} - \mathcal{M} -factorizations for PTI nets and typed attributed graphs is given in Lemma 51 and in [88], respectively. Thus, **PTINet** and **AGraphs_{PNTG}** have the corresponding $\mathcal{E}' - \mathcal{M}$ pair factorizations. Moreover, $\mathcal{F}_{\text{PTIC}} : \mathbf{PTINet}|_{\mathcal{M}_1} \xrightarrow{\sim} \mathbf{SubAGraphs}_{\text{PNTG}}$ is a category equivalence according to Lemma 64. Finally, $I_{\text{PTI}} : \mathbf{SubAGraphs}_{\text{PNTG}} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ satisfies the properties listed in Definition 70 according to Lemma 65 and $(\mathbf{PTINet}, \mathcal{M}_1)$ has initial pushouts according to Fact 11. Thus, the restricted functor \mathcal{F}_{PTI} also satisfies the properties listed in Definition 70 by application of Theorem 29. \square

2 As already mentioned before, the source category of the restricted inclusion functor I_{PTI} is not \mathcal{M} -adhesive.

Part VI

RELATED WORK, CONCLUSION, AND FUTURE WORK

In this chapter, we firstly provide a short overview of our research domain by recalling past developments resulting in the foundations we built upon in this thesis. We focus thereby, in particular, on the algebraic double pushout (DPO) approach and \mathcal{M} -adhesive transformation systems. Secondly, we consider formalisms used in the running examples of this thesis (namely, typed attributed graphs, hypergraphs, and Petri nets with individual tokens (short PTI nets)) by discussing their applications for the modeling and analysis of structure and behavior of systems. We also discuss tools supporting the tasks of modeling and analysis in these settings. Finally, we relate our approach, on the one hand, to the framework of institutions and institution morphisms for the transfer of properties among two formalisms and, on the other hand, to triple graph grammars that are applied to relate models of different formalisms.

13.1 FOUNDATIONS FOR FORMAL TRANSFORMATIONS

In this thesis, we are concerned with the algebraic transformation approach [79, 76], which is based on pushout constructions on the involved structures modeling their gluing. The usage of pushouts allows for the definition of basic theoretical concepts and constructions as well as for the handling of many semantical properties of interest in the context of category theory. We are following the well-established DPO approach [79, 76] explained in detail e.g. in [88] rather than the single pushout (SPO) approach introduced in [265, 200, 199].

As a basis for our approach, we rely on the well-known categorical framework of \mathcal{M} -adhesive transformation systems [94, 88], which evolved over time from earlier predecessors. \mathcal{M} -adhesive transformation systems can be instantiated for many practically relevant formalisms such as a wide range of graphs and different kinds of Petri nets.

Initially, high-level replacement (HLR) systems were introduced in [80, 81] in order to provide a uniform framework for the formal transformation of graphs and high-level structures such as e.g. different kinds of Petri nets and algebraic specifications based on the DPO approach. The definition of the underlying categories of HLR systems is based on certain technical properties, called HLR axioms, which are sufficient to derive the main results of graph transformation and HLR systems theory such as Local Church-Rosser, Concurrency, Parallelism, Embedding, Extension, Completeness of Critical Pairs, and Local Confluence Theorems [88, 259]. These HLR axioms rely on the existence of pushouts and pullbacks and state their compatibility. The concept of \mathcal{M} -adhesive categories generalizes that of weak adhesive HLR [88], adhesive HLR [89], and adhesive categories [185]. In fact, the mentioned classes of categories are constructed similarly using HLR axioms, which are getting strictly more restrictive from left to right, thus adhesive categories are the most restrictive class and \mathcal{M} -adhesive categories are the least restrictive class of categories. In the spirit of this general approach, we provide our

results in this thesis by determining categorical characterizations of required properties to be satisfied by concrete functors and their source and target categories.

Application conditions have been introduced to limit the applicability of rules in graph transformation systems. Their expressiveness has been extended over time from negative application conditions [139] (short NACs) to the logic of nested application conditions [136] used in this thesis, which is equivalent to first-order logic on graphs [270]. While nested application conditions only state local properties in the sense of [119], there is ongoing work on extending the expressiveness of nested application conditions beyond first-order logic [138, 261, 186] to express properties such as connectedness by path of arbitrary length.

13.2 FORMALISMS AND TOOLS FOR MODELING AND ANALYSIS OF STRUCTURE AND BEHAVIOR

In this thesis, we apply our framework of \mathcal{M} -functors to the formalisms of hypergraph and PTI net transformation systems, which can be used in various application domains for modeling and analysis of system structure and behavior. In both cases we translate a source transformation system of the respective \mathcal{M} -functor (i.e., a hypergraph transformation system resp. a PTI net transformation system) into the corresponding typed attributed graph transformation system. In this section, we briefly discuss these three mentioned formalisms regarding their role for the modeling and analysis of system structure and behavior. Furthermore, we provide for all three formalisms an overview of the available tool support for modeling and analysis discussing the respective limitations for the applicability to our analysis problems.

13.2.1 *Graphs*

We consider the formalism of graphs w.r.t. its suitability for the modeling of system structure in Subsubsection 13.2.1.1 as well as for the modeling and analysis of dynamic system behavior by means of graph transformation systems in Subsubsection 13.2.1.2. Moreover, we consider in Subsubsection 13.2.1.3 the available tool support for both, modeling and analysis.

13.2.1.1 *Modeling Structure using Graphs*

Various kinds of graphs are used to represent the structure of systems in a broad range of domains. For example, they are used for the modeling of inter-networking systems [295], the development of knowledge-based design tools [282], and the abstract syntax of visual models such as e.g. UML class diagrams [122]. Graphs are a favorable choice due to their flexibility, due to their capability of representing relations adequately, and due to their visual notation, which enables intuitive modeling. Also, local navigation in a graph can be more efficient compared to relational models as demonstrated by graph databases [233].

Typed attributed graphs have been considered in the past in various works using different formalizations. In [202] the authors represent the graph part of a typed attributed graph as an algebra that extends the given data type algebra. In [258] typed

attributed graphs are defined using labeled graphs and transformed by application of rule schemata dealing with calculations on labels. A typed attributed graph with node and edge attribution is seen as a pair consisting of a graph and a data algebra, whose values are contained as nodes in the graph in [88] whereas only node attribution is allowed in [144]. Finally, the so-called *symbolic graphs* [240, 241] allow for the separation of the graph and the algebra parts of typed attributed graphs. In this thesis, we use typed attributed graphs based on E-graphs following the approach in [88] that supports the node and the edge attribution.

13.2.1.2 Modeling and Analyzing Behavior of Graph Transformation Systems

Dynamic systems that exhibit or perform successive state changes can be described using graph transformation systems. In graph transformation systems, the possible behavior is defined in terms of a set of rules and their application describing admissible state modifications. These rules, together with an initial graph from which the system is initiated, determine a graph grammar, which is conceptually quite similar to Chomsky grammars for formal languages [271, 72, 73, 85].

Various kinds of transformation rules, rule application procedures, and extensions of graphs exist. The *node-label controlled replacement approach* [158, 160, 159, 101], in which transformation steps replace single nodes by replacement graphs, and the *hyperedge replacement approach*, which is designed for hypergraph transformation systems as discussed in more detail in the next section, are two examples of graph transformation approaches. However, as already mentioned before, we are concerned with the *algebraic graph transformation approach*, which allows for the handling of graph transformations in the context of category theory. The DPO approach [79, 76] and the SPO approach [265, 200, 199] are the two mostly used variants of the algebraic graph transformation approach. The fundamental difference between both approaches is that the DPO approach makes use of two pushouts and total morphisms to define a transformation step, whereas the SPO approach uses a single pushout with a partial rule morphism for this purpose. For the detailed comparison of both approaches consult e.g. [83]. In this thesis, we employ the algebraic DPO approach as a foundation for our work.

Single process algorithms as well as distributed multi-process algorithms can be encoded out of the box as graph transformation systems including the involved data structures using application specific algebraic structures [88] such as in [137, 283, 258]. Other application areas of graph transformation systems are, amongst others, formal language theory [179, 133], process algebras and mobile systems [24], database design [14, 115], logical and functional programming [48, 284], compiler construction [151], model transformation [99, 88], model versioning [165], requirements engineering [141], parsing of visual languages [25], automated transfer of visual models into code or into different semantical domains [102, 303, 183], software engineering [144, 88], and implementation of visual modeling techniques such as UML [236].

While various semantical properties of transformation systems are to be analyzed in these settings, we focus in this thesis, in particular, on the analysis of behavioral equivalence, local confluence (including the analysis of parallel and sequential independence of transformations), termination as well as on the thereof resulting analysis of confluence and functional behavior. In the following, we briefly summarize established analysis techniques for our semantical properties of interest.

For the analysis of behavioral equivalence based on bisimulations, the framework of borrowed contexts from [262] (also see [263, 264] for earlier contributions) provides re-

sults, constructions, and algorithmic procedures for deriving behavioral congruences. Moreover, in [242], the adaptation of the before mentioned approach to symbolic graphs is introduced to improve the handling of attributes in graph transformation systems.

There are two standard techniques to analyze local confluence of graph transformation systems [88]. Firstly, the parallel independence of each two arbitrary transformation steps of a system is a sufficient condition for local confluence of a system according to the Local Church-Rosser Theorem. Secondly, the verification of local confluence can proceed by computing all critical pairs (as supported by e.g. AGG) and by then analyzing these critical pairs for strict confluence as discussed in detail in Subsubsection 2.2.2.4.

The problem of deciding termination for arbitrary graph transformation systems is undecidable due to their expressiveness. Indeed, several general techniques to support the process for finding a termination proof (or a proof for non-termination) for different kinds of systems have been developed in the past in [57, 8, 25, 86, 26, 28, 21, 22, 29, 30, 257, 194]. These techniques vary in the required human involvement, i.e., in the degree of their automation.

Finally, local confluence and termination analysis are sufficient according to [88] for the verification of confluence and functional behavior of graph transformation systems.

For more details on the analysis techniques for our semantical properties of interest consult Subsection 2.2.2.

13.2.1.3 Tools for Modeling and Analysis of Graph Transformation Systems

There exist various tools supporting modeling, simulation, and analysis of graph transformation systems. In the following, we discuss a spectrum of such tools focusing on their features. Note that we consider only those tools that offer analysis capabilities.

- AGG (Attributed Graph Grammar System) [292, 104, 287] is a general-purpose tool supporting the specification of algebraic graph transformation systems based on typed attributed graphs with node type inheritance, DPO rules, nested application conditions, and graph constraints [272]. To the best of our knowledge, AGG is the only tool directly implementing the existing theoretical constructions and analysis results for the modeling and analysis of algebraic graph transformation systems with NACs and PACs. For the modeling of graph transformation systems AGG provides several graphical editors, an interpreter for system simulation, and a debugger for system verification. For the analysis of graph transformation systems AGG provides support for critical pair and termination analysis, conflict and dependency detection between transformations, graph parsing as well as consistency and rule applicability checks on graphs [272]. The critical pair and dependency analysis is offered through a graphical user interface that allows for the inspection of the computed critical pairs and dependencies, respectively. The termination analysis for typed graph transformation systems with injective rules, injective match morphisms, and injective NACs using layers for rules and graph types is implemented by checking the so-called *layering conditions*, which are sufficient criteria for termination. Moreover, AGG provides *Java* code for the execution of graph transformations that can be used by any graphical or non-graphical *Java* application.
- *Henshin* [69, 6, 187] is a model transformation and high-level graph rewriting tool including as features specification capabilities for graph transformation systems in

a graphical editor, an engine for critical pair computation, an engine for state space generation, certain control structures on the set of rules, and a model checking tool [157, 6]. Although, *Henshin* is, similarly to AGG, another general-purpose tool for the modeling and analysis of graph transformation systems, compared to AGG, *Henshin* employs for critical pair analysis only transformation rules that may not have NACs and PACs as of now [187].

- *VERIGRAPH* [305] is a verification tool initially developed to support second order rewriting and analysis. *VERIGRAPH* focuses on the DPO approach for typed graphs and implements in the theory proximal manner critical pair/sequence analysis, concurrent rules, second order conflicts, interlevel conflicts, and rule transformations for typed graph transformation systems. Unfortunately, *VERIGRAPH* does not support the handling of attributes as well as transformation systems containing rules with any kind of application conditions as of now.
- *SyGrAV* (Symbolic Graph Analysis and Verification) [53, 290] supports static analysis (including critical pair analysis with the resolution of found conflicts) of typed attributed graph transformation systems using the approach of symbolic graphs from [239]. However, as stated in [53] transformation rules used for critical pair analysis may not have application conditions as of now.
- *GROOVE* (Graphs for Object-Oriented Verification) [298, 121] is a general-purpose graph transformation tool based on typed labeled graphs. Its core functionality is the state space generation. Based on the generated state space, the satisfaction of LTL and CTL properties can be analyzed [157]. Other useful features supported by *GROOVE* are the encoding of node attributes and the control language in combination with complex, recursive, and nested rule applications that also support NACs and regular expressions identifying paths of edges [121].
- *Augur2* [296, 176, 177, 13] supports the analysis of typed attributed graph transformation systems using over- and under-approximations of the state space. Under-approximations permit the analysis of the existence of certain sequences of direct transformations whereas over-approximations allow for the analysis of non-existence of such sequences.
- *Maude* [44, 45] is a language and a tool supporting computations based on equational and rewriting logic. It implements different kinds of formalisms using a powerful algebraic language. Rewriting rules in *Maude* directly support reconfigurations based on conditional term rewriting [34]. *Maude* offers a possibility for termination checks in the Knuth-Bendix completion tool [63], critical pair analysis using a Church-Rosser checker [65, 64], and the analysis of temporal graph properties [302]. However, to the best of our knowledge, the Church-Rosser checker, which is only applicable to terminating order-sorted equational specifications [64, 65], does not support critical pair analysis combined with nested application conditions [65, 45] as considered in this thesis and, in addition, is incomplete in the sense that it cannot determine a definite answer in each case [64, 65].

In conclusion, until now only AGG supports the critical pair analysis for typed attributed graph transformation systems with NACs and PACs. Other mentioned tools

support different analysis problems (e.g. *GROOVE* and *Augur2*), conceptually different input formalisms (e.g. *Maude*) or less expressive graph transformation formalisms (e.g. *Henshin*, *VERIGRAPH*, and *SyGrAV*) as of now. However, *Henshin*, *VERIGRAPH*, and *SyGrAV* are currently maintained more actively and may contain better custom support for critical pair analysis in the future.

Besides incomplete symbolic techniques that are required for systems with infinite state spaces, tools such as *Henshin* can be used to decide (non-)termination for graph transformation systems with finite state spaces. Moreover, the tool *GreZ*, described in [29], supports an approach for termination analysis of graph transformation systems without any kind of application conditions and without a distinguished initial graph. However, we are not aware of tools besides *AGG*, *GreZ*, and *Maude* that have a certain built-in support for termination analysis of any kind of \mathcal{M} -adhesive transformation systems or their encodings.

Finally, considering the analysis of bisimilarity, to the best of our knowledge, there are no tools for any kind of \mathcal{M} -adhesive transformation systems as of now. On the one hand, efficient algorithms for checking bisimilarity for arbitrary finite state systems have been developed such as e.g. in [59], but these algorithms do not exploit domain knowledge and are not capable of handling infinite state spaces. On the other hand, tools such as *ABC* (Advanced Bisimulation Checker) [273] are domain specific (*ABC* handles the open bisimilarity in the context of the π -calculus) and are therefore not applicable to \mathcal{M} -adhesive transformation systems. The development of efficient symbolic algorithms and their tool support for checking our notions of bisimilarity is therefore left for future work.

13.2.2 Hypergraphs

While the formalism of graphs is versatile, additional language constructs are sometimes desirable to express structure and rules more concisely. The transformation of such high-level structures was introduced in [80, 81] using HLR systems as a common foundation. We discuss with hypergraphs and Petri nets two kinds of high-level structures in more detail in this and the next subsection.

As before, we consider in Subsubsection 13.2.2.1 the usage of hypergraphs for the modeling of system structure as well as in Subsubsection 13.2.2.2 the modeling and analysis of dynamic system behavior by means of hypergraph transformation systems. Furthermore, we consider in Subsubsection 13.2.2.3 the available tool support for modeling and analysis of hypergraph transformation systems.

13.2.2.1 Modeling Structure using Hypergraphs

The formalism of hypergraphs is, compared to graphs, still versatile and is also widely used for the modeling of various kinds of systems. Its main strength, compared to graphs, is its built-in capability of succinctly representing relationships between an unbounded number of nodes, i.e., an edge (also called a *hyperedge*) in a hypergraph connects an unbounded number of source nodes with an unbounded number of target nodes. In this sense, hypergraphs where each edge connects precisely one source node with one target node can be understood as ordinary graphs.

Hypergraphs are used in various fields of science such as mathematics [9, 114, 120, 253], biology [168, 113], and chemistry [27, 178, 113, 120]. Moreover, in computer science

they have been used, for example, in the domains of distributed systems [271, 12], inter-networking systems [149, 148, 295], access control policies [174, 175, 173], Online Social Networks (OSN) [226, 286, 225], services in service-oriented architectures for business processes including their mutual dependencies as well as their resource and capacity requirements [37, 40], declustering problems for the implementation of I/O parallelization algorithms in the context of parallel databases and high performance systems [196], designs in the context of knowledge-based design tools [131], analysis of relational databases [120], knowledge representation systems (including e.g. the Semantic Web [16] standards OWL-DL [52] and RDF [306]) [197, 166], parsing of graph languages [61], and diagram representation and design [222, 220, 221].

13.2.2.2 Modeling and Analyzing Behavior of Hypergraph Transformation Systems

Some of the applications mentioned above pertain to systems given by hypergraphs with dynamic behavior. To formalize such systems, hypergraph transformation systems based on the algebraic DPO approach can be employed. However, the two other established approaches of hyperedge replacement and synchronized hyperedge replacement, which are discussed subsequently, are also capable to determine dynamic behavior of hypergraph-based systems.

The *hyperedge replacement* approach [109, 250, 181, 134, 190, 60, 15, 49, 50, 100] allows for the replacement of a single hyperedge by another hypergraph. According to [181], hyperedge replacement systems can be seen as graphical context-free Chomsky grammars.

The *Synchronized Hyperedge Replacement* (SHR) approach was introduced for the modeling of distributed systems and software architectures dealing with different aspects of network applications [149, 111, 295, 110, 189, 112, 39, 54]. The SHR approach combines the ability to express various forms of synchronization and communication futures (typical for process calculi) with a reasonable visual representation of system topology (typical for graph models). It has been used to encode process calculi such as the π -calculus [219, 147], the CSP process algebra [150], the Ambient calculus [295, 38], and the Fusion calculus [188]. Moreover, in [117, 118] hypergraphs are also used as a fully abstract denotational model of the π -calculus for the analysis of spatial logic satisfaction. Finally, in [163] a process calculus for the stochastic modeling and simulation of biochemical reactions is introduced. It extends the κ -calculus based on the idea of hyperedge replacement.

Hypergraph transformation systems have been used to model machine learning processes [279], to formalize Architectural Design Rewriting (ADR) [149, 34, 33], to encode architectural refactorings [132] for the architecture description language COOL [203] using the capability of hypergraphs to reflect hierarchical concepts, and to evaluate functional expressions [253] where a function with n arguments is modeled by a hyperedge with one source node and n target nodes.

Hypergraph transformation systems have been analyzed for the satisfaction of path properties in [11, 13] using state space approximations, for confluence in [253] where a sufficient condition for local confluence based on critical pairs has been given, and for termination in [257] where critical pair analysis is used to determine that the composition of two terminating hypergraph transformation systems is terminating. Further analysis techniques for our semantical properties of interest for hypergraph transformation systems rely on the respective approaches for graph transformation systems discussed

in Subsubsection 13.2.1.2 and are in general applied on an informal level by manually encoding the analysis problems in the domain of graph transformation systems.

13.2.2.3 Tools for Modeling and Analysis of Hypergraph Transformation Systems

The tools *AGG*, *Henshin*, *SyGrAV*, *GROOVE*, *Augur2*, and *Maude* considered for graph transformation systems can also be used in principle for the modeling and analysis of hypergraph transformation systems. However, because these tools do not support hypergraph transformation systems directly, a formally verified translation of a hypergraph transformation system into the input language of these tools is required in each case. In this thesis, we have defined and verified such a suitable translation from hypergraph transformation systems into typed attributed graph transformation systems by means of our concrete \mathcal{M} -functor \mathcal{F}_{HG} .

Since *AGG* is implemented in close relationship to the used theoretical foundations and is powerful enough for our analysis reasons, we see *AGG* as a tool that can be most easily extended in the future by the formally verified translation of the formalisms of interest. This means that *AGG* can be extended to analyze local confluence and termination of hypergraph transformation systems using an implementation of \mathcal{F}_{HG} to translate a given hypergraph transformation system into the corresponding graph transformation system. However, since *AGG* would still internally operate on typed attributed graphs when being provided with a translated hypergraph transformation system, we must support in an additional feature, following our formal approach, that during the verification process *AGG* executes the analysis steps only for typed attributed graphs that are \mathcal{F}_{HG} -images.

With its current capabilities *AGG* has been used in [35] for the modeling and verification of self repairing systems given as typed hypergraph grammars where a manual encoding of a hypergraph transformation system has been used without its formal verification.

Maude is another example of a tool in which hypergraph transformation systems can be analyzed for local confluence and termination by encoding them using (multi-)sets and conditional (multiset) rewriting rules [31, 34].

All in all, the available tool support is not satisfactory since no direct support for the analysis of hypergraph transformation systems exists as of now. The existing tools only support the analysis of hypergraph transformation systems via ad hoc encodings that lack formal treatment.

13.2.3 Petri nets

The formalism of Petri nets for the modeling and analysis of concurrent and distributed systems is different from graphs and hypergraphs discussed before since it comes along with its own semantics (the firing behavior). Subsequently, we first focus in Subsubsection 13.2.3.1 on the usage of Petri nets to model the concurrent system behavior. Note that in this thesis, we are not interested in the analysis of the domain specific Petri net properties. Afterwards, in Subsubsection 13.2.3.2, we discuss the rule-based reconfiguration of Petri nets for the modeling and analysis of dynamic system behavior including the special case of rule-based transition firing for PTI net transformation systems. Finally, we consider in Subsubsection 13.2.3.3 the available tool support for modeling and analysis of Petri net transformation systems.

13.2.3.1 Modeling Concurrent Behavior using Petri Nets

Petri nets [252] are a formal specification language useful for modeling concurrent, distributed, parallel, asynchronous, non-deterministic, and stochastic systems [268] supporting powerful techniques for qualitative and quantitative analysis of domain specific properties [232]. The formalism of Petri nets combines a well-founded mathematical theory with a graphical representation of the dynamic system behavior by firing transitions. A strength of the Petri net approach is its intuitive visual notation and the variety of implemented analysis techniques considering e.g. reachability, liveness, and safety properties.

There are many examples for the application of different kinds of Petri nets to model a variety of dynamic event-driven and concurrent systems [308] such as computer networks [205], communication systems [214, 307], real-time computing systems [294], workflows [299, 195], and logistic networks [301].

In [216], Meseguer and Montanari represented Petri nets as graphs equipped with operations for the composition of transitions. They introduced categories for Petri nets with and without initial markings and functors expressing duality and invariants. Their constructions provide a formal basis for expressing concurrency in terms of algebraic structures over graphs and categories. Based on categorical Petri nets, in [62] Petri nets are related to automata with concurrency relations by establishing a correspondence between the associated categories.

13.2.3.2 Modeling and Analyzing Behavior of Petri Net Transformation Systems

The formalism of Reconfigurable Petri Nets [92, 154] generalized to the notion of Petri net transformation systems [80, 81, 90, 246] allows to combine formal modeling of dynamic systems and controlled model adaptation. The main idea is the stepwise development of Place/Transition nets (short P/T nets) by applying net transformation rules [90, 92, 260]. This approach makes Petri nets more expressive and allows additionally to the firing behavior for a formal description and analysis of structural changes [90, 92, 260].

The application areas for the formalism of Petri net transformation systems comprise e.g. medical information systems [103], train control systems [247], logistics [90], emergency scenarios [92], reconfigurable manufacturing systems [164], complex dynamic structures [130], and component technology [243, 90].

The firing behavior of Petri nets was originally expressed using graph transformation rules in [180]. To allow the modification of markings using transformation rules, it is required that the tokens of the marking are included in the Petri net. This is achieved in this thesis by using the formalism of PTI nets [224] (see [32] for a comparison with the collective token approach) where a set of tokens is added to the Petri net in addition to places and transitions and each token is mapped to one of the places. The practicability of the PTI net transformation approach is shown in [105, 223, 116] introducing case studies on applications to communication spaces and communication platforms.

Different notions of individuality have been introduced in [266, 300] where tokens are equipped with additional information influencing the firing of transitions. This additional information is used to distinguish tokens, to store relevant data, or to store information about the history of a token. Such tokens do not amount any longer to indistinguishable black tokens and are therefore called *high-level tokens*. The concept of

high-level tokens with an additional inner structure is orthogonal to the PTI net concept since PTI nets only extend the standard Petri net formalism by explicit inclusion of a marking into the structure of the net. These two orthogonal approaches are combined in *Algebraic High-Level nets with individual tokens* (short AHLI nets) where tokens are data elements from a given algebra.

Reconfigurable Petri nets have been analyzed in [245] for the satisfaction of LTL properties using the tool *ReConNet* [248], which translates a given reconfigurable Petri net into a *Maude* module with a bisimilar transition system. The resulting *Maude* module can then be checked equivalently to the original reconfigurable Petri net for the satisfaction of LTL properties using the *LTLR* model checker of *Maude* [245]. However, analysis techniques for our semantical properties of interest for Petri net transformation systems rely on the respective approaches for graph transformation systems discussed in Subsubsection 13.2.1.2 and are in general applied on an informal level by manually encoding the analysis problems in the domain of graph transformation systems.

13.2.3.3 Tools for Modeling and Analysis of Petri Net Transformation Systems

As for hypergraph transformation systems, the tools *AGG*, *Henshin*, *SyGrAV*, *GROOVE*, *Augur2*, and *Maude* can be used in principle to model and analyze Petri net transformation systems by encoding them along the lines of the restricted \mathcal{M} -functor \mathcal{F}_{PTI} provided in this thesis (and in the case of *Maude* by using an appropriate domain specific encoding for Petri net transformation systems).

Considering tokens as individuals makes the formalism of PTI nets similar to that of typed attributed graphs giving a possibility for formal-based modeling and analysis of PTI net transformation systems using existing graph transformation tools such as *AGG* [206, 285, 20]. Subsequently, we focus on further tools that are more specific for Petri net transformation systems.

The *RON-tool* (Reconfigurable Object Net Tool) [291] can be used for the modeling, simulation, and analysis of a special kind of high-level nets called *reconfigurable object nets*. Reconfigurable object nets [152, 19, 18, 106] are a simplified kind of higher-order nets with two kinds of tokens, two kinds of places, and a complex firing behavior. The *RON-tool* employs *AGG* [292] for the simulation [20] as well as for the critical pair and independence analysis [18] of P/T net transformation systems by using an ad hoc translation of P/T net transformation systems into the corresponding typed attributed graph transformation systems. In fact, the formalization of this translation was the initial motivation to develop the formal framework of this thesis. However, for continuing applicability, a maintained software is desirable, implementing the translation of P/T nets into the corresponding typed attributed graphs precisely as well as ensuring correctness, usability, and full automation of the analysis process. Furthermore, there are several restrictions on the kind of P/T net transformation systems that can be analyzed using the *RON-tool*. The current version of the *RON-tool* supports the local confluence analysis of P/T net transformation systems (following the collective token approach) containing rules without any kind of application conditions or rules with NACs only, while no analysis support is available for rules containing PACs or rules with nested application conditions. For this reason, we do not use the *RON-tool* for analysis purposes in this thesis and propose the development of suitable tool support.

ReConNet (Reconfigurable Net) [248] is a tool that allows for the modeling, simulation, and analysis of reconfigurable Petri nets. It uses as a technical basis for reconfigurable Petri nets the so-called *decorated Petri nets*, i.e., P/T nets extended by additional annotations like capacities, names for places and transitions as well as transition labels changeable by firing the corresponding transitions [248]. The net transformation part of the tool is based on the cospan approach (in which rules have the form $L \rightarrow K \leftarrow R$ [248]), which is shown to be equivalent to the DPO approach in [93]. While *ReConNet* allows, as discussed above, for the analysis of the satisfaction of LTL properties by translating the reconfigurable Petri net at hand into a *Maude* module, on which the *LTLR* model checker of *Maude* is then applied [245], it does only support net transformation rules with NACs as of now [244].

MCRNet (Marked-Controlled Reconfigurable Nets) [198] is a tool for the modeling, simulation, and analysis of concurrent systems specified by so-called *marked-controlled reconfigurable nets* (short MCRNs). A configuration of a concurrent system is then represented by a Petri net, while configuration changes are defined by Petri net transformation rules. The main intention of *MCRNet* is to translate an MCRN into an equivalent Petri net (containing all possible configurations of the considered MCRN) to be able to analyze subsequently the structure as well as the dynamic behavior of the obtained Petri net using the Petri net tool *PIPE2* (Platform Independent Petri Net Editor) [56].

Other tool environments like *PEP* (Programming Environment based on Petri Nets) [297], *INA* (Integrated Net Analyzer) [156], *PNK* (Petri Net Kernel) [167, 309], *Renew* (The Reference Net Workshop) [293], *Charlie* [289, 145], *Tina* (Time Petri Net Analyzer) [184, 17], *EZPetri* [2, 5], *JARP* [55], *POSEIDON* [191], *PIPE2* [56], *PNTTools* (Petri Net Tools) [42], *Petri Net Toolbox* [288], and many others are indeed capable to model, simulate, and analyze different kinds of Petri nets but they do not handle the net reconfiguration by means of algebraic graph transformation.

We conclude that the available tools do not *directly* support the analysis of the semantical properties we are interested in for Petri net transformation systems. However, we expect that the translation of reconfigurable Petri nets into *Maude* modules using *ReConNet* along the lines of [245] can be adapted to different kinds of Petri net transformation systems. Based on such adapted translations, the analysis of properties such as local confluence and termination could be enabled using the *Maude* specific tools keeping their known limitation in mind. Moreover, most of the graph transformation specific tools discussed in Subsubsection 13.2.1.3 as well as the *RON-tool* can be used for modeling and limited analysis of the desired properties but a formal foundation for the required encodings of Petri net transformation systems into the respective tool languages is called for. Finally, the tool *MCRNet* focuses on Petri net transformation systems but cannot be used for the analysis of the intended semantical properties.

13.3 EXISTING FRAMEWORKS FOR TRANSLATION OF DIFFERENT FORMALISMS

Besides the functorial transfer of objects and morphisms between categories as well as the transfer of property satisfaction pursued in this thesis, there are further frameworks designed for similar purposes. We consider in Subsection 13.3.1 institutions and institution morphisms that are tailored for the relation of logical systems and that, as underlined by related work, can also be used in principle to relate transformation systems as in this the-

sis. However, we believe that the generality of the framework provides too little support for solving the actual research problem of this thesis. Furthermore, we briefly consider in Subsection 13.3.2 the approach of triple graph grammars that allows for the synchronization of different models. In relation to this thesis, triple graph grammars can be used to implement a connection between objects of two transformation systems. However, the approach of triple graph grammars has, to the best of our knowledge, not been extended to verify the transfer of the satisfaction of semantical properties as in this thesis. Hence, triple graph grammars merely implement translations between transformation systems similarly to the applications of \mathcal{M} -functors in this thesis.

13.3.1 Institutions

Institutions were introduced in [36, 126, 127] to formalize the concept of a *logical system* to allow to relate and compare logics (see e.g. [182, 228, 227, 129, 231]) and to inherit results from one logic to another through a relationship expressed by an institution morphism [127, 41]. Intuitively, an institution consists of a collection of signatures Σ together with the corresponding signature morphisms, a set of Σ -sentences for each signature Σ , a set of Σ -models for each signature Σ , and a Σ -satisfaction relation between Σ -models and Σ -sentences. For example, algebraic signatures and Σ -algebras can be used to define an institution (see e.g. [276, 229]) that has the usual many sorted signatures Σ with the usual signature morphisms, Σ -sentences in the form of equations consisting of two terms, Σ -models given by the usual Σ -algebras, and the usual Σ -satisfaction relation where a Σ -algebra satisfies an equation if both contained terms are evaluated to the same element in the carrier set. The important property of an institution is then that changing Σ -models and Σ -sentences using a signature morphism is compatible with the Σ -satisfaction relations of the institution. This compatibility implies that theorem provers can be used interchangeably, once the required institutions are in place.

The framework of institutions has been used for different purposes. In [274, 275] observational equivalence is considered in the context of algebraic specifications (i.e., it has been checked whether two Σ -algebras evaluate their terms in the same manner w.r.t. a set of tests given in the form of Σ -sentences by not only relying on the ultimately obtained element of the carrier set). In these works, an institution is used again for relating different Σ -algebras and universal quantification is added to the above described compatibility definition allowing for the usage of free variables in Σ -sentences. In [230, 238] the CSP process algebra has been considered and the absence of deadlocks has been verified for concrete processes by using structuring techniques introduced by use of institutions. In [171, 172, 170] different UML diagram types (such as UML state machines, UML class diagrams, and UML sequence diagrams [236]) are related using institutions.

In [41] early results on *borrowing* theorems have been developed that are similar to the formal translation of models and analysis results in this thesis. In this work, institution morphisms are employed to relate institutions for different logics (which must be linked by adjoint functors) to derive theorem prover support for one logic from the theorem prover support of the other logic. This approach follows conceptual considerations in [215] on the formal interoperability of formalizations of a system also allowing for the usage of tools supporting different of these formalizations. An attempt to obtain a unified framework for institution morphisms has been presented in [128]. Moreover,

besides the relation of logics, also other formalisms such as the OWL ontology language and the Z specification language have been related using institution morphisms in [204].

For our purposes, we can assume that Σ -sentences of an institution are used to express properties such as the existence of direct transformations and transformation sequences (leading to the behavioral equivalence of the underlying systems) as well as their independence, (local) confluence, termination, and functional behavior, which are then to be defined (in terms of the corresponding Σ -satisfaction relations) for hypergraph, PTI net, and typed attributed graph transformation systems in a compatible way. This assumption also implies that the definition of the institution morphisms corresponds to our (restricted) \mathcal{M} -functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} . However, as pointed out in [127], it can be already tedious to show that something is an institution in the sense of the satisfaction of the compatibility conditions. While we believe that the framework of institutions can be used to characterize the problem tackled in this thesis from a different point of view, it adds, in our opinion, too little support for its formal-based solution. Even though the tool *Maude*, widely used for the analysis of systems specified in terms of institutions, enables the local confluence and termination analysis for single systems, there is no formal theory supporting the sound transfer of analysis results between \mathcal{M} -adhesive transformation systems or concrete instances thereof as considered in this thesis so far. However, institutions and institution morphisms may be seen as an umbrella framework, in which we can state our main problem of translating analysis results among formalisms but existing institution technology seems to be not directly applicable to our problem.

13.3.2 Triple Graph Grammars

Triple graph grammars have been developed for the synchronization of models of different formalisms in [281]. Adaptations on one of the related models (initiated by users or other software components) are then propagated to the other model by application of rules of the employed triple graph grammar allowing to adopt the changes. This synchronization between the models is intended to restore their consistency to ensure that the models provide a consistent perspective on the considered system.

The research in the field of triple graph grammars mainly focuses on the design and verification of the process of such resynchronization that has to be correct w.r.t. the triple graph grammar that serves as a specification [4, 146, 3, 192]. Further scenarios include e.g. physical distribution or concurrent modification of more than one model, which are both rendering the resynchronization task for the consistency restoration more complex [192]. Moreover, suitable data structures that keep track of corresponding elements are employed to minimize the computational effort for consistency restoration [4].

Based on this general approach, triple graph grammars have been also investigated in [68] for relating two transformation systems whose models are, as in this thesis, related by bisimulations that are preserved by the steps of the transformation systems. However, for comparison, the used triple graph grammars merely correspond to the application of our \mathcal{M} -functors and the bisimulations are checked rather than enforced as in this thesis.

CONCLUSION

In this thesis, we have introduced an abstract framework relating different formalisms with respect to semantical properties of interest. The relationship defined within our framework allows for the transfer of semantical properties between the considered concrete formalisms enabling the usage of analysis tools available for the target formalism also for the source formalism as visualized in [Figure 1](#) in the introduction. The class of formalisms covered by our framework is given by \mathcal{M} -adhesive transformation systems, which are based on \mathcal{M} -adhesive categories. For \mathcal{M} -adhesive categories certain technical properties are assumed and many technical notions are defined, which are relevant for the modeling of system transformation and adaptation. We have investigated various semantical properties such as behavioral equivalence, (local) confluence, termination, functional behavior as well as parallel and sequential independence of transformations. In our abstract framework we have established a formal relationship between two \mathcal{M} -adhesive transformation systems by introducing the so-called \mathcal{M} -functors. Within the framework we have treated first transformation systems *without* nested application conditions and have extended the framework subsequently to the case of transformation systems *with* nested application conditions. This extension, compared to the case regarding transformation systems *without* nested application conditions, greatly simplifies the modeling of transformation systems in practice but requires further sufficient properties to be satisfied by the \mathcal{M} -adhesive categories and \mathcal{M} -functors involved. Moreover, we have considered two instantiations of our abstract framework, namely, we have related both, hypergraph transformation systems and Petri net transformation systems with individual tokens, with typed attributed graph transformation systems. For these instantiations we have defined concrete \mathcal{M} -functors from the \mathcal{M} -adhesive category of the source transformation system to the corresponding \mathcal{M} -adhesive category of the target transformation system. In both cases we have ensured a sound instantiation by verifying the sufficient properties stated by our abstract framework for the involved \mathcal{M} -adhesive categories and the defined \mathcal{M} -functors. We have chosen typed attributed graph transformation systems as a target formalism for both instantiations because e.g. the well-established tool AGG, operating on typed attributed graph transformation systems, provides, besides modeling and simulation capabilities, also analysis capabilities such as, in particular, the critical pair analysis, which is the first step towards confluence analysis.

In [Figure 81](#) we present an overview of the main results, which are included in the central Parts [ii–v](#) of this thesis. The tree depicted in the center with “General Approach” as the root contains the results of Part [ii](#) where we have introduced our main theoretical approach consisting of Theorems [1–13](#) for the functorial transfer of the semantical properties P_1 – P_6 given in the table in the bottom of the picture. The two rows below the tree contain the application results from the two consecutive Parts [iii](#) and [iv](#) where we have instantiated all theoretical results of Part [ii](#) for the concrete settings of hypergraph and PTI net transformation systems in Theorems [14–20](#) and Theorems [21–27](#), respectively. Finally, the arrow leading to “General Approach” from the top adds to the overview

the functor decomposition strategy for the requirement verification introduced in Theorems 28 and 29 in Part v allowing for an alternative proof strategy for deriving the theorem instantiations for concrete (restricted) functors at hand. The application of this functor decomposition strategy to the two concrete functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} is then shown in Theorems 30 and 31, respectively.

In Figure 82 we provide an overview of the requirements (given by the previously mentioned sufficient properties) that need to be verified for a (restricted) functor at hand to instantiate the theorems of our abstract framework given in Figure 81. In fact, we focus in the first table (see the upper left part of Figure 82) on the semantical properties P_1 – P_6 (given in the second table in the lower left part of Figure 82) for transformation systems without or with nested application conditions and provide for functors and restricted functors a list of requirements to be verified. These requirements are listed in the third and fourth table depicted in the right part of Figure 82.

The theoretical results given in Theorems 1–13 in Part ii characterize the requirements needed to transfer the semantical properties P_1 – P_6 between \mathcal{M} -adhesive transformation systems using \mathcal{M} -functors. Subsequently, we first recall the main results from the first and the second branch of the tree in Figure 81 considering the transfer of semantical properties P_1 – P_3 , whose analysis in our case is dependent on whether the rules of the involved transformation systems are equipped with nested application conditions. Afterwards, we review our results on the semantical properties P_4 – P_6 where we did not provide separate theorems for the two alternate cases.

Firstly, in Theorems 1–3 we have shown the translation and creation of transformation steps and their sequences (with general or \mathcal{M} -match morphisms) between the corresponding \mathcal{M} -adhesive transformation systems (property P_1) using \mathcal{M} -functors for the case of rules without or with nested application conditions. That is, we have provided sufficient properties ensuring that an \mathcal{M} -functor properly relates each transformation step in the source transformation system with a corresponding transformation step in the target transformation system guaranteeing behavioral equivalence of the functor-related parts of both transformation systems. Secondly, in Theorems 7 and 10 we have dealt with the transfer of the parallel and sequential independence of transformation steps (property P_2), which is a sufficient condition for the absence of conflicting rule applications and, hence, also for local confluence of a transformation system. Thirdly, in Theorems 6, 8, 9, and 11 we have established the transfer of the local confluence property (property P_3) enforcing that the order of rule applications is not relevant in the sense that alternative choices of rules to be applied can be joined in a reasonable way. Fourthly, to capture the notion of behavioral equivalence of the functor-related parts of the source and the target transformation systems as well as to additionally provide a different perspective on the relation of behavior of transformation systems, we have considered in Theorems 4 and 5 the \mathcal{F} -bisimulations between functor-related objects of the underlying source and target categories of an \mathcal{M} -functor as well as the \mathcal{F} -transfer of bisimulations between objects of the source category into the corresponding bisimulations between objects of the target category (property P_4), respectively. Fifthly, in Theorem 12 we have transferred termination/non-termination of transformation sequences (property P_5), i.e., the absence/existence of infinite sequences of transformation steps. While termination is an important property in general, it is also crucial for the analysis of confluence and functional behavior. That is why we have reused our results on termination and local

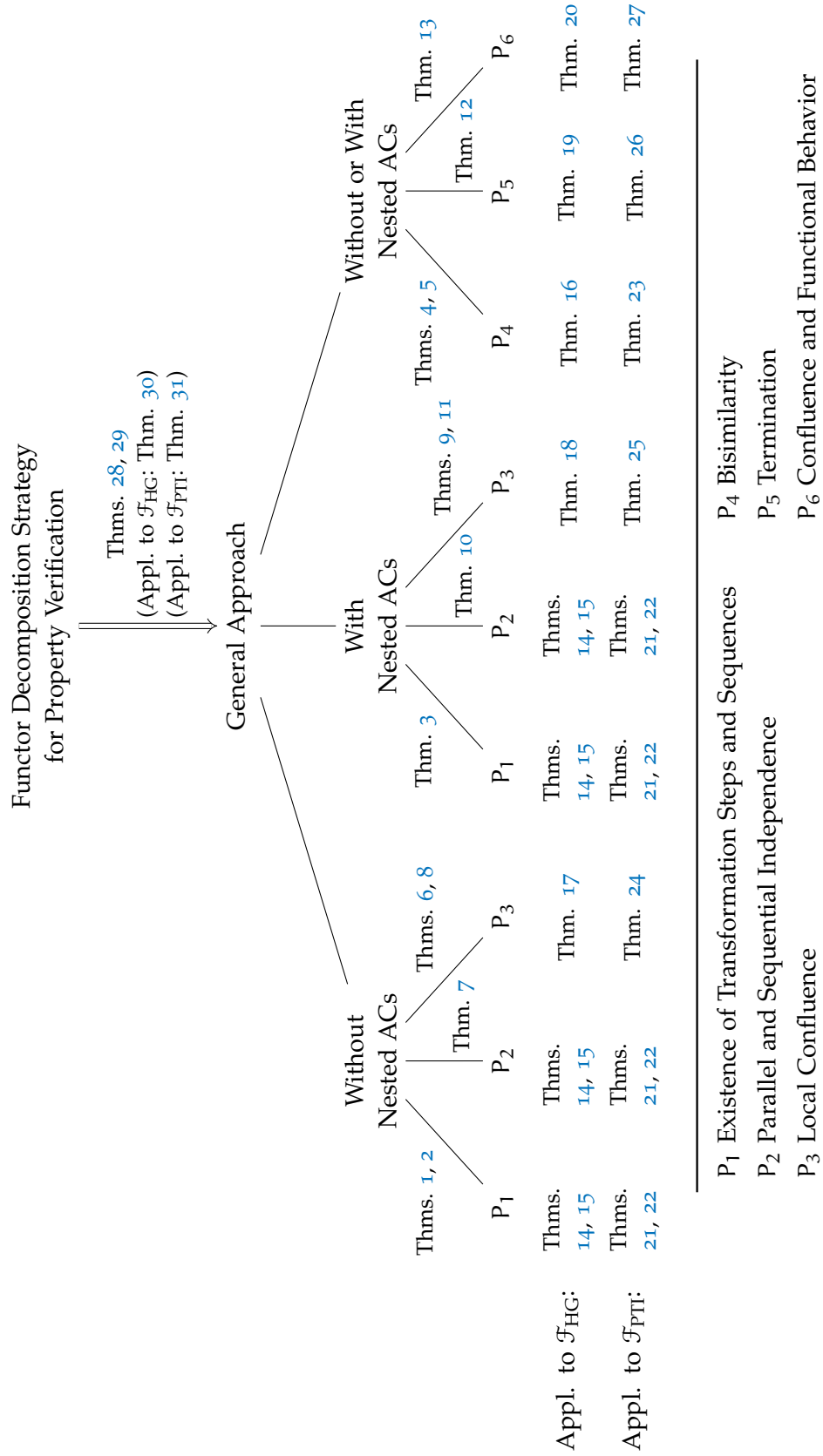


Figure 81: Overview of main results

Semantical Property	Functor	Restricted Functor
P_1	without nested ACs	1a-4a
	with nested ACs	1a-4a, 7a
P_2	without nested ACs	1a-4a
	with nested ACs	1a-4a, 7a
P_3	without nested ACs	1a-6a
	with nested ACs	1a-10a
P_4	without nested ACs	1a-4a
	with nested ACs	1a-4a, 7a
P_5	without nested ACs	1a-4a
	with nested ACs	1a-4a, 7a
P_6	without nested ACs	1a-4a
	with nested ACs	1a-4a, 7a

Semantical Property to be Transferred	Requirements to be Verified for Restricted Functors
P_1 Existence of Transformation Steps and Sequences	1a \mathcal{F} preserves monomorphisms
P_2 Parallel and Sequential Independence	2a \mathcal{F} preserves pushouts along \mathcal{M} -morphisms
P_3 Local Confluence	3a \mathcal{F} creates morphisms
P_4 Bisimilarity	4a \mathcal{F} preserves initial pushouts
P_5 Termination	5a \mathcal{F} preserves epimorphisms
P_6 Confluence and Functional Behavior	6a \mathcal{F} preserves coproducts
	7a \mathcal{F} creates \mathcal{M} -morphisms
	8a \mathcal{F} preserves pullbacks of \mathcal{M} -morphisms
	9a \mathcal{F} preserves \mathcal{E}' -instances
	10a \mathcal{F} creates \mathcal{E}' -instances

Semantical Property to be Transferred	Requirements to be Verified for Restricted Functors
P_1 Existence of Transformation Steps and Sequences	1b \mathcal{F}_R preserves monomorphisms
P_2 Parallel and Sequential Independence	2b \mathcal{F}_R preserves pushouts of \mathcal{M} -morphisms
P_3 Local Confluence	3b \mathcal{F}_R creates \mathcal{M} -morphisms
P_4 Bisimilarity	4b \mathcal{F}_R preserves initial pushouts over \mathcal{M} -morphisms
P_5 Termination	5b \mathcal{F}_R preserves epimorphisms
P_6 Confluence and Functional Behavior	6b \mathcal{F}_R preserves coproducts of \mathcal{M} -morphisms
	7b \mathcal{F}_R preserves pullbacks of \mathcal{M} -morphisms
	8b \mathcal{F}_R preserves \mathcal{E}' -instances
	9b \mathcal{F}_R creates \mathcal{E}' -instances

Figure 82: Requirements to be verified for a (restricted) functor at hand for the instantiation of the theorems of our abstract framework given in Figure 81, which allow for the transfer of the semantical properties P_1 – P_6

confluence transfer for Theorem 13 to ensure the transfer of confluence and functional behavior (property P_6), i.e., to guarantee that all transformation sequences ultimately lead to a common result that is unique in case of functional behavior.

In Part iii we have introduced the concrete \mathcal{M} -functor \mathcal{F}_{HG} that translates hypergraphs into typed attributed graphs over a specific attributed type graph $HGTG$ and have verified that \mathcal{F}_{HG} satisfies the required properties for Theorems 14–20, which allow for the sound transfer of the semantical properties P_1 – P_6 for the setting of hypergraph transformation systems. Using our first running example, in which we model a Mobile Processes scenario as a hypergraph transformation system, we have firstly provided some small examples for the theoretical concepts introduced in Part ii and, secondly, we have shown in detail the AGG-supported analysis of local confluence for our considered concrete hypergraph transformation system facilitating the theoretical results from our abstract framework.

In Part iv we have defined the restricted \mathcal{M} -functor \mathcal{F}_{PTI} for the translation of Petri nets with individual tokens into typed attributed graphs over a Petri net specific attributed type graph $PNTG$. The mentioned restriction refers to the class of morphisms of the underlying category of the source transformation system that have to be translated by the defined restricted \mathcal{M} -functor. In the case of Petri nets with individual tokens, we have used as the source category of \mathcal{F}_{PTI} the category of PTI nets restricted to monomorphisms only. This restriction to monomorphisms is a suitable choice for PTI net transformation systems because the usage of non-monomorphic morphisms alters markings of PTI nets in an undesired way when executing transformation steps for transition firing. However, the results introduced in Part ii are carefully designed to also handle restricted functors like this. As in Part iii, we have proven that our concrete restricted \mathcal{M} -functor \mathcal{F}_{PTI} satisfies the adapted sufficient properties that are required for Theorems 2–13 allowing for the instantiation of these abstract results in Theorems 21–27 for PTI net transformation systems. Based on our second running example on the Mobile Dining Philosophers problem, we have provided, similarly to Part iii, some small examples for the theoretical concepts introduced in Part ii and have revisited the AGG-based analysis of local confluence for our considered concrete PTI net transformation system.

In Part v we have introduced a functor decomposition strategy for the verification of the requirements characterizing the applicability of the developed general theory of (restricted) \mathcal{M} -functors presented in Part ii. This strategy, compared to the regular strategy introduced in Part ii, is based on functor decomposition and requires essentially (a) the definition of a category equivalence (directly based on the functor at hand) between the source category of the functor and the subcategory of the target category containing only the objects and morphisms that are images of the considered functor and (b) the verification of the required properties only for the inclusion functor between the considered subcategory of the target category and the entire target category. This decomposition of the proof process into the two steps (a) and (b) may often be advantageous for concrete instantiations because the proofs for the satisfaction of the required properties for the inclusion functor are usually less complex compared to verifying these properties directly for the \mathcal{M} -functor at hand. As in Part ii, we have also adapted the functor decomposition strategy to the case of restricted \mathcal{M} -functors and have demonstrated the applicability of the functor decomposition strategy to our both concrete functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} .

To conclude, the results of this thesis provide a formally verified framework for relating transformation-based modeling formalisms that are instantiations of \mathcal{M} -adhesive transformation systems and their analysis techniques. Our theory covers a wide range of modeling formalisms such as many kinds of graph transformation systems (based on e.g. typed and/or attributed graphs or hypergraphs) as well as different variants of Petri net transformation systems (based on e.g. PTI nets or algebraic high-level nets). The requirements assumed for the theoretical results of our framework are shown to be sufficient to transfer a reasonable set of semantical properties between two concrete modeling formalisms. The formal \mathcal{M} -functor-based relation between two modeling formalisms enables modelers to apply analysis techniques and tools designed for one modeling formalism also to another modeling formalism. Hence, it provides a way to overcome the lack of suitable analysis techniques and tools for specific modeling formalisms.

FUTURE WORK

In this chapter, we propose possible future research directions concerning our theoretical framework as well as its application.

15.1 EXTENSION OF THE FRAMEWORK

The theoretical results presented in this thesis can be extended in two fundamental directions allowing for their usage in more specific practical contexts.

Firstly, we want to extend our results by considering further semantical properties to be transferred among the involved transformation systems, enlarging as a consequence the application scope of our theory to further scenarios.

On the one hand, we are interested in an extension of our handling of the bisimulation notion from Section 3.2 to weak bisimulation [218], additionally accounting for silent internal steps not observable by a context, or even to behavioral congruences [193], possibly along the ideas introduced in [78] considering various kinds of contexts to enable the analysis whether a bisimulation is a congruence with respect to these contexts. While the notion of weak bisimulation can be formalized straightforwardly based on our definitions and results, the integration of the concept of behavioral congruence based on borrowed contexts [78] into our framework might be much more challenging.

On the other hand, it would be possible to transfer invariants and specific termination criteria among the involved transformation systems to support the verification of transformation systems on a more advanced level.

Invariants for transformation systems can be defined, for example, in the form of general nested graph conditions [135, 136] to be preserved by every transformation step. Such nested graph conditions can already be translated using our definitions from a source transformation system into the corresponding nested graph conditions in a target transformation system. Hence, the integration of the invariant transfer into our framework should be possible straightforwardly.

Termination analysis is already an important aspect of this thesis. However, since transformation systems are typically Turing powerful, we cannot expect termination to be decidable for many applications. We believe that user-assisted termination analysis based on suitable termination criteria is an appropriate approach to this problem resembling similar developments for functional programs [123, 124]. For example, using the idea from [57], we may define for a transformation system a measure for termination in the form of a mapping from objects of the underlying category to natural numbers such that any non-empty sequence of transformation steps of a certain length strictly decreases the natural number resulting from applying the mapping to the first and the last object of the sequence. Then, based on a functorial translation of such a measure, we can delegate the analysis whether a given mapping is indeed a measure to the target transformation system.

Secondly, it is interesting to extend the expressiveness of the involved transformation systems.

On the one hand, the transformation systems that are currently considered do not involve control structures, which are commonly used in practical applications to restrict the applicable rules without the need to encode such control in the transformation rules. For example, *GROOVE* [121, 298] allows for the usage of priorities or a certain control language. Similarly, in *Henshin* [69, 6] priorities can be used. The handling of (possibly quite different) control structures in the source and the target transformation systems would require our approach to be extended with a translation of these control structures to be able to transfer the system behavior appropriately.

On the other hand, the expressiveness of the used type graphs (allowing e.g. for inheritance and multiplicities as used in AGG [292]) and nested application conditions (allowing for more complex graph patterns beyond first-order logic) can be improved.

The usage of type graphs ensures that transformation steps are disabled when producing graphs that are disallowed by the type graph. Hence, the expressiveness of type graphs is very important. Until now we have not introduced results on the translation of type graphs since the two source transformation systems considered in our running examples do not require explicit type graphs.

Besides nested application conditions, as we used in this thesis, representing local graph properties in the sense of Gaifman [119], we could also allow for handling of non-local graph properties, like the existence of an arbitrary-length path between two nodes. Such modifications would affect not only the definition of the translation using \mathcal{M} -functors but also the computation and analysis of critical pairs for the case of transformations with nested application conditions.

15.2 APPLICATION OF THE FRAMEWORK

There are also several further research directions considering the application of our theoretical approach.

Firstly, it is interesting to study the relationship between other \mathcal{M} -adhesive transformation systems by developing further \mathcal{M} -functors to provide analysis support to other domains beyond hypergraph transformation systems and Petri net transformation systems with individual tokens. Examples of such \mathcal{M} -adhesive transformation systems between which \mathcal{M} -functors could be possibly considered are Petri nets with individual data-tokens, called algebraic high-level nets with individual tokens [224] (short AHLI nets), and typed attributed graphs (see [206] for initial ideas) as well as triple graphs [281] and their flattening (see [91] for a general motivation).

Secondly, tool support is very important as a long-term goal to ensure the applicability of our theoretical results in concrete settings. Since for hypergraph transformation systems and Petri net transformation systems with individual tokens there are no verification tools directly suitable for our analysis purposes, a possibility for future work would be to develop new tools or to adapt existing tools (such as *Henshin* [69] for hypergraph transformation systems and *ReConNet* [248] or *RON-tool* [291] for Petri net transformation systems with individual tokens) to allow for theory-based modeling, simulation, and analysis of these formalisms. This would ideally entail additional support for nested application conditions in all involved tools. To support the developed theory in these

tools, we want to implement the concrete functors \mathcal{F}_{HG} and \mathcal{F}_{PTI} translating hypergraphs and Petri nets with individual tokens into their corresponding typed attributed graph counterparts. Moreover, considering the inverse direction, one could also implement the translation of the analysis results in the form of artifacts of typed attributed graphs and their transformations back into the hypergraph and Petri net transformation system formalisms to present the obtained results to the user in the formalism originally employed for modeling.

For the verification of the local confluence property via the critical pair analysis in AGG, a line of future work would be also to implement a suitable automated detection of \mathcal{F}_{HG} -resp. \mathcal{F}_{PTI} -reachable critical overlapping graphs, which would facilitate the analysis of the calculated critical pairs.

Furthermore, one could extend AGG by implementing a semi-decision procedure for strict confluence analysis of the calculated critical pairs consisting of the following two steps. Firstly, for a critical pair $P_1 \Leftarrow K \Rightarrow P_2$ an interactive approach would be helpful allowing stepwise application of transformation rules to P_1 and P_2 to determine a graph, to which both transformation sequences starting in P_1 and P_2 can be merged. Secondly, we want to implement the algorithm allowing for the analysis of the strictness condition (see Definition 14) for a calculated critical pair at hand.

Additionally, for the context of transformation systems with application conditions, we intend to implement in AGG the required test for $\text{AC}(\mathcal{F})$ -compatibility of application conditions in the sense of Definition 59.

To allow for an alternative analysis of termination in the context of the AGG-tool, the implementation of a naive reachability graph generating algorithm, which is sufficient for transformation systems with finite loop-free reachability graphs, would be helpful. Furthermore, we want to implement the checking of termination criteria in the sense as explained above, to support user-assisted termination analysis of hypergraph transformation systems and Petri net transformation systems with individual tokens. Similarly, one could implement the transfer and testing of invariants as mentioned before between the source and the target transformation systems of considered functors.

- [1] J. Adámek, H. Herrlich, and G. E. Strecker. *Abstract and Concrete Categories: The Joy of Cats*. Pure and applied mathematics. Wiley, 1990.
- [2] G. Alves, A. Arcoverde, R. M. F. Lima, and P. R. M. Maciel. Ezpetri: A Petri net interchange framework for Eclipse based on PNML. In *ISO/LA*, pages 143–149, 2004.
- [3] A. Anjorin. An introduction to triple graph grammars as an implementation of the Delta-Lens framework. In Gibbons and Stevens, editors, *Bidirectional Transformations - International Summer School, Tutorial Lectures*, volume 9715 of *LNCS*, pages 29–72. Springer, 2016.
- [4] A. Anjorin, E. Leblebici, and A. Schürr. 20 years of triple graph grammars: A roadmap for future research. *ECEASST*, 73, 2015.
- [5] A. Arcoverde, G. Alves, and R. M. F. Lima. Petri nets tools integration through Eclipse. In Storey, Burke, et al., editors, *ETX*, pages 90–94. ACM, 2005.
- [6] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer. Henshin: Advanced concepts and tools for in-place EMF model transformations. In Petriu, Rouquette, et al., editors, *MODELS*, volume 6394 of *LNCS*, pages 121–135. Springer, 2010.
- [7] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *J. ACM*, 40(5):1134–1164, 1993.
- [8] U. Aßmann. Graph rewrite systems for program optimization. *ACM Trans. Program. Lang. Syst.*, 22(4):583–637, 2000.
- [9] T. Austin. Deducing the density Hales–Jewett theorem from an infinitary removal lemma. *J. of Theoretical Probability*, 24(3):615–633, 2011.
- [10] M. Awedh and F. Somenzi. Termination criteria for bounded model checking: Extensions and comparison. *ENTCS*, 144(1):51–66, 2006.
- [11] P. Baldan and B. König. Approximating the behaviour of graph transformation systems. In Corradini et al. [46], pages 14–29.
- [12] P. Baldan, A. Corradini, and B. König. Static analysis of distributed systems with mobility specified by graph grammars—A case study. In Ehrig, Krämer, et al., editors, *Proc. of Int. Conf. on Integrated Design & Process Technology*. SDPS, 2002.
- [13] P. Baldan, A. Corradini, and B. König. A framework for the verification of infinite-state graph transformation systems. *Inf. Comput.*, 206(7):869–907, 2008.
- [14] C. Batini and A. D’Atri. Rewriting systems as a tool for relational data base design. In Claus et al. [43], pages 139–154.
- [15] M. Bauderon and B. Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(2-3):83–127, 1987.
- [16] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
- [17] B. Berthomieu and F. Vernadat. Time Petri nets analysis with TINA. In *QEST*, pages 123–124. IEEE Computer Society, 2006.
- [18] E. Biermann and T. Modica. Independence analysis of firing and rule-based net transformations in reconfigurable object nets. *ECEASST*, 10, 2008.

- [19] E. Biermann, C. Ermel, F. Hermann, and T. Modica. A visual editor for reconfigurable object nets based on the Eclipse graphical editor framework. In *AWPN*, 2007.
- [20] E. Biermann, C. Ermel, T. Modica, and P. Sylopp. Implementing Petri net transformations using graph transformation tools. *ECEASST*, 14, 2008.
- [21] D. Bisztray and R. Heckel. Combining termination criteria by isolating deletion. In Ehrig et al. [75], pages 203–217.
- [22] D. Bisztray and R. Heckel. Combining termination proofs in model transformation systems. *MSCS*, 24(4), 2014.
- [23] H. L. Bodlaender and B. d. Fluiter. Reduction algorithms for constructing solutions in graphs with small treewidth. In Cai and Wong, editors, *COCOON*, volume 1090 of *LNCS*, pages 199–208. Springer, 1996.
- [24] F. Bonchi, F. Gadducci, and B. König. Process bisimulation via a graphical encoding. In Corradini et al. [47], pages 168–183.
- [25] P. Bottoni, G. Taentzer, and A. Schürr. Efficient parsing of visual languages based on critical pair analysis and contextual layered graph transformation. In *VL*, pages 59–60. IEEE Computer Society, 2000.
- [26] P. Bottoni, M. Koch, F. Parisi-Presicce, and G. Taentzer. Termination of high-level replacement units with application to model transformation. *ENTCS*, 127(4):71–86, 2005.
- [27] A. Bretto. *Applications of Hypergraph Theory: A Brief Overview*, pages 111–116. Springer, 2013.
- [28] H. J. S. Bruggink. Towards a systematic method for proving termination of graph transformation systems. *Theor. Comput. Sci.*, 213(1):23–38, 2008.
- [29] H. J. S. Bruggink, B. König, and H. Zantema. Termination analysis for graph transformation systems. In Díaz, Lanese, et al., editors, *TCS*, volume 8705 of *LNCS*, pages 179–194. Springer, 2014.
- [30] H. J. S. Bruggink, B. König, D. Nolte, and H. Zantema. Proving termination of graph transformation systems using weighted type graphs over semirings. In Parisi-Presicce and Westfechtel [249], pages 52–68.
- [31] R. Bruni and J. Meseguer. Implementing tile systems: Some examples from process calculi, 1998.
- [32] R. Bruni and U. Montanari. Zero-safe nets: Comparing the collective and individual token approaches. *Inf. Comput.*, 156(1-2):46–89, 2000.
- [33] R. Bruni, A. Lluch-Lafuente, U. Montanari, and E. Tuosto. Service oriented architectural design. In Barthe and Fournet, editors, *TGC*, volume 4912 of *LNCS*, pages 186–203. Springer, 2007.
- [34] R. Bruni, A. Lluch-Lafuente, and U. Montanari. Hierarchical design rewriting with Maude. *ENTCS*, 238(3):45–62, 2009.
- [35] A. Bucchiarone, P. Pelliccione, C. Vattani, and O. Runge. Self-repairing systems modeling and verification using AGG. In *WICSA/ECSCA*, pages 181–190. IEEE, 2009.
- [36] R. M. Burstall and J. A. Goguen. The semantics of CLEAR, a specification language. In Bjørner, editor, *Abstract Software Specifications*, volume 86 of *LNCS*, pages 292–332. Springer, 1979.

- [37] B. B. Cambazoglu and C. Aykanat. Hypergraph-partitioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids. *IEEE Transactions on Parallel and Distributed Systems*, 18(1):3–16, 2007.
- [38] L. Cardelli and A. D. Gordon. Mobile ambients. In Nivat [235], pages 140–155.
- [39] I. Castellani and U. Montanari. Graph grammars for distributed systems. In Ehrig et al. [70], pages 20–38.
- [40] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdag, R. T. Heaphy, and L. A. Riesen. A repartitioning hypergraph model for dynamic load balancing. *J. of Parallel and Distributed Computing*, 69(8):711–724, 2009.
- [41] M. Cerioli and J. Meseguer. May I borrow your logic? (Transporting logical structures along maps). *Theor. Comput. Sci.*, 173(2):311–347, 1997.
- [42] A. Chaoui and I. Hadjadj. PNTTools: A multi-language environment to integrate Petri nets tools. In Amine, Mohamed, et al., editors, *CIIA*, volume 547. CEUR-WS.org, 2009.
- [43] Claus, Ehrig, et al., editors. *Graph Grammars and Their Application to Computer Science and Biology*, volume 73 of *LNCS*, 1979. Springer.
- [44] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and J. Quesada. Maude: Specification and programming in rewriting logic. *TCS*, 285(2):187–243, 2002.
- [45] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and C. Talcott. *All About Maude—A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic*. Springer, 2007.
- [46] Corradini, Ehrig, et al., editors. *ICGT*, volume 2505 of *LNCS*, 2002. Springer.
- [47] Corradini, Ehrig, et al., editors. *ICGT*, volume 4178 of *LNCS*, 2006. Springer.
- [48] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, and M. Löwe. Graph grammars and logic programming. In Ehrig et al. [71], pages 221–237.
- [49] B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of TCS, Vol. B: Formal Models and Semantics*, pages 193–242. The MIT Press, 1990.
- [50] B. Courcelle. Context-free graph grammars: Separating vertex replacement from hyperedge replacement. In Ésik [107], pages 181–193.
- [51] B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In Rozenberg [271], pages 313–400.
- [52] *OWL Web Ontology Language Overview*. D. L. McGuinness and F. v. Harmelen, 2004. URL <http://www.w3.org/TR/2004/REC-owl-features-20040210/>.
- [53] F. Deckwerth. *Static Verification Techniques for Attributed Graph Transformations*. PhD thesis, Technische Universität, Darmstadt, 2017.
- [54] P. Degano and U. Montanari. A model for distributed systems based on graph rewriting. *J. ACM*, 34(2):411–449, 1987.
- [55] *JARP: Petri Nets Analyzer*. Department of Automation and Systems, Santa Catarina Federal University, 2001. URL <http://jarp.sourceforge.net/us/index.html>.
- [56] *Platform Independent Petri Net Editor 2 (PIPE2)*. Department of Computing, Imperial College London, 2009. URL <http://pipe2.sourceforge.net>.
- [57] N. Dershowitz. Termination of rewriting. *J. Symb. Comput.*, 3(1/2):69–116, 1987.
- [58] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Inf.*, 1(2):115–138, 1971.

- [59] A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci.*, 311(1-3):221–256, 2004.
- [60] F. Drewes. NP-completeness of k-connected hyperedge-replacement languages of order k. *Inf. Process. Lett.*, 45(2):89–94, 1993.
- [61] F. Drewes, B. Hoffmann, and M. Minas. Predictive top-down parsing for hyperedge replacement grammars. In Parisi-Presicce and Westfechtel [249], pages 19–34.
- [62] M. Droste and R. M. Shortt. From Petri nets to automata with concurrency. *Applied Categorical Structures*, 10(2):173–191, 2002.
- [63] F. Durán. Termination checker and Knuth-Bendix completion tools for Maude equational specifications. Technical report, Universidad de Málaga, 2000. URL http://maude.cs.uiuc.edu/papers/postscript/Dknuthbendix_2000.ps.gz.
- [64] F. Durán and J. Meseguer. A Church-Rosser checker tool for Maude equational specifications. Technical report, Universidad de Málaga, 2000. URL http://maude.cs.uiuc.edu/papers/postscript/DMchurchrosser_2000.ps.gz.
- [65] F. Durán and J. Meseguer. A Church-Rosser checker tool for conditional order-sorted equational Maude specifications. In Ölveczky, editor, *Rewriting Logic and Its Applications*, volume 6381 of LNCS, pages 69–85. Springer, 2010.
- [66] J. Dyck and H. Giese. Inductive invariant checking with partial negative application conditions. In Parisi-Presicce and Westfechtel [249], pages 237–253.
- [67] J. Dyck and H. Giese. K-inductive invariant checking for graph transformation systems. In Lara and Plump, editors, *ICGT*, volume 10373 of LNCS, pages 142–158. Springer, 2017.
- [68] J. Dyck, H. Giese, L. Lambers, S. Schlesinger, and S. Glesner. Towards the automatic verification of behavior preservation at the transformation level for operational model transformations. In Dingel, Kokaly, et al., editors, *Proc. of AMT 2015*, volume 1500 of CEUR, pages 36–45. CEUR-WS.org, 2015.
- [69] *EMF Henshin*. The Eclipse Foundation, 2013. URL <http://www.eclipse.org/modeling/emft/henshin>.
- [70] Ehrig, Nagl, et al., editors. *Graph Grammars and Their Application to Computer Science*, volume 153 of LNCS, 1983. Springer.
- [71] Ehrig, Kreowski, et al., editors. *Graph Grammars and Their Application to Computer Science*, volume 532 of LNCS, 1991. Springer.
- [72] Ehrig, Engels, et al., editors. *Handbook of Graph Grammars and Computing by Graph Transformation: Vol. 2: Applications, Languages, and Tools*. World Scientific, 1999.
- [73] Ehrig, Kreowski, et al., editors. *Handbook of Graph Grammars and Computing by Graph Transformation: Vol. 3: Concurrency, Parallelism, and Distribution*. World Scientific, 1999.
- [74] Ehrig, Engels, et al., editors. *ICGT*, volume 3256 of LNCS, 2004. Springer.
- [75] Ehrig, Rensink, et al., editors. *ICGT*, volume 6372 of LNCS, 2010. Springer.
- [76] H. Ehrig. Introduction to the algebraic theory of graph grammars (A survey). In Claus et al. [43], pages 1–69.
- [77] H. Ehrig and A. Habel. Graph grammars with application conditions. In *The Book of L*, pages 87–100. Springer, 1986.
- [78] H. Ehrig and B. König. Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. *MSCS*, 16(6):1133–1163, 2006.

- [79] H. Ehrig, M. Pfender, and H. J. Schneider. Graph grammars: An algebraic approach. In *SWAT (FOCS)*, pages 167–180. IEEE Computer Society, 1973.
- [80] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. From graph grammars to high-level replacement systems. In Ehrig et al. [71], pages 269–291.
- [81] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *MSCS*, 1(3):361–404, 1991.
- [82] H. Ehrig, J. Padberg, and L. Ribeiro. Algebraic high-level nets: Petri nets revisited. In Ehrig, editor, *COMPASS*, volume 785 of *LNCS*, pages 188–206. Springer, 1992.
- [83] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic approaches to graph transformation—Part II: Single pushout approach and comparison with double pushout approach. In Rozenberg [271], pages 247–312.
- [84] H. Ehrig, K. Ehrig, A. Habel, and K.-H. Pennemann. Constraints and application conditions: From graphs to high-level structures. In Ehrig et al. [74], pages 287–303.
- [85] H. Ehrig, U. Prange, and G. Taentzer. Fundamental theory for typed attributed graph transformation. In Ehrig et al. [74], pages 161–177.
- [86] H. Ehrig, K. Ehrig, J. d. Lara, G. Taentzer, D. Varró, and S. Varró-Gyapay. Termination criteria for model transformation. In Cerioli, editor, *FASE*, volume 3442 of *LNCS*, pages 49–63. Springer, 2005.
- [87] H. Ehrig, K. Ehrig, A. Habel, and K.-H. Pennemann. Theory of constraints and application conditions: From graphs to high-level structures. *Fundam. Inform.*, 74(1):135–166, 2006.
- [88] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in TCS. Springer, 2006.
- [89] H. Ehrig, A. Habel, J. Padberg, and U. Prange. Adhesive high-level replacement systems: A new categorical framework for graph transformation. *Fundam. Inform.*, 74(1):1–29, 2006.
- [90] H. Ehrig, K. Hoffmann, and J. Padberg. Transformations of Petri nets. *ENTCS*, 148(1):151–172, 2006.
- [91] H. Ehrig, C. Ermel, and F. Hermann. On the relationship of model transformations based on triple and plain graph grammars. In Karsai and Taentzer, editors, *GraMoT*, pages 9–16. ACM, 2008.
- [92] H. Ehrig, K. Hoffmann, J. Padberg, C. Ermel, U. Prange, E. Biermann, and T. Modica. Petri net transformations. In *Petri Net Theory and Applications*, pages 1–16. I-Tech Education and Publication, 2008.
- [93] H. Ehrig, F. Hermann, and U. Prange. Cospan DPO approach: An alternative for DPO graph transformations. *Bulletin of the EATCS*, 98:139–149, 2009.
- [94] H. Ehrig, U. Golas, and F. Hermann. Categorical frameworks for graph transformation and HLR systems based on the DPO approach. *EATCS*, 102:111–121, 2010.
- [95] H. Ehrig, A. Habel, and L. Lambers. Parallelism and concurrency theorems for rules with nested application conditions. *ECEASST*, 26:1–24, 2010.
- [96] H. Ehrig, A. Habel, L. Lambers, F. Orejas, and U. Golas. Local confluence for rules with nested application conditions. In Ehrig, Rensink, et al., editors, *ICGT*, volume 6372 of *LNCS*, pages 330–345. Springer, 2010.

- [97] H. Ehrig, C. Ermel, F. Hüffner, R. Niedermeier, and O. Runge. Confluence in data reduction: Bridging graph transformation and kernelization. *Computability*, 2(1): 31–49, 2013.
- [98] H. Ehrig, C. Ermel, U. Golas, and F. Hermann. *Graph and Model Transformation: General Framework and Applications*. EATCS Monographs in TCS. Springer, 2015.
- [99] K. Ehrig, E. Guerra, J. d. Lara, L. Lengyel, T. Levendovszky, U. Prange, G. Taentzer, D. Varró, and S. Varró-Gyapay. Model transformation by graph transformation: A comparative study. In *MTiP*, 2005.
- [100] J. Engelfriet and G. Rozenberg. A comparison of boundary graph grammars and context-free hypergraph grammars. *Inf. Comput.*, 84(2):163–206, 1990.
- [101] J. Engelfriet and G. Rozenberg. Node replacement graph grammars. In Rozenberg [271], pages 1–94.
- [102] G. Engels, R. Heckel, and J. M. Küster. Rule-based specification of behavioral consistency based on the UML meta-model. In *UML*, pages 272–286, 2001.
- [103] C. Ermel, J. Padberg, and H. Ehrig. Requirements engineering of a medical information system using rule-based refinement of Petri nets. In *Proc. of Integrated Design and Process Technology*, volume 1, pages 186–193. Citeseer, 1996.
- [104] C. Ermel, M. Rudolf, and G. Taentzer. The AGG approach: Language and environment. In Ehrig et al. [72], pages 551–603.
- [105] C. Ermel, T. Modica, E. Biermann, H. Ehrig, and K. Hoffmann. Modeling multicasting in communication spaces by reconfigurable high-level Petri nets. In *VL/HCC*, pages 47–50, 2009.
- [106] C. Ermel, S. Shareef, and W. Fischer. RONS revisited: General approach to model reconfigurable object nets based on algebraic high-level nets. *ECEASST*, 40, 2010.
- [107] Ésik, editor. *FCT*, volume 710 of *LNCS*, 1993. Springer.
- [108] R. Farrow, K. Kennedy, and L. Zucconi. Graph grammars and global program data flow analysis. In *FOCS*, pages 42–56. IEEE Computer Society, 1976.
- [109] J. Feder. Plex languages. *Inf. Sci.*, 3(3):225–241, 1971.
- [110] G. L. Ferrari, U. Montanari, and E. Tuosto. An LTS semantics of ambients via graph synchronization with mobility. In Restivo, Rocca, et al., editors, *ICTCS*, volume 2202 of *LNCS*, pages 1–16. Springer, 2001.
- [111] G. L. Ferrari, U. Montanari, and E. Tuosto. Graph-based models of internetworking systems. In Aichernig and Maibaum, editors, *Formal Methods at the Crossroads. From Panacea to Foundational Support, 10th Anniversary Colloquium of UNU/IIST*, volume 2757 of *LNCS*, pages 242–266. Springer, 2002.
- [112] G. L. Ferrari, D. Hirsch, I. Lanese, U. Montanari, and E. Tuosto. Synchronised hyperedge replacement as a model for service oriented computing. In Boer, Bonsangue, et al., editors, *FMCO*, volume 4111 of *LNCS*, pages 22–43. Springer, 2005.
- [113] C. Flamm, B. M. Stadler, and P. F. Stadler. Generalized topologies: Hypergraphs, chemical reactions, and biological evolution. *Advances in Mathematical Chemistry: With applications to chemoinformatics, bioinformatics, drug discovery, and predictive toxicology*, 2:300–327, 2016.
- [114] H. Furstenberg and Y. Katznelson. An ergodic Szemerédi theorem for commuting transformations. *J. Analyse Math.*, 34:275–291, 1978.
- [115] A. L. Furtado. Transformations of data base structures. In Claus et al. [43], pages 224–236.

- [116] K. Gabriel. *Interaction on Human-Centric Communication Platforms*. PhD thesis, Technische Universität Berlin, 2014.
- [117] F. Gadducci and A. Lluch-Lafuente. Graphical verification of a spatial logic for the Pi-calculus. *ENTCS*, 154(2):31–46, 2006.
- [118] F. Gadducci and A. Lluch-Lafuente. Graphical encoding of a spatial logic for the Pi-calculus. In Mossakowski, Montanari, et al., editors, *CALCO*, volume 4624 of *LNCS*, pages 209–225. Springer, 2007.
- [119] H. Gaifman. On local and non-local properties. In Stern, editor, *Proc. of the Herbrand Symposium*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105–135. Elsevier, 1982.
- [120] G. Gallo, G. Longo, and S. Pallottino. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2):177–201, 1993.
- [121] A. H. Ghamarian, M. d. Mol, A. Rensink, E. Zambon, and M. Zimakova. Modelling and analysis using GROOVE. *STTT*, 14(1):15–40, 2012.
- [122] H. Giese, L. Lambers, B. Becker, S. Hildebrandt, S. Neumann, T. Vogel, and S. Wätzoldt. Graph transformations for MDE, adaptation, and models at runtime. In Bernardo, Cortellessa, et al., editors, *SFM*, volume 7320 of *LNCS*, pages 137–191. Springer, 2012.
- [123] J. Giesl. Termination analysis for functional programs using term orderings. In Mycroft, editor, *SAS*, volume 983 of *LNCS*, pages 154–171. Springer, 1995.
- [124] J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM Trans. Program. Lang. Syst.*, 33(2):7:1–7:39, 2011.
- [125] A. Gill. *Introduction to the Theory of Finite-State Machines*. McGraw-Hill, 1962.
- [126] J. A. Goguen and R. M. Burstall. Introducing institutions. In Clarke and Kozen, editors, *Logics of Programs*, volume 164 of *LNCS*, pages 221–256. Springer, 1983.
- [127] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *J. ACM*, 39(1):95–146, 1992.
- [128] J. A. Goguen and G. Rosu. Institution morphisms. *Formal Asp. Comput.*, 13(3-5):274–307, 2002.
- [129] J. A. Goguen, T. Mossakowski, V. d. Paiva, F. Rabe, and L. Schröder. An institutional view on categorical logic. *Int. J. Software and Informatics*, 1(1):129–152, 2007.
- [130] S. Gottmann and N. Nachtigall. Modeling the living place project using algebraic higher-order nets. Diploma thesis, Technische Universität Berlin, 2011.
- [131] E. Grabska, G. M. Slusarczyk, and T. L. Le. Visual design and reasoning with the use of hypergraph transformations. *ECEASST*, 10, 2008.
- [132] L. Grunske. Formalizing architectural refactorings as graph transformation systems. In Chung and Song, editors, *SNPD*, pages 324–329. IEEE Computer Society, 2005.
- [133] A. Habel and H.-J. Kreowski. On context-free graph languages generated by edge replacement. In Ehrig et al. [70], pages 143–158.
- [134] A. Habel and H.-J. Kreowski. Some structural aspects of hypergraph languages generated by hyperedge replacement. In Brandenburg, Vidal-Naquet, et al., editors, *STACS*, volume 247 of *LNCS*, pages 207–219. Springer, 1987.
- [135] A. Habel and K.-H. Pennemann. Nested constraints and application conditions for high-level structures. In Kreowski, Montanari, et al., editors, *Formal Methods*

- in *Software and Systems Modeling*, volume 3393 of *LNCS*, pages 294–308. Springer, 2005.
- [136] A. Habel and K.-H. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *MSCS*, 19:1–52, 2009.
 - [137] A. Habel and D. Plump. Computational completeness of programming languages based on graph transformation. In Honsell and Miculan [155], pages 230–245.
 - [138] A. Habel and H. Radke. Expressiveness of graph conditions with variables. *ECE-ASST*, 30, 2010.
 - [139] A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. *Fundam. Inform.*, 26(3/4):287–313, 1996.
 - [140] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987.
 - [141] J. H. Hausmann, R. Heckel, and G. Taentzer. Detection of conflicting functional requirements in a use case driven approach: A static analysis technique based on graph transformation. In Tracz, Young, et al., editors, *ICSE*, pages 105–115. ACM, 2002.
 - [142] Heckel and Taentzer, editors. *Graph Transformation, Specifications, and Nets—In Memory of Hartmut Ehrig*, volume 10800 of *LNCS*, 2018. Springer.
 - [143] R. Heckel and A. Wagner. Ensuring consistency of conditional graph grammars—A constructive approach. *ENTCS*, 2:118–126, 1995.
 - [144] R. Heckel, J. M. Küster, and G. Taentzer. Confluence of typed attributed graph transformation systems. In Corradini et al. [46], pages 161–176.
 - [145] M. Heiner, M. Schwarick, and J.-T. Wegener. Charlie—An extensible Petri net analysis tool. In Devillers and Valmari, editors, *PETRI NETS*, volume 9115 of *LNCS*, pages 200–211. Springer, 2015.
 - [146] F. Hermann, H. Ehrig, F. Orejas, K. Czarnecki, Z. Diskin, Y. Xiong, S. Gottmann, and T. Engel. Model synchronization based on triple graph grammars: Correctness, completeness, and invertibility. *Software and System Modeling*, 14(1):241–269, 2015.
 - [147] D. Hirsch. *Graph Transformation Models for Software Architecture Styles*. PhD thesis, Departamento de Computacion, Universidad de Buenos Aires, 2003.
 - [148] D. Hirsch and U. Montanari. Synchronized hyperedge replacement with name mobility. In Larsen and Nielsen, editors, *CONCUR*, volume 2154 of *LNCS*, pages 121–136. Springer, 2001.
 - [149] D. Hirsch, P. Inverardi, and U. Montanari. Reconfiguration of software architecture styles with name mobility. In Porto and Roman, editors, *COORDINATION*, volume 1906 of *LNCS*, pages 148–163. Springer, 2000.
 - [150] C. A. R. Hoare. A model for communicating sequential processes. In *On the Construction of Programs*, pages 229–254. Cambridge University Press New York, NY, USA, 1980.
 - [151] B. Hoffmann. Modelling compiler generation by graph grammars. In Ehrig et al. [70], pages 159–171.
 - [152] K. Hoffmann and T. Mossakowski. Algebraic higher-order nets: Graphs and Petri nets as tokens. In Wirsing, Pattinson, et al., editors, *WADT*, volume 2755 of *LNCS*, pages 253–267. Springer, 2002.

- [153] K. Hoffmann, H. Ehrig, and T. Mossakowski. High-level nets with nets and rules as tokens. In Ciardo and Darondeau, editors, *ICATPN*, volume 3536 of *LNCS*, pages 268–288. Springer, 2005.
- [154] K. Hoffmann, H. Ehrig, and J. Padberg. Flexible modeling of emergency scenarios using reconfigurable systems. *ECEASST*, 12, 2008.
- [155] Honsell and Miculan, editors. *FOSSACS*, volume 2030 of *LNCS*, 2001. Springer.
- [156] *Integrated Net Analyzer 2.2 (INA 2.2)*. Humboldt-Universität zu Berlin, 2003. URL <http://www2.informatik.hu-berlin.de/~starke/ina.html>.
- [157] E. Jakumeit, S. Buchwald, D. Wagelaar, L. Dan, Á. Hegedüs, M. Herrmannsdörfer, T. Horn, E. Kalnina, C. Krause, K. Lano, M. Lepper, A. Rensink, L. M. Rose, S. Wätzdolt, and S. Mazanek. A survey and comparison of transformation tools based on the transformation tool contest. *Sci. Comput. Program.*, 85:41–99, 2014.
- [158] D. Janssens and G. Rozenberg. Node-label controlled graph grammars (Extended abstract). In Dembiński, editor, *MFCS*, pages 334–347. Springer, 1980.
- [159] D. Janssens and G. Rozenberg. Graph grammars with node-label controlled rewriting and embedding. In Ehrig et al. [70], pages 186–205.
- [160] D. Janssens, G. Rozenberg, and R. Verraedt. On sequential and parallel node-rewriting graph grammars. *Computer Graphics and Image Processing*, 18(3):279–304, 1982.
- [161] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*, volume 1. Springer, 2013.
- [162] X. Jin, Y. Lembachar, and G. Ciardo. Symbolic termination and confluence checking for ECA rules. *Trans. Petri Nets and Other Models of Concurrency*, 9:99–123, 2014.
- [163] M. John, C. Lhoussaine, J. Niehren, and C. Versari. Biochemical reaction rules with constraints. In Barthe, editor, *ESOP*, volume 6602 of *LNCS*, pages 338–357. Springer, 2011.
- [164] L. Kahloul, S. Bourekkache, and K. Djouani. Designing reconfigurable manufacturing systems using reconfigurable object Petri nets. *Int. J. Computer Integrated Manufacturing*, 29(8):889–906, 2016.
- [165] T. Kehrer. *Calculation and Propagation of Model Changes Based on User-Level Edit Operations: A Foundation for Version and Variant Management in Model-Driven Engineering*. PhD thesis, University of Siegen, 2015.
- [166] L. A. Khuzayem and P. McBrien. Knowledge transformation using a hypergraph data model. In Jones, editor, *ICCSW*, volume 28 of *OASICS*, pages 1–7. Schloss Dagstuhl — Leibniz-Zentrum fuer Informatik, Germany, 2012.
- [167] E. Kindler and M. Weber. The Petri net kernel—An infrastructure for building Petri net tools. *STTT*, 3(4):486–497, 2001.
- [168] S. Klamt, U.-U. Haus, and F. J. Theis. Hypergraphs and cellular networks. *PLoS Computational Biology*, 5(5), 2009.
- [169] R. Klempien-Hinrichs. Net refinement by pullback rewriting. In Nivat [235], pages 189–202.
- [170] A. Knapp and T. Mossakowski. UML interactions meet state machines—An institutional approach. In Bonchi and König, editors, *CALCO*, volume 72 of *LIPICs*, pages 15:1–15:15. Schloss Dagstuhl — Leibniz-Zentrum fuer Informatik, 2017.
- [171] A. Knapp, T. Mossakowski, and M. Roggenbach. An institutional framework for heterogeneous formal development in UML. *CoRR*, abs/1403.7747, 2014.

- [172] A. Knapp, T. Mossakowski, M. Roggenbach, and M. Glauer. An institution for simple UML state machines. In Egyed and Schaefer, editors, *FASE*, volume 9033 of *LNCS*, pages 3–18. Springer, 2015.
- [173] M. Koch and F. Parisi-Presicce. Describing policies with graph constraints and rules. In Corradini et al. [46], pages 223–238.
- [174] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A formal model for role-based access control using graph transformation. In Cuppens, Deswarte, et al., editors, *ESORICS*, volume 1895 of *LNCS*, pages 122–139. Springer, 2000.
- [175] M. Koch, L. V. Mancini, and F. Parisi-Presicce. Foundations for a graph-based approach to the specification of access control policies. In Honsell and Miculan [155], pages 287–302.
- [176] B. König and V. Kozioura. Augur—A tool for the analysis of graph transformation systems. *Bulletin of the EATCS*, 87:126–137, 2005.
- [177] B. König and V. Kozioura. Augur 2—A new version of a tool for the analysis of graph transformation systems. *ENTCS*, 211:201–210, 2008.
- [178] E. V. Konstantinova and V. A. Skorobogatov. Application of hypergraph theory in chemistry. *Discrete Mathematics*, 235(1-3):365–383, 2001.
- [179] H.-J. Kreowski. A pumping lemma for context-free graph languages. In Claus et al. [43], pages 270–283.
- [180] H.-J. Kreowski. A comparison between Petri nets and graph grammars. In *GTCCS*, volume 100 of *LNCS*, pages 1–19. Springer, 1981.
- [181] H.-J. Kreowski. Five facets of hyperedge replacement beyond context-freeness. In Ésik [107], pages 69–86.
- [182] H.-J. Kreowski and T. Mossakowski. Equivalence and difference between institutions: Simulating Horn clause logic with based algebras. *MSCS*, 5(2):189–215, 1995.
- [183] S. Kuske. A formal semantics of UML state machines based on structured graph transformation. In *UML*, pages 241–256, 2001.
- [184] *Time Petri Net Analyzer (Tina)*. Laboratory for Analysis and Architecture of Systems, Toulouse, 2006. URL <http://projects.laas.fr/tina>.
- [185] S. Lack and P. Sobocinski. Adhesive categories. In Walukiewicz, editor, *FOSSACS*, volume 2987 of *LNCS*, pages 273–288. Springer, 2004.
- [186] L. Lambers, M. Navarro, F. Orejas, and E. Pino. Towards a navigational logic for graphical structures. In Heckel and Taentzer [142], pages 124–141.
- [187] L. Lambers, D. Strüder, G. Taentzer, K. Born, and J. Huebert. Multi-granular conflict and dependency analysis in software engineering based on graph transformation. In Chaudron, Crnkovic, et al., editors, *ICSE*, pages 716–727. ACM, 2018.
- [188] I. Lanese and U. Montanari. A graphical Fusion calculus. *ENTCS*, 104:199–215, 2004.
- [189] I. Lanese and E. Tuosto. Synchronized hyperedge replacement for heterogeneous systems. In Jacquet and Picco, editors, *COORDINATION*, volume 3454 of *LNCS*, pages 220–235. Springer, 2005.
- [190] C. Lautemann. The complexity of graph languages generated by hyperedge replacement. *Acta Inf.*, 27(5):399–421, 1990.
- [191] K. Lautenbach, J. R. Müller, and S. Philippi. Modellierung, Simulation und Analyse mit dem Petri-Netz-Tool POSEIDON. In Desel and Weske, editors, *Promise*, volume 21 of *LNI*, pages 163–174. GI, 2002.

- [192] E. Leblebici. *Inter-Model Consistency Checking and Restoration with Triple Graph Grammars*. PhD thesis, Darmstadt University of Technology, 2018.
- [193] J. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In *CONCUR*, pages 243–258. Springer, 2000.
- [194] T. Levendovszky, U. Prange, and H. Ehrig. Termination criteria for DPO transformations with injective matches. *Theor. Comput. Sci.*, 175(4):87–100, 2007.
- [195] C. Lin, Y. Q. II, F. Ren, and D. C. Marinescu. Performance equivalent analysis of workflow systems based on stochastic Petri net models. In Han, Tai, et al., editors, *EDCIS*, volume 2480 of *LNCS*, pages 64–79. Springer, 2002.
- [196] D.-R. Liu and M.-Y. Wu. A hypergraph based approach to declustering problems. *Distributed and Parallel Databases*, 10(3):269–288, 2001.
- [197] H. Liu, D. Dou, R. Jin, P. LePendu, and N. Shah. Mining biomedical ontologies and data using hypergraphs. In *ICMLA*, pages 141–146, 2013.
- [198] M. Llorens and J. Oliver. MCRenNet: A tool for marked-controlled reconfigurable nets. In *QEST*, pages 255–256. IEEE Computer Society, 2005.
- [199] M. Löwe. Algebraic approach to single-pushout graph transformation. *TCS*, 109(1&2):181–224, 1993.
- [200] M. Löwe and H. Ehrig. Algebraic approach to graph transformation based on single pushout derivations. In Möhring, editor, *WG*, volume 484 of *LNCS*, pages 338–353. Springer, 1990.
- [201] M. Löwe and J. Müller. Critical pair analysis in single-pushout graph rewriting. In Feruglio and Llompарт, editors, *Colloquium on Graph Transformation and its Application in Computer Science*. B-19, 1995.
- [202] M. Löwe, M. Korff, and A. Wagner. An algebraic framework for the transformation of attributed graphs. In Sleep, Plasmeijer, et al., editors, *Term Graph Rewriting*, pages 185–199. John Wiley and Sons Ltd., 1993.
- [203] J. P. Loyall and S. M. Kaplan. Visual concurrent programming with grammars. *J. Vis. Lang. Comput.*, 3(2):107–133, 1992.
- [204] D. Lucanu, Y.-F. Li, and J. S. Dong. Institution morphisms for relating OWL and Z. In Chu, Juzgado, et al., editors, *SEKE*, pages 286–291, 2005.
- [205] M. A. Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems*. MIT Press, 1986.
- [206] M. Maximova. Formal relationship between Petri net and graph transformation systems based on functors between \mathcal{M} -adhesive categories. Diploma thesis, Technische Universität Berlin, 2011.
- [207] M. Maximova, H. Ehrig, and C. Ermel. Formal relationship between Petri net and graph transformation systems based on functors between \mathcal{M} -adhesive categories. *ECEASST*, 40, 2010.
- [208] M. Maximova, H. Ehrig, and C. Ermel. Functors between \mathcal{M} -adhesive categories applied to Petri net and graph transformation systems. Technical Report 4, Technische Universität Berlin, 2011. URL https://www.eecs.tu-berlin.de/fileadmin/f4/TechReports/2011/tr_2011-04.pdf.
- [209] M. Maximova, H. Ehrig, and C. Ermel. Transfer of local confluence and termination between Petri net and graph transformation systems based on \mathcal{M} -functors. *ECEASST*, 51, 2012.

- [210] M. Maximova, H. Ehrig, and C. Ermel. Transfer of local confluence and termination between Petri net and graph transformation systems based on \mathcal{M} -functors: Extended version. Technical Report 8, Technische Universität Berlin, 2012. URL https://www.eecs.tu-berlin.de/fileadmin/f4/TechReports/2012/tr_2012-08.pdf.
- [211] M. Maximova, H. Ehrig, and C. Ermel. Analysis of hypergraph transformation systems in AGG based on \mathcal{M} -functors. *ECEASST*, 58, 2013.
- [212] M. Maximova, H. Ehrig, and C. Ermel. Analysis of hypergraph transformation systems in AGG based on \mathcal{M} -functors: Extended version. Technical Report 2, Technische Universität Berlin, 2013. URL https://www.eecs.tu-berlin.de/fileadmin/f4/TechReports/2013/tr_2013-02.pdf.
- [213] M. Maximova, H. Ehrig, and C. Ermel. Local confluence analysis of hypergraph transformation systems with application conditions based on \mathcal{M} -functors and AGG. *Sci. Comput. Program.*, 104:44–70, 2015.
- [214] P. Merlin and D. Farber. Recoverability of communication protocols—Implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976.
- [215] J. Meseguer and N. Martí-Oliet. From abstract data types to logical frameworks. In Astesiano, Reggio, et al., editors, *COMPASS*, volume 906 of *LNCS*, pages 48–80. Springer, 1994.
- [216] J. Meseguer and U. Montanari. Petri nets are monoids. *Inf. Comput.*, 88(2):105–155, 1990.
- [217] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- [218] R. Milner. *Communicating and Mobile Systems—The Pi-Calculus*. Cambridge University Press, 1999.
- [219] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Inf. Comput.*, 100(1):1–77, 1992.
- [220] M. Minas. Hypergraphs as a uniform diagram representation model. In Ehrig, Engels, et al., editors, *TAGT*, volume 1764 of *LNCS*, pages 281–295. Springer, 1998.
- [221] M. Minas. Concepts and realization of a diagram editor generator based on hypergraph transformation. *Sci. Comput. Program.*, 44(2):157–180, 2002.
- [222] M. Minas and O. Köth. Generating diagram editors with *DiaGen*. In Nagl, Schürr, et al., editors, *AGTIVE*, volume 1779 of *LNCS*, pages 433–440. Springer, 1999.
- [223] T. Modica. *Formal Modeling, Simulation, and Validation of Communication Platforms*. PhD thesis, Technische Universität Berlin, 2012.
- [224] T. Modica, K. Gabriel, H. Ehrig, K. Hoffmann, S. Shareef, C. Ermel, U. Golas, F. Hermann, and E. Biermann. Low- and high-level Petri nets with individual tokens. Technical Report 13, Technische Universität Berlin, 2009. URL <http://orbi.lu.uni.lu/bitstream/10993/5620/1/tr-2009-13.pdf>.
- [225] B. Molnár. Applications of hypergraphs in informatics: A survey and opportunities for research. *Ann. Univ. Sci. Budapest. Sect. Comput.*, 42:261–282, 2014.
- [226] B. Molnár and Z. Vincellér. Comparative study of architecture for Twitter analysis and a proposal for an improved approach. In *CogInfoCom*, pages 11–16, 2013.
- [227] T. Mossakowski. *Representations, Hierarchies, and Graphs of Institutions*. PhD thesis, University of Bremen, Germany, 1996.
- [228] T. Mossakowski. Translating OBJ3 into CASL: The institution level. In Fiadeiro, editor, *WADT*, volume 1589 of *LNCS*, pages 198–215. Springer, 1998.

- [229] T. Mossakowski. Specifications in an arbitrary institution with symbols. In Bert, Choppy, et al., editors, *WADT*, volume 1827 of *LNCS*, pages 252–270. Springer, 1999.
- [230] T. Mossakowski and M. Roggenbach. Structured CSP—A process algebra as an institution. In Fiadeiro and Schobbens, editors, *WADT*, volume 4409 of *LNCS*, pages 92–110. Springer, 2006.
- [231] T. Mossakowski, R. Diaconescu, and A. Tarlecki. What is a logic translation? *Logica Universalis*, 3(1):95–124, 2009.
- [232] T. Murata. Petri nets: Properties, analysis, and applications. *Proc. of the IEEE*, 77(4): 541–580, 1989.
- [233] *Neo4j*. Neo Technology, 2012. URL <http://neo4j.org/>.
- [234] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.
- [235] Nivat, editor. *FOSSACS*, volume 1378 of *LNCS*, 1998. Springer.
- [236] *Unified Modeling Language Specification (UML)*. Object Management Group, 1998. URL <http://www.omg.org>.
- [237] *Business Process Model and Notation (BPMN): Version 2.0 Specification*. Object Management Group, 2011. URL <http://www.bpmn.org>.
- [238] L. O'Reilly, T. Mossakowski, and M. Roggenbach. Compositional modelling and reasoning in an institution for processes and data. In Mossakowski and Kreowski, editors, *WADT*, volume 7137 of *LNCS*, pages 251–269. Springer, 2010.
- [239] F. Orejas and L. Lambers. Symbolic attributed graphs for attributed graph transformation. *ECEASST*, 30, 2010.
- [240] F. Orejas and L. Lambers. Delaying constraint solving in symbolic graph transformation. In Ehrig et al. [75], pages 43–58.
- [241] F. Orejas and L. Lambers. Lazy graph transformation. *Fundam. Inform.*, 118(1-2): 65–96, 2012.
- [242] F. Orejas, A. Boronat, and N. Mylonakis. Borrowed contexts for attributed graphs. In Ehrig, Engels, et al., editors, *ICGT*, volume 7562 of *LNCS*, pages 126–140. Springer, 2012.
- [243] J. Padberg. Petri net modules. *Transactions of the SDPS*, 6(4):105–120, 2002.
- [244] J. Padberg and L. Kahloul. Overview of reconfigurable Petri nets. In Heckel and Taentzer [142], pages 201–222.
- [245] J. Padberg and A. Schulz. Model checking reconfigurable Petri nets with Maude. In Echahed and Minas, editors, *ICGT*, volume 9761 of *LNCS*, pages 54–70. Springer, 2016.
- [246] J. Padberg, H. Ehrig, and L. Ribeiro. Algebraic high-level net transformation systems. *MSCS*, 5:217–256, 1995.
- [247] J. Padberg, P. Schiller, and H. Ehrig. New concepts for high-level Petri nets in the application domain of train control systems. In *IFAC '00*, 2001.
- [248] J. Padberg, M. Ede, G. Oelker, and K. Hoffmann. ReConNet: A tool for modeling and simulating with reconfigurable Place/Transition nets. *ECEASST*, 54, 2012.
- [249] Parisi-Presicce and Westfechtel, editors. *ICGT*, volume 9151 of *LNCS*, 2015. Springer.
- [250] T. Pavlidis. Linear and context-free graph grammars. *J. ACM*, 19(1):11–22, 1972.

- [251] M. Pedicchio and W. Tholen. *Categorical Foundations: Special Topics in Order, Topology, Algebra, and Sheaf Theory*. Cambridge University Press, 2004.
- [252] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Bonn, 1962.
- [253] D. Plump. *Evaluation of Functional Expressions by Hypergraph Rewriting*. PhD thesis, Universität Bremen, 1993.
- [254] D. Plump. Hypergraph rewriting: Critical pairs and undecidability of confluence. In Sleep, Plasmeijer, et al., editors, *Term Graph Rewriting*, pages 201–213. John Wiley and Sons Ltd., 1993.
- [255] D. Plump. *On Termination of Graph Rewriting*, volume 1017 of LNCS. Springer, 1995.
- [256] D. Plump. Termination of graph rewriting is undecidable. *Fundam. Inform.*, 33(2): 201–209, 1998.
- [257] D. Plump. Modular termination of graph transformation. In Heckel and Taentzer [142], pages 231–244.
- [258] D. Plump and S. Steinert. Towards graph programs for graph algorithms. In Ehrig et al. [74], pages 128–143.
- [259] U. Prange and H. Ehrig. From algebraic graph transformation to adhesive HLR categories and systems. In *Algebraic Informatics. Proc. of CAI 2007*, volume 4728 of LNCS, pages 122–146. Springer, 2007.
- [260] U. Prange, H. Ehrig, K. Hoffman, and J. Padberg. Transformations in reconfigurable Place/Transition systems. In *Concurrency, Graphs, and Models: Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, volume 5065 of LNCS, pages 96–113. Springer, 2008.
- [261] H. Radke. Hr* graph conditions between counting monadic second-order and second-order graph formulas. *ECEASST*, 61, 2013.
- [262] G. Rangel. *Behavioral Congruences and Verification of Graph Transformation Systems with Applications to Model Refactoring*. PhD thesis, Technische Universität Berlin, 2008.
- [263] G. Rangel, B. König, and H. Ehrig. Deriving bisimulation congruences in the presence of negative application conditions. In Amadio, editor, *FOSSACS*, volume 4962 of LNCS, pages 413–427. Springer, 2008.
- [264] G. Rangel, L. Lambers, B. König, H. Ehrig, and P. Baldan. Behavior preservation in model refactoring using DPO transformations with borrowed contexts. In Ehrig, Heckel, et al., editors, *ICGT*, volume 5214 of LNCS, pages 242–256. Springer, 2008.
- [265] J.-C. Raoult. On graph rewritings. *TCS*, 32:1–24, 1984.
- [266] W. Reisig. Petri nets with individual tokens. *TCS*, 41:185–213, 1985.
- [267] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in TCS*. Springer, 1985.
- [268] W. Reisig and G. Rozenberg. Informal introduction to Petri nets. In Reisig and Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of LNCS, pages 1–11. Springer, 1996.
- [269] J. Rekers and A. Schürr. Defining and parsing visual languages with layered graph grammars. *J. Vis. Lang. Comput.*, 8(1):27–55, 1997.
- [270] A. Rensink. Representing first-order logic using graphs. In Ehrig et al. [74], pages 319–335.
- [271] Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Vol. 1: Foundations*. World Scientific, 1997.

- [272] O. Runge, C. Ermel, and G. Taentzer. AGG 2.0—New features for specifying and analyzing algebraic graph transformations. In Schürr, Varró, et al., editors, *AGTIVE*, volume 7233 of *LNCS*, pages 81–88. Springer, 2011.
- [273] *Advanced Bisimulation Checker (ABC)*. S. Briaïs, 2018. URL <http://sbriaïs.free.fr/tools/abc/>.
- [274] D. Sannella and A. Tarlecki. On observational equivalence and algebraic specification. In Ehrig, Floyd, et al., editors, *TAPSOFT CAAP*, volume 185 of *LNCS*, pages 308–322. Springer, 1985.
- [275] D. Sannella and A. Tarlecki. On observational equivalence and algebraic specification. *J. Comput. Syst. Sci.*, 34(2/3):150–178, 1987.
- [276] D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Inf. Comput.*, 76(2/3):165–210, 1988.
- [277] H. J. Schneider. Graph transformations: An introduction to the categorical approach. Technical report, Friedrich-Alexander Universität Erlangen-Nürnberg, 2010. URL <https://www2.informatik.uni-erlangen.de/staff/schneider/gtbook/references.pdf>.
- [278] S. Schneider, L. Lambers, and F. Orejas. Symbolic model generation for graph properties. In Huisman and Rubin, editors, *FASE*, volume 10202 of *LNCS*, pages 226–243. Springer, 2017.
- [279] B. Schölkopf, J. Platt, and T. Hofmann. *Learning with Hypergraphs: Clustering, Classification, and Embedding*, pages 1601–1608. MIT Press, 2007.
- [280] H. Schölzel, H. Ehrig, M. Maximova, K. Gabriel, and F. Hermann. Satisfaction, restriction, and amalgamation of constraints in the framework of \mathcal{M} -adhesive categories. In Golas and Soboll, editors, *ACCAT*, volume 93 of *EPTCS*, pages 83–104, 2012.
- [281] A. Schürr. Specification of graph translators with triple graph grammars. In Mayr, Schmidt, et al., editors, *GTCCS*, volume 903 of *LNCS*, pages 151–163. Springer, 1994.
- [282] A. Schürr, A. J. Winter, and A. Zündorf. Graph grammar engineering with PROGRES. In Schäfer and Botella, editors, *Sitges*, volume 989 of *LNCS*, pages 219–234. Springer, 1995.
- [283] A. Schürr, A. J. Winter, and A. Zündorf. The PROGRES approach: Language and environment. In Ehrig et al. [72], pages 487–550.
- [284] J. Staples. A graph-like Lambda calculus for which leftmost-overmost reduction is optimal. In Claus et al. [43], pages 440–455.
- [285] P. Sylopp. Konzeption und Implementierung von Analysetechniken für rekonfigurierbare Objektnetze. Diploma thesis, Technische Universität Berlin, 2009.
- [286] I. Szücs, G. Gombos, and A. Kiss. Five Ws, one H, and many tweets. In *CogInfoCom*, pages 441–446, 2013.
- [287] G. Taentzer. AGG: A graph transformation environment for modeling and validation of software. In Pfaltz, Nagl, et al., editors, *AGTIVE*, volume 3062 of *LNCS*, pages 446–453. Springer, 2003.
- [288] *Petri Net Toolbox for MATLAB*. Technical University Gh. Asachi of Iasi, Romania, 2016. URL <http://www.pntool.ac.tuiasi.ro>.
- [289] *Charlie*. Technische Universität Cottbus, 2015. URL <http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Charlie>.

- [290] *SyGrAV—Symbolic Graph Analysis and Verification*. Technische Universität Darmstadt, 2017. URL <https://github.com/eMoflon/emoflon-sygrav>.
- [291] *Reconfigurable Object Nets Environment (RON)*. TFS-Group, Technische Universität Berlin, 2007. URL <http://www.tfs.cs.tu-berlin.de/roneditor>.
- [292] *AGG 2.0*. TFS-Group, Technische Universität Berlin, 2011. URL <http://tfs.cs.tu-berlin.de/agg>.
- [293] *The Reference Net Workshop 2.5 (Renew 2.5)*. Theoretical Foundations Group, Universität Hamburg, 2016. URL <http://www.renew.de>.
- [294] J. J. P. Tsai, S. J. Yang, and Y.-H. Chang. Timing constraint Petri nets and their application to schedulability analysis of real-time system specifications. *IEEE Transactions on Software Engineering*, 21(1):32–49, 1995.
- [295] E. Tuosto. *Non-Functional Aspects of Wide Area Network Programming*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 2003.
- [296] *Augur 2*. Universität Duisburg-Essen, 2008. URL <http://www.ti.inf.uni-due.de/en/research/tools/augur2>.
- [297] *Programming Environment Based on Petri Nets 2.0 (PEP 2.0)*. Universität Oldenburg, 2004. URL <http://theoretica.informatik.uni-oldenburg.de/~pep>.
- [298] *Graphs for Object-Oriented Verification (GROOVE)*. University of Twente, 2011. URL <http://groove.cs.utwente.nl>.
- [299] W. M. P. v. d. Aalst and K. M. v. Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.
- [300] R. J. v. Glabbeek. The individual and collective token interpretations of Petri nets. In Abadi and Alfaro, editors, *CONCUR*, volume 3653 of *LNCS*, pages 323–337. Springer, 2005.
- [301] R. v. Landeghem and C. Bobeanu. Formal modelling of supply chain: An incremental approach using Petri nets. In *ESS*, pages 323–327, 2002.
- [302] A. Vandin and A. Lluch-Lafuente. Towards a Maude tool for model checking temporal graph properties. *ECEASST*, 41, 2011.
- [303] D. Varró, G. Varró, and A. Pataricza. Designing the automatic transformation of visual languages. *Sci. Comput. Program.*, 44(2):205–227, 2002.
- [304] D. Varró, S. Varró-Gyapay, H. Ehrig, U. Prange, and G. Taentzer. Termination analysis of model transformations by Petri nets. In Corradini et al. [47], pages 260–274.
- [305] *VERIGRAPH: Software Specification and Verification Tool Based on Graph Rewriting*. Verites-Group, Universidade Federal do Rio Grande do Sul, 2016. URL <https://github.com/Verites/verigraph>.
- [306] *Resource Description Framework (RDF)*. W3-Consortium, 2009. URL <http://www.w3.org/RDF/>.
- [307] J. Wang. Charging information collection modeling and analysis of GPRS networks. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 37(4):473–481, 2007.
- [308] J. Wang. Petri nets for dynamic event-driven system modeling. In Fishwick, editor, *Handbook of Dynamic System Modeling*. Chapman and Hall/CRC, 2007.
- [309] M. Weber and E. Kindler. The Petri net kernel. In Ehrig, Reisig, et al., editors, *Petri Net Technology for Communication-Based Systems—Advances in Petri Nets*, volume 2472 of *LNCS*, pages 109–124. Springer, 2003.

Part VII

APPENDICES

BASICS OF CATEGORY THEORY

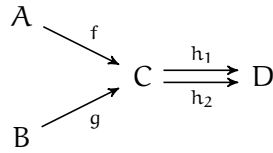
In this appendix, we recall some basic categorical definitions as well as give the detailed proofs for some technical characteristics from the main part of this work.

For some categorical constructions we need two morphisms to be jointly epimorphic. Intuitively, this is the case if two morphisms together behave similar to an epimorphism, i.e., the object C can be considered as a suitable gluing of objects A and B (see the diagram in the definition below). The formal definition of this notion (see e.g. [277]) is given in the following.

Definition 71 (Jointly Epimorphic Morphisms [277]).

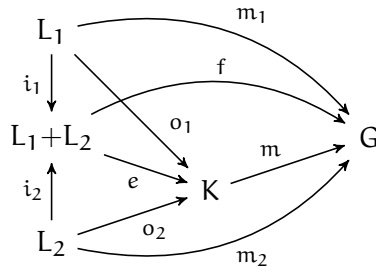
A pair $(f : A \rightarrow C, g : B \rightarrow C)$ of morphisms is called jointly epimorphic iff the following holds:

$$\forall h_1, h_2 : C \rightarrow D. (h_1 \circ f = h_2 \circ f \wedge h_1 \circ g = h_2 \circ g) \Rightarrow (h_1 = h_2)$$



Lemma 1: ($\mathcal{E}' - \mathcal{M}$ Pair Factorization Based on $\mathcal{E} - \mathcal{M}$ -Factorization, see page 27)

Consider a category \mathcal{C} with $\mathcal{E} - \mathcal{M}$ -factorizations, as given in Definition 10, and binary coproducts. Let furthermore $(m_1 : L_1 \rightarrow G, m_2 : L_2 \rightarrow G)$ be a morphism pair with the common codomain, $(L_1 + L_2, i_1 : L_1 \rightarrow L_1 + L_2, i_2 : L_2 \rightarrow L_1 + L_2)$ be a binary coproduct of (L_1, L_2) with induced coproduct morphism $f : L_1 + L_2 \rightarrow G$ and $L_1 + L_2 \xrightarrow{e} K \xrightarrow{m} G$ be an $\mathcal{E} - \mathcal{M}$ -factorization of the morphism f as given in Definition 11. Then $((o_1, o_2), m)$ with $o_1 = e \circ i_1$ and $o_2 = e \circ i_2$ is an $\mathcal{E}' - \mathcal{M}$ pair factorization of (m_1, m_2) .



Proof.

According to Definition 11, $((o_1, o_2), m)$ is an $\mathcal{E}' - \mathcal{M}$ pair factorization of (m_1, m_2) if there are a unique up to isomorphism object K and morphisms $m : K \rightarrow G$, $o_j : L_j \rightarrow K$ for $j \in \{1, 2\}$ such that $(o_1, o_2) \in \mathcal{E}'$, $m \in \mathcal{M}$ and $m \circ o_j = m_j$ for $j \in \{1, 2\}$. By $\mathcal{E} - \mathcal{M}$ -factorization of f according to Definition 10, we already have that $m \in \mathcal{M}$. Moreover, by coproduct construction and $\mathcal{E} - \mathcal{M}$ -factorization of f we have that $m_j \stackrel{\text{univ. prop. of coprod.}}{=} f \circ i_j \stackrel{\mathcal{E} - \mathcal{M}\text{-fact. of } f}{=} m \circ e \circ i_j \stackrel{\text{Def. } o_j}{=} m \circ o_j$ for

$j \in \{1, 2\}$. To get that $(o_1, o_2) \in \mathcal{E}'$, we have to show that the morphisms (o_1, o_2) are jointly epimorphic. According to Definition 71 this formally means the following (see the diagram below to the left):

$$\forall h_1, h_2 : K \rightarrow X. ((h_1 \circ o_1 = h_2 \circ o_1) \wedge (h_1 \circ o_2 = h_2 \circ o_2)) \Rightarrow h_1 = h_2$$

Fix $h_1, h_2 : K \rightarrow X$ such that holds $h_1 \circ o_1 = h_2 \circ o_1$ and $h_1 \circ o_2 = h_2 \circ o_2$. Then we have the following:

$$\begin{aligned} & (h_1 \circ o_1 = h_2 \circ o_1) \wedge (h_1 \circ o_2 = h_2 \circ o_2) \\ & \xRightarrow{\text{Def. } o_1, o_2} (h_1 \circ e \circ i_1 = h_2 \circ e \circ i_1) \wedge (h_1 \circ e \circ i_2 = h_2 \circ e \circ i_2) \\ & \xRightarrow{(i_1, i_2) \in \mathcal{E}'} h_1 \circ e = h_2 \circ e \\ & \xRightarrow{e \in \mathcal{E}} h_1 = h_2 \end{aligned}$$

It remains to show that the object K of the considered $\mathcal{E}' - \mathcal{M}$ pair factorization is unique up to isomorphism. For this reason, we first assume that there are another object K' and morphisms $m' : K' \rightarrow G$, $o'_1 : L_1 \rightarrow K'$, $o'_2 : L_2 \rightarrow K'$ for which it holds that $(o'_1, o'_2) \in \mathcal{E}'$, $m' \in \mathcal{M}$ and $m' \circ o'_j = m_j$ for $j \in \{1, 2\}$ (see the diagram below to the right). Furthermore, there is an induced morphism $e' : L_1 + L_2 \rightarrow K'$ such that $o'_j = e' \circ i_j$ holds for $j \in \{1, 2\}$ by the coproduct property. We have to show that $K \cong K'$. It holds the following:

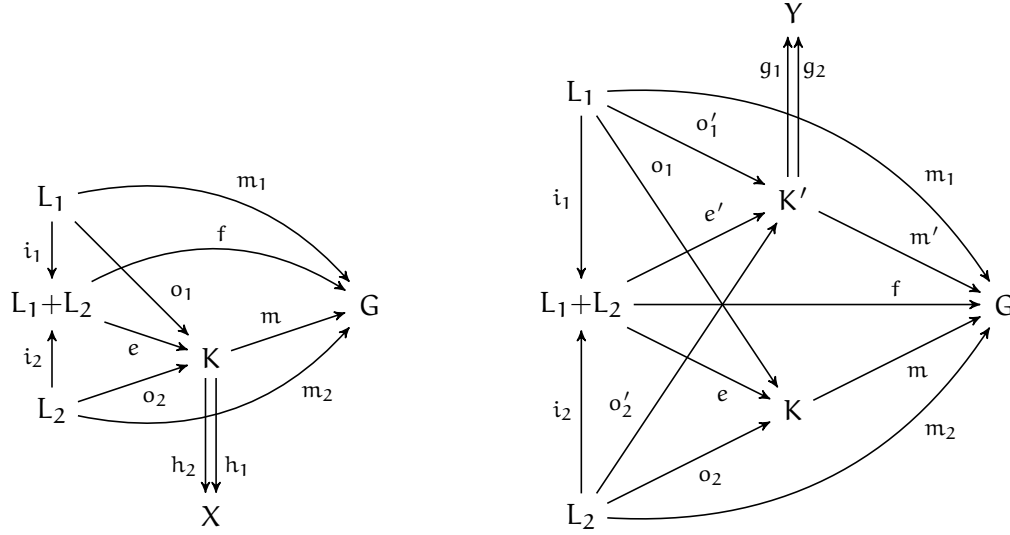
$$\begin{aligned} & (m' \circ o'_1 = m_1) \wedge (m' \circ o'_2 = m_2) \\ & \xRightarrow{m_j = m \circ o_j \text{ for } j \in \{1, 2\}} (m' \circ o'_1 = m \circ o_1) \wedge (m' \circ o'_2 = m \circ o_2) \\ & \xRightarrow{\text{Def. } o_1, o_2} (m' \circ o'_1 = m \circ e \circ i_1) \wedge (m' \circ o'_2 = m \circ e \circ i_2) \\ & \xRightarrow{\text{Def. } o'_1, o'_2} (m' \circ e' \circ i_1 = m \circ e \circ i_1) \wedge (m' \circ e' \circ i_2 = m \circ e \circ i_2) \\ & \xRightarrow{(i_1, i_2) \in \mathcal{E}'} m' \circ e' = m \circ e \\ & \xRightarrow{\mathcal{E} - \mathcal{M} \text{-fact. of } f} m' \circ e' = f \end{aligned}$$

By assumption, we know that $m' \in \mathcal{M}$ and for arbitrary morphisms $g_1, g_2 : K' \rightarrow Y$ (see the diagram below to the right) we have that:

$$\begin{aligned} & g_1 \circ e' = g_2 \circ e' \\ & \Rightarrow (g_1 \circ e' \circ i_1 = g_2 \circ e' \circ i_1) \wedge (g_1 \circ e' \circ i_2 = g_2 \circ e' \circ i_2) \\ & \xRightarrow{\text{Def. } o'_1, o'_2} (g_1 \circ o'_1 = g_2 \circ o'_1) \wedge (g_1 \circ o'_2 = g_2 \circ o'_2) \\ & \xRightarrow{(o'_1, o'_2) \in \mathcal{E}'} g_1 = g_2 \end{aligned}$$

Thus, $e' \in \mathcal{E}$ implying that $m' \circ e'$ is also an $\mathcal{E} - \mathcal{M}$ -factorization of f according to Definition 10. Since $\mathcal{E} - \mathcal{M}$ -factorizations are unique up to isomorphism, we have that $K \cong K'$.

1 From the universal property of binary coproducts we can deduce that (i_1, i_2) are jointly epimorphic.



□

For some applications we need special kinds of pair factorization construction e.g. as given in the lemma below.

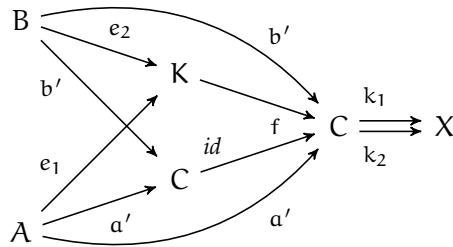
Lemma 66 (A Special Case of Pair Factorization).

Consider a category \mathbf{C} with \mathcal{E}' - \mathcal{M} pair factorizations and morphisms $a' : A \rightarrow C$, $b' : B \rightarrow C$ in $\text{Mor}_{\mathbf{C}}$. Then $((a', b'), id)$ is an \mathcal{E}' - \mathcal{M} pair factorization of (a', b') if (a', b') are jointly epimorphic.

Proof.

According to Definition 11, we have to show that there is a unique up to isomorphism object $C \in \text{Ob}_{\mathbf{C}}$ such that $(a', b') \in \mathcal{E}'$, $id \in \mathcal{M}$ and $id \circ a' = a'$, $id \circ b' = b'$. All mentioned properties except the uniqueness hold trivially. For the uniqueness of C , we have to show that for another object $K \in \text{Ob}_{\mathbf{C}}$ with morphisms $e_1 : A \rightarrow K$, $e_2 : B \rightarrow K$, $f : K \rightarrow C$ such that it holds $(e_1, e_2) \in \mathcal{E}'$, $f \in \mathcal{M}$, $f \circ e_1 = a'$, $f \circ e_2 = b'$, we have that $K \cong C$. $K \cong C$ holds if the morphism f is a monomorphism and epimorphism at once. By assumption we already have that $f \in \mathcal{M}$ and hence a monomorphism. It remains to show that f is an epimorphism, i.e., $\forall k_1, k_2 : C \rightarrow X. (k_1 \circ f = k_2 \circ f) \Rightarrow (k_1 = k_2)$. Fix $k_1, k_2 \in \text{Mor}_{\mathbf{C}}$. Then we have the following:

$$\begin{aligned}
 & k_1 \circ f = k_2 \circ f \\
 \Rightarrow & k_1 \circ f \circ e_1 = k_2 \circ f \circ e_1 \wedge k_1 \circ f \circ e_2 = k_2 \circ f \circ e_2 \\
 \stackrel{f \circ e_1 = a'}{\Rightarrow} & k_1 \circ a' = k_2 \circ a' \wedge k_1 \circ f \circ e_2 = k_2 \circ f \circ e_2 \\
 \stackrel{f \circ e_2 = b'}{\Rightarrow} & k_1 \circ a' = k_2 \circ a' \wedge k_1 \circ b' = k_2 \circ b' \\
 \stackrel{\text{Def. 71}}{\Rightarrow} & k_1 = k_2
 \end{aligned}$$



□

For the verification of local confluence we need a suitable decomposition property for pushouts consisting of a pullback and special morphisms in \mathcal{M} and \mathcal{M}' as assumption for the Completeness of Critical Pairs Lemma (see Fact 2). According to [88] this technical property is called $\mathcal{M} - \mathcal{M}'$ pushout-pullback decomposition property.

Definition 72 ($\mathcal{M} - \mathcal{M}'$ Pushout-Pullback Decomposition Property [88]).

An \mathcal{M} -adhesive category $(\mathbf{C}, \mathcal{M})$ with a morphism class \mathcal{M}' has the $\mathcal{M} - \mathcal{M}'$ pushout-pullback decomposition property if the following property holds: Given the following commutative diagram with $l \in \mathcal{M}$ and $w \in \mathcal{M}'$, where (1) + (2) is a pushout and (2) a pullback, then (1) and (2) are pushouts and also pullbacks.

$$\begin{array}{ccccc}
 A & \xrightarrow{k} & B & \xrightarrow{r} & E \\
 l \downarrow & & \downarrow s & & \downarrow v \\
 C & \xrightarrow{u} & D & \xrightarrow{w} & F
 \end{array}
 \quad
 \begin{array}{ccc}
 & (1) & \\
 & (2) &
 \end{array}$$

According to [88], an extension diagram describes how a transformation $t : G_0 \xRightarrow{*} G_n$ can be extended to a transformation $t' : G'_0 \xRightarrow{*} G'_n$ via an extension morphism $k_0 : G_0 \rightarrow G'_0$ mapping G_0 to G'_0 .

Definition 73 (Extension Diagram [88]).

An extension diagram is a diagram (1), where $k_0 : G_0 \rightarrow G'_0$ is a morphism, called an extension morphism, and $t : G_0 \xRightarrow{*} G_n$, $t' : G'_0 \xRightarrow{*} G'_n$ are transformations via the same productions (p_0, \dots, p_{n-1}) and matches (m_0, \dots, m_{n-1}) , $(k_0 \circ m_0, \dots, k_{n-1} \circ m_{n-1})$ respectively, defined by the following double pushout diagrams:

$$\begin{array}{ccc}
 G_0 & \xRightarrow{t} & G_n \\
 k_0 \downarrow & (1) & \downarrow k_n \\
 G'_0 & \xRightarrow{t'} & G'_n
 \end{array}
 \quad
 p_i : \begin{array}{ccccc}
 L_i & \xleftarrow{l_i} & K_i & \xrightarrow{r_i} & R_i \\
 m_i \downarrow & & \downarrow j_i & & \downarrow n_i \\
 G_i & \xleftarrow{f_i} & D_i & \xrightarrow{g_i} & G_{i+1} \\
 k_i \downarrow & & \downarrow d_i & & \downarrow k_{i+1} \\
 G'_i & \xleftarrow{f'_i} & D'_i & \xrightarrow{g'_i} & G'_{i+1}
 \end{array}$$

$i \in \{0, \dots, n-1\}, n > 0$

For $n = 0$, the extension diagram is given up to isomorphism by

$$\begin{array}{ccccc}
 G_0 & \xleftarrow{id_{G_0}} & G_0 & \xrightarrow{id_{G_0}} & G_0 \\
 k_0 \downarrow & & \downarrow k_0 & & \downarrow k_0 \\
 G'_0 & \xleftarrow{id'_{G_0}} & G'_0 & \xrightarrow{id'_{G_0}} & G'_0
 \end{array}$$

In order to apply a rule via a match morphism to some typed attributed graph, we have to check whether the gluing condition is satisfied. The gluing condition can be formulated on the abstract level of \mathcal{M} -adhesive transformation systems using initial pushouts as already introduced in Subsection 2.2.1 or, alternatively, by the following more constructive definition in the set-based notation. These two concepts for gluing condition definition given in Definition 74 below and Fact 8 from Subsection 2.4.1 are equivalent as already shown in [88].

Definition 74 (Gluing Condition in $\mathbf{AGraphs}_{\text{ATG}}$ [88]).

Consider a typed attributed graph rule $\rho = (L \xleftarrow{l} K \xrightarrow{r} R)$ with injective morphisms l, r in $\text{Mor}_{\mathbf{AGraphs}_{\text{ATG}}}$, a typed attributed graph G and a match morphism $m : L \rightarrow G$ in $\text{Mor}_{\mathbf{AGraphs}_{\text{ATG}}}$ with $X = (((V_G^X, V_D^X = \mathbb{N}, E_G^X, E_{NA}^X, E_{EA}^X, (s_j^X, t_j^X)_{j \in \{G, NA, EA\}}), \text{NAT}), \text{type}^X)$ and $\text{type}^X : ((V_G^X, V_D^X = \mathbb{N}, E_G^X, E_{NA}^X, E_{EA}^X, (s_j^X, t_j^X)_{j \in \{G, NA, EA\}}), \text{NAT}) \rightarrow \text{ATG}$ for $X \in \{L, K, R, G\}$. Then we can state the following definitions:

- The gluing points GP are those graph items in L that are not deleted by the rule ρ , i.e., $GP = l_{V_G}(V_G^K) \cup l_{E_G}(E_G^K) \cup l_{E_{NA}}(E_{NA}^K) \cup l_{E_{EA}}(E_{EA}^K)$.
- The identification points IP are those graph items in L that are identified by the match morphism m , i.e., $IP = IP_{V_G} \cup IP_{E_G} \cup IP_{E_{NA}} \cup IP_{E_{EA}}$, where

$$IP_{V_G} = \{a \in V_G^L \mid \exists a' \in V_G^L \wedge a \neq a' \wedge m_{V_G}(a) = m_{V_G}(a')\},$$

$$IP_{E_j} = \{a \in E_j^L \mid \exists a' \in E_j^L \wedge a \neq a' \wedge m_{E_j}(a) = m_{E_j}(a')\} \text{ for } j \in \{G, NA, EA\}.$$

- The dangling points DP are those graph items in L , whose images are the source or the target of an item (see Definition 30) that does not belong to $m(L)$, i.e., $DP = DP_{V_G} \cup DP_{E_G}$, where

$$DP_{V_G} = \{a \in V_G^L \mid (\exists a' \in (E_{NA}^G \setminus m_{E_{NA}}(E_{NA}^L)). m_{V_G}(a) = s_{NA}^G(a'))$$

$$\vee (\exists a' \in (E_G^G \setminus m_{E_G}(E_G^L)). (m_{V_G}(a) = s_G^G(a')) \vee (m_{V_G}(a) = t_G^G(a'))))\}$$

$$DP_{E_G} = \{a \in E_G^L \mid \exists a' \in (E_{EA}^G \setminus m_{E_{EA}}(E_{EA}^L)). m_{E_G}(a) = s_{EA}^G(a')\}$$

DETAILED PROOFS FOR GENERAL THEORY

In this appendix, we give the detailed proofs for indicated lemmas from the main part of this work concerning our general theory.

Lemma 8: (\mathcal{F} Creates Identities and Isomorphisms, see [page 83](#))

Consider \mathcal{M} -adhesive categories $(\mathbf{C}_1, \mathcal{M}_1)$ and $(\mathbf{C}_2, \mathcal{M}_2)$. Then an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ creates identities and isomorphisms if \mathcal{F} creates morphisms.

Proof.

- **\mathcal{F} creates identities:**

We have to show the following: $\forall (m' : \mathcal{F}(L) \rightarrow \mathcal{F}(G)) \in \text{Mor}_{\mathbf{C}_2}. (m' \text{ identity} \Rightarrow \exists!(m : L \rightarrow G) \in \text{Mor}_{\mathbf{C}_1}. \mathcal{F}(m) = m' \wedge m \text{ identity})$.

Fix $m' \in \text{Mor}_{\mathbf{C}_2}$ and assume the premise of the statement. It remains to show the conclusion of the statement.

The fact that m' is an identity morphism implies that $m' = \text{id}_{\mathcal{F}(L)} = \text{id}_{\mathcal{F}(G)}$ with $\mathcal{F}(L) = \mathcal{F}(G)$. Let us now consider without loss of generality that m' is of the type $\mathcal{F}(L) \rightarrow \mathcal{F}(L)$. Since \mathcal{F} creates morphisms by assumption, we have that there is a unique morphism $m : L \rightarrow L$ in $\text{Mor}_{\mathbf{C}_1}$ such that $\mathcal{F}(m) = m'$. We now want to show that m is an identity morphism, i.e., $m = \text{id}_L$. We have the following:

$$\mathcal{F}(\text{id}_L) \stackrel{\text{funct. prop.}}{=} \text{id}_{\mathcal{F}(L)} \stackrel{m' \text{ ident.}}{=} m' \stackrel{m' \text{ creat. of } m}{=} \mathcal{F}(m)$$

This implies by uniqueness of creation of m that $m = \text{id}_L$.

- **\mathcal{F} creates isomorphisms:**

We have to show the following: $\forall (m' : \mathcal{F}(L) \rightarrow \mathcal{F}(G)) \in \text{Mor}_{\mathbf{C}_2}. (m' \text{ isomorphism} \Rightarrow \exists!(m : L \rightarrow G) \in \text{Mor}_{\mathbf{C}_1}. \mathcal{F}(m) = m' \wedge m \text{ isomorphism})$.

Fix $m' \in \text{Mor}_{\mathbf{C}_2}$ and assume the premise of the statement. It remains to show the conclusion of the statement.

The fact that m' is an isomorphism implies by definition that m' is a retraction and coretraction, i.e., there is a morphism $\overline{m}' : \mathcal{F}(G) \rightarrow \mathcal{F}(L)$ in $\text{Mor}_{\mathbf{C}_2}$ such that $m' \circ \overline{m}' = \text{id}_{\mathcal{F}(G)}$ and $\overline{m}' \circ m' = \text{id}_{\mathcal{F}(L)}$ (see the diagram below). Since \mathcal{F} creates morphisms by assumption, there are unique morphisms $m : L \rightarrow G$ and $\overline{m} : G \rightarrow L$ in $\text{Mor}_{\mathbf{C}_1}$ such that $\mathcal{F}(m) = m'$, $\mathcal{F}(\overline{m}) = \overline{m}'$. Thus, we have exactly one morphism $m : L \rightarrow G$ satisfying $\mathcal{F}(m) = m'$.

It remains to show that m is an isomorphism, i.e., there is a morphism $\overline{m} : G \rightarrow L$ in $\text{Mor}_{\mathbf{C}_1}$ such that $m \circ \overline{m} = \text{id}_G$ and $\overline{m} \circ m = \text{id}_L$ (see the diagram below).

Let \overline{m} be the unique morphism from the morphism creation property of \overline{m}' as given before. Since \mathcal{F} creates identity morphisms according to the first part of this lemma, we have that for the identity morphism $\text{id}_{\mathcal{F}(G)}$ there is a unique morphism id_G in $\text{Mor}_{\mathbf{C}_1}$ which is the identity morphism satisfying the property $\mathcal{F}(\text{id}_G) = \text{id}_{\mathcal{F}(G)}$. But it holds as well the following:

$$\mathcal{F}(m \circ \overline{m}) \stackrel{\text{funct. prop.}}{=} \mathcal{F}(m) \circ \mathcal{F}(\overline{m}) \stackrel{\text{creat. of } m, \overline{m}}{=} m' \circ \overline{m}' \stackrel{m' \text{ isom.}}{=} \text{id}_{\mathcal{F}(G)}$$

This implies by uniqueness of creation of id_G that $m \circ \bar{m} = \text{id}_G$.

Similarly, using the identity morphism creation property for $\text{id}_{\mathcal{F}(L)}$, we get that $\bar{m} \circ m = \text{id}_L$. Thus, we have that m is an isomorphism.

$$\begin{array}{ccc}
 \text{id}_L & & \text{id}_{\mathcal{F}(L)} \\
 \downarrow & & \downarrow \\
 L & & \mathcal{F}(L) \\
 \bar{m} \uparrow & \searrow & \bar{m}' \uparrow \\
 & G & \mathcal{F}(G) \\
 & \downarrow & \downarrow \\
 & \text{id}_G & \text{id}_{\mathcal{F}(G)}
 \end{array}$$

□

Lemma 9: (\mathcal{F} is Injective on Objects, see page 83)

Consider \mathcal{M} -adhesive categories $(\mathbf{C}_1, \mathcal{M}_1)$ and $(\mathbf{C}_2, \mathcal{M}_2)$. Then an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ is injective on objects, i.e.,

$$\forall H_1, H_2 \in \text{Ob}_{\mathbf{C}_1}. (\mathcal{F}(H_1) = \mathcal{F}(H_2)) \Rightarrow (H_1 = H_2)$$

if \mathcal{F} creates morphisms.

Proof.

Fix $H_1, H_2 \in \text{Ob}_{\mathbf{C}_1}$ and assume that $\mathcal{F}(H_1) = \mathcal{F}(H_2)$. This means that there is an identity morphism $\text{id}' : \mathcal{F}(H_1) \rightarrow \mathcal{F}(H_2)$. Using the fact that \mathcal{F} creates identities from Lemma 8, we get the unique identity morphism $\text{id} : H_1 \rightarrow H_2$ such that $\mathcal{F}(\text{id}) = \text{id}'$, which implies that $H_1 = H_2$. □

Lemma 10: (\mathcal{F} is Injective on Morphisms, see page 83)

Consider \mathcal{M} -adhesive categories $(\mathbf{C}_1, \mathcal{M}_1)$ and $(\mathbf{C}_2, \mathcal{M}_2)$. Then an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ is injective on morphisms, i.e.,

$$\forall (\mathcal{F}(m), \mathcal{F}(n) : \mathcal{F}(G) \rightarrow \mathcal{F}(H)) \in \text{Mor}_{\mathbf{C}_2}. (\mathcal{F}(m) = \mathcal{F}(n)) \Rightarrow (m = n)$$

if \mathcal{F} creates morphisms.

Proof.

Fix $\mathcal{F}(m), \mathcal{F}(n) : \mathcal{F}(G) \rightarrow \mathcal{F}(H) \in \text{Mor}_{\mathbf{C}_2}$ and assume that $\mathcal{F}(m) = \mathcal{F}(n)$. Let now the morphism m be of the type $X_1 \rightarrow X_2$. Applying \mathcal{F} to this morphism m , we get the morphism $\mathcal{F}(m)$ of the type $\mathcal{F}(X_1) \rightarrow \mathcal{F}(X_2)$. Furthermore, according to the assumption we have that $\mathcal{F}(m)$ is of the type $\mathcal{F}(G) \rightarrow \mathcal{F}(H)$, which implies that $\mathcal{F}(X_1) = \mathcal{F}(G)$ and $\mathcal{F}(X_2) = \mathcal{F}(H)$. Now we can apply Lemma 9 and get that $X_1 = G$, $X_2 = H$ implying that m is of the type $G \rightarrow H$. For the morphism n , we can similarly show that n is of the type $G \rightarrow H$ as well. Since \mathcal{F} creates morphisms by assumption, we furthermore have that there is a unique morphism $f : G \rightarrow H$ such that $\mathcal{F}(f) = \mathcal{F}(m)$. Both morphisms $m, n : G \rightarrow H$ satisfy this property, because $\mathcal{F}(m) = \mathcal{F}(n)$ by assumption, which implies by uniqueness of creation of f that $m = n$. □

Lemma 11: (Satisfaction of Nested Application Conditions [213], see page 90)

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $\text{AS}_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (\mathcal{M}) -morphisms. Then a morphism $p : P \rightarrow G$ satisfies a nested application condition ac_P in AS_1 iff the corresponding morphism $\mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G)$ satisfies the nested application condition $\mathcal{F}(\text{ac}_P)$ in AS_2 .

Proof (by Induction over the Depth of a Nested Application Condition).**Basis :**

Let $ac_P = \text{true}$. Every morphism in C_1 satisfies true . Every morphism in C_2 satisfies $\mathcal{F}(\text{true}) = \text{true}$.

Induction Hypothesis :

- For some ac_C over the object C holds: $\forall p' : C \rightarrow G \in \text{Mor}_{C_1}. p' \models ac_C \Leftrightarrow \mathcal{F}(p') \models \mathcal{F}(ac_C)$
- For some $ac_{P,i}$ ($i \in \mathcal{I}$) over the object P holds: $\forall p : P \rightarrow G \in \text{Mor}_{C_1}. p \models ac_{P,i} \Leftrightarrow \mathcal{F}(p) \models \mathcal{F}(ac_{P,i})$

It remains to show:

1. $\forall a : P \rightarrow C \in \text{Mor}_{C_1}, p : P \rightarrow G \in \text{Mor}_{C_1}. p \models \exists(a, ac_C) \Leftrightarrow \mathcal{F}(p) \models \mathcal{F}(\exists(a, ac_C))$
2. $\forall p : P \rightarrow G \in \text{Mor}_{C_1}. p \models \neg ac_P \Leftrightarrow \mathcal{F}(p) \models \mathcal{F}(\neg ac_P)$
3. $\forall p : P \rightarrow G \in \text{Mor}_{C_1}. p \models \bigwedge_{i \in \mathcal{I}} ac_{P,i} \Leftrightarrow \mathcal{F}(p) \models \mathcal{F}(\bigwedge_{i \in \mathcal{I}} ac_{P,i})$
4. $\forall p : P \rightarrow G \in \text{Mor}_{C_1}. p \models \bigvee_{i \in \mathcal{I}} ac_{P,i} \Leftrightarrow \mathcal{F}(p) \models \mathcal{F}(\bigvee_{i \in \mathcal{I}} ac_{P,i})$

Induction Step :

1. Fix $a : P \rightarrow C \in \text{Mor}_{C_1}$ and $p : P \rightarrow G \in \text{Mor}_{C_1}$.

a) (\Rightarrow) :

$$p : P \rightarrow G \models \exists(a, ac_C) \\ \xRightarrow{\text{Def. 16}} \exists q : C \rightarrow G \in \mathcal{M}_1. q \circ a = p \wedge q \models ac_C$$

Fix $q : C \rightarrow G$ s.t. $q \circ a = p \wedge q \models ac_C$.

Then $q \models ac_C \xRightarrow{\text{Ind. H.}} \mathcal{F}(q) \models \mathcal{F}(ac_C)$

$$\xRightarrow{\text{pres. of com. diag.}} \exists \mathcal{F}(q) : \mathcal{F}(C) \rightarrow \mathcal{F}(G) \in \mathcal{M}_2. \mathcal{F}(q) \circ \mathcal{F}(a) = \mathcal{F}(p) \wedge$$

$$\mathcal{F}(q) \models \mathcal{F}(ac_C)$$

$$\Rightarrow \exists q' : \mathcal{F}(C) \rightarrow \mathcal{F}(G) = \mathcal{F}(q) \in \mathcal{M}_2. q' \circ \mathcal{F}(a) = \mathcal{F}(p) \wedge q' \models \mathcal{F}(ac_C)$$

$$\xRightarrow{\text{Rem. 11}} \mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G) \models \exists(\mathcal{F}(a), \mathcal{F}(ac_C))$$

$$\xRightarrow{\text{Def. 47}} \mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G) \models \mathcal{F}(\exists(a, ac_C))$$

$$\begin{array}{ccc} \mathcal{F}(ac_P) \triangleright \mathcal{F}(P) & \xrightarrow{\mathcal{F}(a)} & \mathcal{F}(C) \triangleleft \mathcal{F}(ac_C) \\ & \searrow \mathcal{F}(p) & \swarrow \mathcal{F}(q) = q' \in \mathcal{M}_2 \\ & \mathcal{F}(G) & \end{array}$$

b) (\Leftarrow) :

$$\mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G) \models \mathcal{F}(\exists(a, ac_C))$$

$$\xRightarrow{\text{Def. 47}} \mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G) \models \exists(\mathcal{F}(a), \mathcal{F}(ac_C))$$

$$\xRightarrow{\text{Rem. 11}} \exists q' : \mathcal{F}(C) \rightarrow \mathcal{F}(G) \in \mathcal{M}_2. q' \circ \mathcal{F}(a) = \mathcal{F}(p) \wedge q' \models \mathcal{F}(\text{ac}_C)$$

Fix $q' : \mathcal{F}(C) \rightarrow \mathcal{F}(G) \in \mathcal{M}_2$ s.t. $q' \circ \mathcal{F}(a) = \mathcal{F}(p) \wedge q' \models \mathcal{F}(\text{ac}_C)$.

Then $\exists! q : C \rightarrow G \in \mathcal{M}_1. \mathcal{F}(q) = q'$ since \mathcal{F} creates \mathcal{M} -morphisms.

$$\xRightarrow{\mathcal{F} \text{ prop.}} \exists q : C \rightarrow G \in \mathcal{M}_1. \mathcal{F}(q) \circ \mathcal{F}(a) = \mathcal{F}(q \circ a) = \mathcal{F}(p) \wedge \mathcal{F}(q) \models \mathcal{F}(\text{ac}_C)$$

$$\xRightarrow{\mathcal{F} \text{ creat. morph.} + \text{Lem. 10}} \exists q : C \rightarrow G \in \mathcal{M}_1. q \circ a = p \wedge \mathcal{F}(q) \models \mathcal{F}(\text{ac}_C)$$

$$\xRightarrow{\text{Ind. H.}} \exists q : C \rightarrow G \in \mathcal{M}_1. q \circ a = p \wedge q \models \text{ac}_C$$

$$\xRightarrow{\text{Def. 16}} p : P \rightarrow G \models \exists(a, \text{ac}_C)$$

$$\begin{array}{ccc} \text{ac}_P \triangleright P & \xrightarrow{a} & C \triangleleft \text{ac}_C \\ & \searrow p & \swarrow q \in \mathcal{M}_1 \\ & G & \end{array} \quad \begin{array}{c} \\ \\ = \end{array}$$

2. Fix $p : P \rightarrow G \in \text{Mor}_{C_1}$.

$$p : P \rightarrow G \models \neg \text{ac}_P$$

$$\xLeftrightarrow{(*)} \neg(p : P \rightarrow G \models \text{ac}_P)$$

$$\xLeftrightarrow{\text{Ind. H.}} \neg(\mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G) \models \mathcal{F}(\text{ac}_P))$$

$$\xLeftrightarrow{(*)} \mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G) \models \neg \mathcal{F}(\text{ac}_P)$$

$$\xLeftrightarrow{\text{Def. 47}} \mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G) \models \mathcal{F}(\neg \text{ac}_P)$$

(*): The compatibility of \models with \neg, \wedge, \vee is given for morphisms by definition.

3. Fix $p : P \rightarrow G \in \text{Mor}_{C_1}$.

$$p : P \rightarrow G \models \bigwedge_{i \in \mathcal{I}} \text{ac}_{P,i}$$

$$\xLeftrightarrow{\text{Def. 16}} \bigwedge_{i \in \mathcal{I}} (p \models \text{ac}_{P,i})$$

$$\xLeftrightarrow{\text{Ind. H.}} \bigwedge_{i \in \mathcal{I}} (\mathcal{F}(p) \models \mathcal{F}(\text{ac}_{P,i}))$$

$$\xLeftrightarrow{\text{Def. 16}} \mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G) \models \bigwedge_{i \in \mathcal{I}} \mathcal{F}(\text{ac}_{P,i})$$

$$\xLeftrightarrow{\text{Def. 47}} \mathcal{F}(p) : \mathcal{F}(P) \rightarrow \mathcal{F}(G) \models \mathcal{F}\left(\bigwedge_{i \in \mathcal{I}} \text{ac}_{P,i}\right)$$

4. Fix $p : P \rightarrow G \in \text{Mor}_{C_1}$.

$$p : P \rightarrow G \models \bigvee_{i \in \mathcal{I}} \text{ac}_{P,i}$$

$$\xLeftrightarrow{\text{Def. 16}} \bigvee_{i \in \mathcal{I}} (p \models \text{ac}_{P,i})$$

$$\xLeftrightarrow{\text{Ind. H.}} \bigvee_{i \in \mathcal{I}} (\mathcal{F}(p) \models \mathcal{F}(\text{ac}_{P,i}))$$

$$\begin{aligned} \stackrel{\text{Def. 16}}{\Leftrightarrow} \mathcal{F}(\mathbf{p}) : \mathcal{F}(\mathbf{P}) \rightarrow \mathcal{F}(\mathbf{G}) &\models \bigvee_{i \in \mathcal{I}} \mathcal{F}(\text{ac}_{\mathbf{P},i}) \\ \stackrel{\text{Def. 47}}{\Leftrightarrow} \mathcal{F}(\mathbf{p}) : \mathcal{F}(\mathbf{P}) \rightarrow \mathcal{F}(\mathbf{G}) &\models \mathcal{F}\left(\bigvee_{i \in \mathcal{I}} \text{ac}_{\mathbf{P},i}\right) \end{aligned}$$

□

Lemma 12: (R-Simulations are Closed under Isomorphism Closure, see page 94)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, P_2)$, a rule relation $R \subseteq P_1 \times P_2$, and an R -simulation $S_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$. Then the isomorphism closure $\text{IC}(S_R) \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ of S_R is an R -simulation.

Proof.

Let $R \subseteq P_1 \times P_2$ be a rule relation. Let furthermore $S_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ be an R -simulation, i.e., according to Definition 49 we have

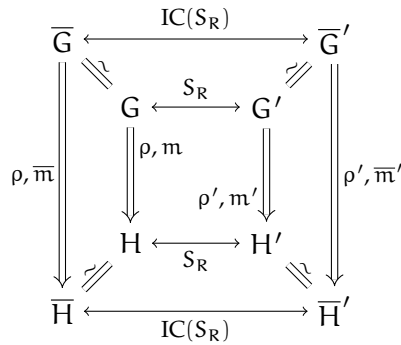
$$\begin{aligned} &\forall (G, G') \in S_R, H \in \text{Ob}_{\mathbf{C}_1}, \rho \in P_1, m : L \rightarrow G. ((G \xrightarrow{\rho, m} H) \\ &\Rightarrow (\exists H' \in \text{Ob}_{\mathbf{C}_2}, \rho' \in P_2, m' : L' \rightarrow G'. (G' \xrightarrow{\rho', m'} H' \wedge (\rho, \rho') \in R \wedge (H, H') \in S_R))) \end{aligned}$$

We have to show that $\text{IC}(S_R) = \{(\overline{G}, \overline{G}') \mid \exists (G, G') \in S_R. G \cong \overline{G} \wedge G' \cong \overline{G}'\}$ is an R -simulation, i.e., according to Definition 49 it holds that

$$\begin{aligned} &\forall (\overline{G}, \overline{G}') \in \text{IC}(S_R), \overline{H} \in \text{Ob}_{\mathbf{C}_1}, \rho \in P_1, \overline{m} : L \rightarrow \overline{G}. ((\overline{G} \xrightarrow{\rho, \overline{m}} \overline{H}) \\ &\Rightarrow (\exists \overline{H}' \in \text{Ob}_{\mathbf{C}_2}, \rho' \in P_2, \overline{m}' : L' \rightarrow \overline{G}'. (\overline{G}' \xrightarrow{\rho', \overline{m}'} \overline{H}' \wedge (\rho, \rho') \in R \wedge (\overline{H}, \overline{H}') \in \text{IC}(S_R)))) \end{aligned}$$

Assume the premise of the statement and fix $\overline{G}, \overline{H} \in \text{Ob}_{\mathbf{C}_1}$, $\overline{G}' \in \text{Ob}_{\mathbf{C}_2}$, $\rho \in P_1$ and $\overline{m} \in \text{Mor}_{\mathbf{C}_1}$. It remains to show the conclusion of the statement.

Because of the definition of $\text{IC}(S_R)$, we know that there are objects $G \in \text{Ob}_{\mathbf{C}_1}$ and $G' \in \text{Ob}_{\mathbf{C}_2}$ which are isomorphic to objects $\overline{G}, \overline{G}'$, respectively. Since transformation steps are unique up to isomorphism, we have that there is an object $H \in \text{Ob}_{\mathbf{C}_1}$ and a match morphism $m : L \rightarrow G$ in $\text{Mor}_{\mathbf{C}_1}$ such that the transformation step $G \xrightarrow{\rho, m} H$ is possible and $\overline{H} \cong H$. Furthermore, we have by definition of $\text{IC}(S_R)$ that $(G, G') \in S_R$, which means according to Definition 49 as given before that there is an object $H' \in \text{Ob}_{\mathbf{C}_2}$ and a match morphism $m' : L' \rightarrow G'$ in $\text{Mor}_{\mathbf{C}_2}$ such that the transformation step $G' \xrightarrow{\rho', m'} H'$ is possible, $(\rho, \rho') \in R$ and $(H, H') \in S_R$. Using again the fact that transformation steps are unique up to isomorphism, we get that there is an object $\overline{H}' \in \text{Ob}_{\mathbf{C}_2}$ and a match morphism $\overline{m}' : L' \rightarrow \overline{G}'$ in $\text{Mor}_{\mathbf{C}_2}$ such that the transformation step $\overline{G}' \xrightarrow{\rho', \overline{m}'} \overline{H}'$ is possible and $\overline{H}' \cong H'$. The definition of $\text{IC}(S_R)$ now obviously implies that $(\overline{H}, \overline{H}') \in \text{IC}(S_R)$. Thus, \overline{S}_R as defined above is an R -simulation.



□

Lemma 13: (R-Bisimulations are Closed under Isomorphism Closure, see page 95)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, P_2)$, a rule relation $R \subseteq P_1 \times P_2$, and an R -bisimulation $B_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$. Then the isomorphism closure $\text{IC}(B_R) \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_2}$ of B_R is an R -bisimulation.

Proof.

Assume that B_R is an R -bisimulation, therefore B_R and B_R^{-1} are R - resp. R^{-1} -simulations according to Definition 50. We have to show that $\text{IC}(B_R) = \{(\overline{G}, \overline{G}') \mid \exists (G, G') \in B_R. G \cong \overline{G} \wedge G' \cong \overline{G}'\}$ and $\text{IC}(B_R)^{-1}$ are R - resp. R^{-1} -simulations. Using the fact that R -similarity is closed under isomorphisms according to Lemma 12, we get that $\text{IC}(B_R)$ and $\text{IC}(B_R^{-1})$ are R - resp. R^{-1} -simulations. Furthermore, it holds the following:

$$\begin{aligned} \text{IC}(B_R^{-1}) &\stackrel{\text{Def. 48}}{=} \{(\overline{G}, \overline{G}') \mid \exists (G, G') \in B_R^{-1}. G \cong \overline{G} \wedge G' \cong \overline{G}'\} \\ &\stackrel{\text{Def.}^{-1}}{=} \{(\overline{G}, \overline{G}') \mid \exists (G', G) \in B_R. G \cong \overline{G} \wedge G' \cong \overline{G}'\} \\ &\stackrel{\text{renam.}}{=} \{(\overline{G}', \overline{G}) \mid \exists (G, G') \in B_R. G \cong \overline{G} \wedge G' \cong \overline{G}'\} \\ &\stackrel{\text{Def.}^{-1}}{=} \{(\overline{G}, \overline{G}') \mid \exists (G, G') \in B_R. G \cong \overline{G} \wedge G' \cong \overline{G}'\}^{-1} \\ &\stackrel{\text{Def. 48}}{=} \text{IC}(B_R)^{-1} \end{aligned}$$

Therefore, $\text{IC}(B_R)$ is an R -bisimulation. □

Lemma 14: (\mathcal{F} -Image Restricted \mathcal{F} -Transfer of R-Bisimilarity, see page 97)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $AS_2 = (\mathbf{C}_1, \mathcal{M}_1, P_2)$, $AS_3 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P_1))$, $AS_4 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P_2))$, rule relation $R \subseteq P_1 \times P_2$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (direct) transformations according to Theorem 3 and (\mathcal{M} -)morphisms. Then \mathcal{F} translates and creates \mathcal{F} -image restricted R -bisimilarity of objects, i.e., $B_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_1}$ is an R -bisimulation iff $\mathcal{F}(B_R) \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ is an $\mathcal{F}(R)$ -bisimulation.

Proof.

- (\Rightarrow) :

Let $R \subseteq P_1 \times P_2$ be a rule relation. Let furthermore $B_R \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_1}$ be an R -bisimulation, i.e., according to Definition 50 B_R, B_R^{-1} are R - resp. R^{-1} -simulations. This means according to Definition 49 for B_R that

$$\forall (G, G') \in B_R, H \in \text{Ob}_{\mathbf{C}_1}, \rho \in P_1, m : L \rightarrow G. ((G \xrightarrow{\rho, m} H)$$

$$\Rightarrow (\exists H' \in \text{Ob}_{\mathbf{C}_1}, \rho' \in P_2, m' : L' \rightarrow G'. (G' \xrightarrow{\rho', m'} H' \wedge (\rho, \rho') \in R \wedge (H, H') \in B_R)))$$

and for B_R^{-1} that

$$\forall (G', G) \in B_R^{-1}, H' \in \text{Ob}_{\mathbf{C}_1}, \rho' \in P_2, m' : L' \rightarrow G'. ((G' \xrightarrow{\rho', m'} H')$$

$$\Rightarrow (\exists H \in \text{Ob}_{\mathbf{C}_1}, \rho \in P_1, m : L \rightarrow G. (G \xrightarrow{\rho, m} H \wedge (\rho', \rho) \in R^{-1} \wedge (H', H) \in B_R^{-1}))).$$

We have to show that $\mathcal{F}(B_R) \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ is an $\mathcal{F}(R)$ -bisimulation, i.e., according to Definition 50 $\mathcal{F}(B_R), \mathcal{F}(B_R^{-1})$ are $\mathcal{F}(R)$ - resp. $\mathcal{F}(R)^{-1}$ -simulations.

– **Part 1:**

Let $\mathcal{F}(\mathbf{R}) \subseteq \mathcal{F}(\mathbf{P}_1) \times \mathcal{F}(\mathbf{P}_2)$ be a rule relation. $\mathcal{F}(\mathbf{B}_\mathbf{R})$ is an $\mathcal{F}(\mathbf{R})$ -simulation means according to Definition 49 that

$$\begin{aligned} & \forall (\mathcal{F}(\mathbf{G}), \mathcal{F}(\mathbf{G}')) \in \mathcal{F}(\mathbf{B}_\mathbf{R}), X \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho) \in \mathcal{F}(\mathbf{P}_1), \overline{\mathbf{m}} : \mathcal{F}(\mathbf{L}) \rightarrow \mathcal{F}(\mathbf{G}). \\ & ((\mathcal{F}(\mathbf{G}) \xrightarrow{\mathcal{F}(\rho), \overline{\mathbf{m}}} X) \\ & \Rightarrow (\exists X' \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho') \in \mathcal{F}(\mathbf{P}_2), \overline{\mathbf{m}}' : \mathcal{F}(\mathbf{L}') \rightarrow \mathcal{F}(\mathbf{G}'). \\ & (\mathcal{F}(\mathbf{G}') \xrightarrow{\mathcal{F}(\rho'), \overline{\mathbf{m}}'} X' \wedge (\mathcal{F}(\rho), \mathcal{F}(\rho')) \in \mathcal{F}(\mathbf{R}) \wedge (X, X') \in \mathcal{F}(\mathbf{B}_\mathbf{R})))) \end{aligned}$$

Assume the premise of the statement and fix $\mathbf{G}, \mathbf{G}' \in \text{Ob}_{\mathbf{C}_1}$, $X \in \text{Ob}_{\mathbf{C}_2}$, $\rho \in \mathbf{P}_1$ and $\overline{\mathbf{m}} \in \text{Mor}_{\mathbf{C}_2}$. It remains to show the conclusion of the statement.

Since \mathcal{F} creates morphisms by assumption, we have the unique match morphism $\mathbf{m} : \mathbf{L} \rightarrow \mathbf{G}$ such that $\mathcal{F}(\mathbf{m}) = \overline{\mathbf{m}}$. Furthermore, since \mathcal{F} creates (direct) transformations by assumption, we get the transformation step $\mathbf{G} \xrightarrow{\rho, \mathbf{m}} \mathbf{H}$ in AS_1 with $\mathcal{F}(\mathbf{H}) \cong X$ leading to the transformation step $\mathcal{F}(\mathbf{G}) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(\mathbf{m})} \mathcal{F}(\mathbf{H})$ in AS_3 , because transformation steps are unique up to isomorphism. Since $\mathbf{B}_\mathbf{R} \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_1}$ is an \mathbf{R} -bisimulation by assumption, we additionally know that there is also the transformation step $\mathbf{G}' \xrightarrow{\rho', \mathbf{m}'} \mathbf{H}'$ in AS_2 such that $(\rho, \rho') \in \mathbf{R}$ and $(\mathbf{H}, \mathbf{H}') \in \mathbf{B}_\mathbf{R}$. Since \mathcal{F} preserves commuting diagrams by general functor property and pushouts along \mathcal{M} -morphisms by \mathcal{M} -functor property, we have the transformation step $\mathcal{F}(\mathbf{G}') \xrightarrow{\mathcal{F}(\rho'), \mathcal{F}(\mathbf{m}')} \mathcal{F}(\mathbf{H}')$ in AS_4 . Thus, there are an object $X' = \mathcal{F}(\mathbf{H}')$ with the transformation step $\mathcal{F}(\mathbf{G}') \xrightarrow{\mathcal{F}(\rho'), \mathcal{F}(\mathbf{m}')} \mathcal{F}(\mathbf{H}')$ and the match morphism $\overline{\mathbf{m}}' = \mathcal{F}(\mathbf{m}')$. This implies that $\mathcal{F}(\mathbf{B}_\mathbf{R})$ is an $\mathcal{F}(\mathbf{R})$ -simulation, because we additionally have that $(\mathcal{F}(\rho), \mathcal{F}(\rho')) \in \mathcal{F}(\mathbf{R})$ since $(\rho, \rho') \in \mathbf{R}$ and $(\mathcal{F}(\mathbf{H}), \mathcal{F}(\mathbf{H}')) \in \mathcal{F}(\mathbf{B}_\mathbf{R})$ since $(\mathbf{H}, \mathbf{H}') \in \mathbf{B}_\mathbf{R}$.

$$\begin{array}{ccc} \mathcal{F}(\mathbf{G}) & \xleftarrow{\mathcal{F}(\mathbf{B}_\mathbf{R})} & \mathcal{F}(\mathbf{G}') \\ \mathcal{F}(\rho), \overline{\mathbf{m}} = \mathcal{F}(\mathbf{m}) \downarrow & & \downarrow \mathcal{F}(\rho'), \mathcal{F}(\mathbf{m}') = \overline{\mathbf{m}}' \\ \mathbf{X} \cong \mathcal{F}(\mathbf{H}) & \xleftarrow{\mathcal{F}(\mathbf{B}_\mathbf{R})} & \mathcal{F}(\mathbf{H}') = \mathbf{X}' \end{array} \quad \begin{array}{ccc} \mathbf{G} & \xleftarrow{\mathbf{B}_\mathbf{R}} & \mathbf{G}' \\ \rho, \mathbf{m} \downarrow & & \downarrow \rho', \mathbf{m}' \\ \mathbf{H} & \xleftarrow{\mathbf{B}_\mathbf{R}} & \mathbf{H}' \end{array}$$

– **Part 2:**

$\mathcal{F}(\mathbf{B}_\mathbf{R}^{-1})$ is an $\mathcal{F}(\mathbf{R})^{-1}$ -simulation means according to Definition 49 that

$$\begin{aligned} & \forall (\mathcal{F}(\mathbf{G}'), \mathcal{F}(\mathbf{G})) \in \mathcal{F}(\mathbf{B}_\mathbf{R}^{-1}), X' \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho') \in \mathcal{F}(\mathbf{P}_2), \overline{\mathbf{m}}' : \mathcal{F}(\mathbf{L}') \rightarrow \mathcal{F}(\mathbf{G}'). \\ & ((\mathcal{F}(\mathbf{G}') \xrightarrow{\mathcal{F}(\rho'), \overline{\mathbf{m}}'} X') \\ & \Rightarrow (\exists X \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho) \in \mathcal{F}(\mathbf{P}_1), \overline{\mathbf{m}} : \mathcal{F}(\mathbf{L}) \rightarrow \mathcal{F}(\mathbf{G}). \\ & (\mathcal{F}(\mathbf{G}) \xrightarrow{\mathcal{F}(\rho), \overline{\mathbf{m}}} X \wedge (\mathcal{F}(\rho'), \mathcal{F}(\rho)) \in \mathcal{F}(\mathbf{R})^{-1} \wedge (X', X) \in \mathcal{F}(\mathbf{B}_\mathbf{R}^{-1})))) \end{aligned}$$

The proof works analogously to the Part 1 above.

• (\Leftarrow):

Let $\mathcal{F}(\mathbf{B}_\mathbf{R}) \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ be an $\mathcal{F}(\mathbf{R})$ -bisimulation, i.e., according to Definition 50 $\mathcal{F}(\mathbf{B}_\mathbf{R})$, $\mathcal{F}(\mathbf{B}_\mathbf{R}^{-1})$ are $\mathcal{F}(\mathbf{R})$ - resp. $\mathcal{F}(\mathbf{R})^{-1}$ -simulations. This means according to Definition 49 for $\mathcal{F}(\mathbf{B}_\mathbf{R})$ that

$$\forall (\mathcal{F}(\mathbf{G}), \mathcal{F}(\mathbf{G}')) \in \mathcal{F}(\mathbf{B}_\mathbf{R}), X \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho) \in \mathcal{F}(\mathbf{P}_1), \overline{\mathbf{m}} : \mathcal{F}(\mathbf{L}) \rightarrow \mathcal{F}(\mathbf{G}). ((\mathcal{F}(\mathbf{G}) \xrightarrow{\mathcal{F}(\rho), \overline{\mathbf{m}}} X$$

$$\Rightarrow (\exists X' \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho') \in \mathcal{F}(\mathbf{P}_2), \overline{m}' : \mathcal{F}(\mathbf{L}') \rightarrow \mathcal{F}(\mathbf{G}')).$$

$$(\mathcal{F}(\mathbf{G}') \xrightarrow{\mathcal{F}(\rho'), \overline{m}'} X' \wedge (\mathcal{F}(\rho), \mathcal{F}(\rho')) \in \mathcal{F}(\mathbf{R}) \wedge (X, X') \in \mathcal{F}(\mathbf{B}_\mathbf{R})))$$

and for $\mathcal{F}(\mathbf{B}_\mathbf{R}^{-1})$ that

$$\forall (\mathcal{F}(\mathbf{G}'), \mathcal{F}(\mathbf{G})) \in \mathcal{F}(\mathbf{B}_\mathbf{R}^{-1}), X' \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho') \in \mathcal{F}(\mathbf{P}_2), \overline{m}' : \mathcal{F}(\mathbf{L}') \rightarrow \mathcal{F}(\mathbf{G}').$$

$$((\mathcal{F}(\mathbf{G}') \xrightarrow{\mathcal{F}(\rho'), \overline{m}'} X'))$$

$$\Rightarrow (\exists X \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho) \in \mathcal{F}(\mathbf{P}_1), \overline{m} : \mathcal{F}(\mathbf{L}) \rightarrow \mathcal{F}(\mathbf{G})).$$

$$(\mathcal{F}(\mathbf{G}) \xrightarrow{\mathcal{F}(\rho), \overline{m}} X \wedge (\mathcal{F}(\rho'), \mathcal{F}(\rho)) \in \mathcal{F}(\mathbf{R})^{-1} \wedge (X', X) \in \mathcal{F}(\mathbf{B}_\mathbf{R}^{-1})))$$

We have to show that $\mathbf{B}_\mathbf{R} \subseteq \text{Ob}_{\mathbf{C}_1} \times \text{Ob}_{\mathbf{C}_1}$ is an \mathbf{R} -bisimulation, i.e., according to Definition 50 $\mathbf{B}_\mathbf{R}, \mathbf{B}_\mathbf{R}^{-1}$ are \mathbf{R} - resp. \mathbf{R}^{-1} -simulations.

– **Part 1:**

$\mathbf{B}_\mathbf{R}$ is an \mathbf{R} -simulation means according to Definition 49 that

$$\forall (G, G') \in \mathbf{B}_\mathbf{R}, H \in \text{Ob}_{\mathbf{C}_1}, \rho \in \mathbf{P}_1, m : \mathbf{L} \rightarrow G. ((G \xrightarrow{\rho, m} H))$$

$$\Rightarrow (\exists H' \in \text{Ob}_{\mathbf{C}_1}, \rho' \in \mathbf{P}_2, m' : \mathbf{L}' \rightarrow G').$$

$$(G' \xrightarrow{\rho', m'} H' \wedge (\rho, \rho') \in \mathbf{R} \wedge (H, H') \in \mathbf{B}_\mathbf{R}))$$

Assume the premise of the statement and fix $G, G', H \in \text{Ob}_{\mathbf{C}_1}$, $\rho \in \mathbf{P}_1$ and $m \in \text{Mor}_{\mathbf{C}_1}$. It remains to show the conclusion of the statement.

Since \mathcal{F} preserves commuting diagrams by general functor property and pushouts along \mathcal{M} -morphisms by \mathcal{M} -functor property, we have the transformation step $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(m)} \mathcal{F}(H)$ in AS_3 with the match morphism $\mathcal{F}(m) : \mathcal{F}(\mathbf{L}) \rightarrow \mathcal{F}(\mathbf{G})$. Since $\mathcal{F}(\mathbf{B}_\mathbf{R})$ is an $\mathcal{F}(\mathbf{R})$ -bisimulation by assumption, we now have the transformation step $\mathcal{F}(G') \xrightarrow{\mathcal{F}(\rho'), \overline{m}'} X'$ in AS_4 such that $(\mathcal{F}(\rho), \mathcal{F}(\rho')) \in \mathcal{F}(\mathbf{R})$ and $(\mathcal{F}(H), X') \in \mathcal{F}(\mathbf{B}_\mathbf{R})$. Since \mathcal{F} creates morphisms by assumption, we have the unique match morphism $m' : \mathbf{L}' \rightarrow G'$ such that $\mathcal{F}(m') = \overline{m}'$. Furthermore, since \mathcal{F} creates (direct) transformations by assumption, we get the transformation step $G' \xrightarrow{\rho', m'} H'$ in AS_2 with $\mathcal{F}(H') \cong X'$ leading to the transformation step $\mathcal{F}(G') \xrightarrow{\mathcal{F}(\rho'), \mathcal{F}(m')} \mathcal{F}(H')$ in AS_4 , because transformation steps are unique up to isomorphism. This implies that $\mathbf{B}_\mathbf{R}$ is an \mathbf{R} -simulation, because we additionally have that $(\rho, \rho') \in \mathbf{R}$ since it holds that $(\mathcal{F}(\rho), \mathcal{F}(\rho')) \in \mathcal{F}(\mathbf{R})$ as well as $(H, H') \in \mathbf{B}_\mathbf{R}$, because it holds that $(\mathcal{F}(H), \mathcal{F}(H')) \in \mathcal{F}(\mathbf{B}_\mathbf{R})$ and \mathcal{F} creates morphisms (thus also identities implying injectivity of \mathcal{F} on objects according to Lemmas 8 and 9) by assumption.

$$\begin{array}{ccc} G & \xleftarrow{\mathbf{B}_\mathbf{R}} & G' \\ \rho, m \downarrow & & \downarrow \rho', m' \\ H & \xleftarrow{\mathbf{B}_\mathbf{R}} & H' \end{array} \quad \begin{array}{ccc} \mathcal{F}(G) & \xleftarrow{\mathcal{F}(\mathbf{B}_\mathbf{R})} & \mathcal{F}(G') \\ \mathcal{F}(\rho), \mathcal{F}(m) = \overline{m} \downarrow & & \downarrow \mathcal{F}(\rho'), \overline{m}' = \mathcal{F}(m') \\ \mathcal{F}(H) = X & \xleftarrow{\mathcal{F}(\mathbf{B}_\mathbf{R})} & X' \cong \mathcal{F}(H') \end{array}$$

– **Part 2:**

$\mathbf{B}_\mathbf{R}^{-1}$ is an \mathbf{R}^{-1} -simulation means according to Definition 49 that

$$\forall (G', G) \in \mathbf{B}_\mathbf{R}^{-1}, H' \in \text{Ob}_{\mathbf{C}_1}, \rho' \in \mathbf{P}_2, m' : \mathbf{L}' \rightarrow G'. ((G' \xrightarrow{\rho', m'} H'))$$

$$\Rightarrow (\exists H \in \text{Ob}_{\mathbf{C}_1}, \rho \in P_1, \mathfrak{m} : L \rightarrow G. (G \xrightarrow{\rho, \mathfrak{m}} H \wedge (\rho', \rho) \in R^{-1} \wedge (H', H) \in B_R^{-1}))).$$

The proof works analogously to the Part 1 above.

□

Lemma 15: (R-Bisimulation is Preserved under \mathcal{F} -Image Restriction, see page 97)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P_1)$, $AS_2 = (\mathbf{C}_1, \mathcal{M}_1, P_2)$, $AS_3 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P_1))$, $AS_4 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P_2))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (direct) transformations according to Theorem 3 and (\mathcal{M} -)morphisms. Consider furthermore a rule relation $R \subseteq P_1 \times P_2$ as well as an $\mathcal{F}(R)$ -bisimulation $B'_{\mathcal{F}(R)} \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ that is closed under isomorphisms, i.e., $\text{IC}(B'_{\mathcal{F}(R)}) = B'_{\mathcal{F}(R)}$. Then $\bar{B}_{\mathcal{F}(R)} \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ with $\bar{B}_{\mathcal{F}(R)} = B'_{\mathcal{F}(R)} \cap (\mathcal{F}(\text{Ob}_{\mathbf{C}_1}) \times \mathcal{F}(\text{Ob}_{\mathbf{C}_1}))$ is also an $\mathcal{F}(R)$ -bisimulation.

Proof.

Let $B'_{\mathcal{F}(R)} \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ be an $\mathcal{F}(R)$ -bisimulation, i.e., according to Definition 50 $B'_{\mathcal{F}(R)}$, $B'^{-1}_{\mathcal{F}(R)}$ are $\mathcal{F}(R)$ - resp. $\mathcal{F}(R)^{-1}$ -simulations. This means according to Definition 49 for $B'_{\mathcal{F}(R)}$ that

$$\begin{aligned} & \forall (\bar{G}, \bar{G}') \in B'_{\mathcal{F}(R)}, X \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho) \in \mathcal{F}(P_1), \underline{\mathfrak{m}} : \mathcal{F}(L) \rightarrow \bar{G}. ((\bar{G} \xrightarrow{\mathcal{F}(\rho), \underline{\mathfrak{m}}} X) \\ & \Rightarrow (\exists X' \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho') \in \mathcal{F}(P_2), \underline{\mathfrak{m}}' : \mathcal{F}(L') \rightarrow \bar{G}'. \\ & (\bar{G}' \xrightarrow{\mathcal{F}(\rho'), \underline{\mathfrak{m}}'} X' \wedge (\mathcal{F}(\rho), \mathcal{F}(\rho')) \in \mathcal{F}(R) \wedge (X, X') \in B'_{\mathcal{F}(R)}))) \end{aligned}$$

and for $B'^{-1}_{\mathcal{F}(R)}$ that

$$\begin{aligned} & \forall (\bar{G}', \bar{G}) \in B'^{-1}_{\mathcal{F}(R)}, X' \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho') \in \mathcal{F}(P_2), \underline{\mathfrak{m}}' : \mathcal{F}(L') \rightarrow \bar{G}'. ((\bar{G}' \xrightarrow{\mathcal{F}(\rho'), \underline{\mathfrak{m}}'} X') \\ & \Rightarrow (\exists X \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho) \in \mathcal{F}(P_1), \underline{\mathfrak{m}} : \mathcal{F}(L) \rightarrow \bar{G}. \\ & (\bar{G} \xrightarrow{\mathcal{F}(\rho), \underline{\mathfrak{m}}} X \wedge (\mathcal{F}(\rho'), \mathcal{F}(\rho)) \in \mathcal{F}(R)^{-1} \wedge (X', X) \in B'^{-1}_{\mathcal{F}(R)}))) \end{aligned}$$

We have to show that $\bar{B}_{\mathcal{F}(R)} \subseteq \text{Ob}_{\mathbf{C}_2} \times \text{Ob}_{\mathbf{C}_2}$ with $\bar{B}_{\mathcal{F}(R)} = B'_{\mathcal{F}(R)} \cap (\mathcal{F}(\text{Ob}_{\mathbf{C}_1}) \times \mathcal{F}(\text{Ob}_{\mathbf{C}_1}))$ is an $\mathcal{F}(R)$ -bisimulation, i.e., according to Definition 50 $\bar{B}_{\mathcal{F}(R)}$, $\bar{B}_{\mathcal{F}(R)}^{-1}$ are $\mathcal{F}(R)$ - resp. $\mathcal{F}(R)^{-1}$ -simulations.

• **Part 1:**

$\bar{B}_{\mathcal{F}(R)}$ is an $\mathcal{F}(R)$ -simulation means according to Definition 49 that

$$\begin{aligned} & \forall (\mathcal{F}(G), \mathcal{F}(G')) \in \bar{B}_{\mathcal{F}(R)}, Y \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho) \in \mathcal{F}(P_1), \bar{\mathfrak{m}} : \mathcal{F}(L) \rightarrow \mathcal{F}(G). ((\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \bar{\mathfrak{m}}} Y) \\ & \Rightarrow (\exists Y' \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho') \in \mathcal{F}(P_2), \bar{\mathfrak{m}}' : \mathcal{F}(L') \rightarrow \mathcal{F}(G'). \\ & (\mathcal{F}(G') \xrightarrow{\mathcal{F}(\rho'), \bar{\mathfrak{m}}'} Y' \wedge (\mathcal{F}(\rho), \mathcal{F}(\rho')) \in \mathcal{F}(R) \wedge (Y, Y') \in \bar{B}_{\mathcal{F}(R)}))) \end{aligned}$$

Assume the premise of the statement and fix $G, G' \in \text{Ob}_{\mathbf{C}_1}$, $Y \in \text{Ob}_{\mathbf{C}_2}$, $\rho \in P_1$ and $\bar{\mathfrak{m}} \in \text{Mor}_{\mathbf{C}_2}$. It remains to show the conclusion of the statement.

Since \mathcal{F} creates morphisms by assumption, we have the unique match morphism $\mathfrak{m} : L \rightarrow G$ such that $\mathcal{F}(\mathfrak{m}) = \bar{\mathfrak{m}}$. Furthermore, since \mathcal{F} creates (direct) transformations by assumption, we get the transformation step $G \xrightarrow{\rho, \mathfrak{m}} H$ in AS_1 with $\mathcal{F}(H) \cong Y$ leading to the transformation step $\mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(\mathfrak{m})} \mathcal{F}(H)$ in AS_3 , because transformation steps are unique up to isomorphism. Because of the definition of $\bar{B}_{\mathcal{F}(R)}$, we additionally know that $(\mathcal{F}(G), \mathcal{F}(G')) \in$

$B'_{\mathcal{F}(\mathbf{R})}$, which implies according to the assumption above that for the transformation step $\mathcal{F}(\mathbf{G}) \xrightarrow{\mathcal{F}(\rho), \mathcal{F}(\mathbf{m})} \mathcal{F}(\mathbf{H})$ there are object $X' \in \text{Ob}_{\mathbf{C}_2}$, a match morphism $\underline{\mathbf{m}}' : \mathcal{F}(\mathbf{L}') \rightarrow \mathcal{F}(\mathbf{G}')$ and a rule $\rho' \in \mathbf{P}_2$ such that $(\rho, \rho') \in \mathbf{R}$ and $\mathcal{F}(\mathbf{G}') \xrightarrow{\mathcal{F}(\rho'), \underline{\mathbf{m}}'} X'$ and $(\mathcal{F}(\mathbf{H}), X') \in B'_{\mathcal{F}(\mathbf{R})}$. Since \mathcal{F} creates morphisms by assumption, we get the unique match morphism $\mathbf{m}' : \mathbf{L}' \rightarrow \mathbf{G}'$ such that $\mathcal{F}(\mathbf{m}') = \underline{\mathbf{m}}'$. Furthermore, since \mathcal{F} creates (direct) transformations by assumption, we get the transformation step $\mathbf{G}' \xrightarrow{\rho', \mathbf{m}'} \mathbf{H}'$ in AS_2 with $\mathcal{F}(\mathbf{H}') \cong X'$ leading to the transformation step $\mathcal{F}(\mathbf{G}') \xrightarrow{\mathcal{F}(\rho'), \mathcal{F}(\mathbf{m}')} \mathcal{F}(\mathbf{H}')$ in AS_4 , because transformation steps are unique up to isomorphism. Thus, there are an object $\mathbf{Y}' = \mathcal{F}(\mathbf{H}')$, a rule $\mathcal{F}(\rho')$, and a match morphism $\overline{\mathbf{m}}' = \mathcal{F}(\mathbf{m}')$ with $(\mathcal{F}(\rho), \mathcal{F}(\rho')) \in \mathcal{F}(\mathbf{R})$ and the transformation step $\mathcal{F}(\mathbf{G}') \xrightarrow{\mathcal{F}(\rho'), \mathcal{F}(\mathbf{m}')} \mathcal{F}(\mathbf{H}')$. Furthermore, we have that $(\mathcal{F}(\mathbf{H}), \mathcal{F}(\mathbf{H}')) \in \overline{B}_{\mathcal{F}(\mathbf{R})}$, because $(\mathcal{F}(\mathbf{H}), X' \cong \mathcal{F}(\mathbf{H}')) \in B'_{\mathcal{F}(\mathbf{R})}$ and obviously $(\mathcal{F}(\mathbf{H}), \mathcal{F}(\mathbf{H}')) \in (\mathcal{F}(\text{Ob}_{\mathbf{C}_1}) \times \mathcal{F}(\text{Ob}_{\mathbf{C}_1}))$ as well.

$$\begin{array}{ccccc}
\mathcal{F}(\mathbf{G}) & \xleftarrow{\overline{B}_{\mathcal{F}(\mathbf{R})} \subseteq B'_{\mathcal{F}(\mathbf{R})}} & \mathcal{F}(\mathbf{G}') & & \mathbf{G} & & \mathbf{G}' \\
\mathcal{F}(\rho), \overline{\mathbf{m}} = \mathcal{F}(\mathbf{m}) \Downarrow & & \Downarrow \mathcal{F}(\rho'), \underline{\mathbf{m}}' = \mathcal{F}(\mathbf{m}') = \overline{\mathbf{m}}' & \rho, \mathbf{m} \Downarrow & & & \Downarrow \rho', \mathbf{m}' \\
\mathbf{Y} \cong \mathcal{F}(\mathbf{H}) & \xleftarrow{\overline{B}_{\mathcal{F}(\mathbf{R})} \subseteq B'_{\mathcal{F}(\mathbf{R})}} & \mathbf{X}' \cong \mathcal{F}(\mathbf{H}') = \mathbf{Y}' & & \mathbf{H} & & \mathbf{H}'
\end{array}$$

• **Part 2:**

$\overline{B}_{\mathcal{F}(\mathbf{R})}^{-1}$ is an $\mathcal{F}(\mathbf{R})^{-1}$ -simulation means according to Definition 49 that

$$\forall (\mathcal{F}(\mathbf{G}'), \mathcal{F}(\mathbf{G})) \in \overline{B}_{\mathcal{F}(\mathbf{R})}^{-1}, \mathbf{Y}' \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho') \in \mathcal{F}(\mathbf{P}_2), \overline{\mathbf{m}}' : \mathcal{F}(\mathbf{L}') \rightarrow \mathcal{F}(\mathbf{G}').$$

$$((\mathcal{F}(\mathbf{G}') \xrightarrow{\mathcal{F}(\rho'), \overline{\mathbf{m}}'} \mathbf{Y}') \Rightarrow (\exists \mathbf{Y} \in \text{Ob}_{\mathbf{C}_2}, \mathcal{F}(\rho) \in \mathcal{F}(\mathbf{P}_1), \overline{\mathbf{m}} : \mathcal{F}(\mathbf{L}) \rightarrow \mathcal{F}(\mathbf{G})).$$

$$(\mathcal{F}(\mathbf{G}) \xrightarrow{\mathcal{F}(\rho), \overline{\mathbf{m}}} \mathbf{Y} \wedge (\mathcal{F}(\rho'), \mathcal{F}(\rho)) \in \mathcal{F}(\mathbf{R})^{-1} \wedge (\mathbf{Y}', \mathbf{Y}) \in \overline{B}_{\mathcal{F}(\mathbf{R})}^{-1}))$$

The proof works analogously to the Part 1 above. □

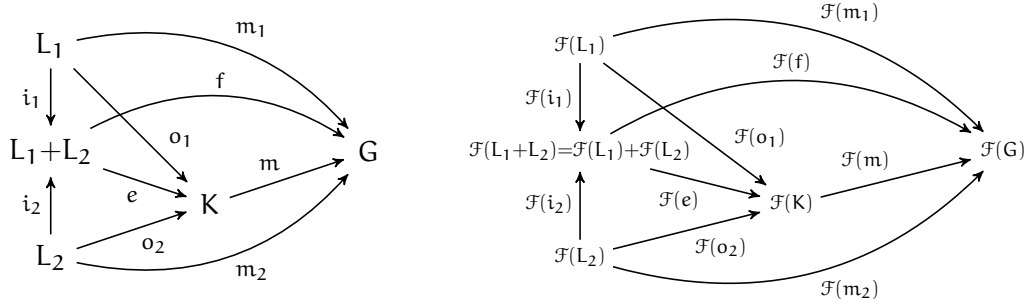
Lemma 16: (\mathcal{F} Preserves $\mathcal{E}' - \mathcal{M}$ Pair Factorization, see page 102)

Consider \mathcal{M} -adhesive categories $(\mathbf{C}_i, \mathcal{M}_i)$ with binary coproducts and $\mathcal{E}_i - \mathcal{M}_i$ -factorizations for $i \in \{1, 2\}$ as well as an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$. Then \mathcal{F} preserves $\mathcal{E}' - \mathcal{M}$ pair factorization based on an $\mathcal{E} - \mathcal{M}$ -factorization if \mathcal{F} preserves coproducts and epimorphisms.

Proof.

Consider arbitrary morphisms $\mathbf{m}_1 : \mathbf{L}_1 \rightarrow \mathbf{G}$ and $\mathbf{m}_2 : \mathbf{L}_2 \rightarrow \mathbf{G}$ with common codomain in $\text{Mor}_{\mathbf{C}_1}$. Construct first an $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization of $(\mathbf{m}_1, \mathbf{m}_2)$ in $(\mathbf{C}_1, \mathcal{M}_1)$ according to Lemma 1. As given in the diagram below to the left, we get an $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization $((\mathbf{o}_1, \mathbf{o}_2), \mathbf{m})$ with $\mathbf{o}_j = \mathbf{e} \circ \mathbf{i}_j$ for $j \in \{1, 2\}$. Applying \mathcal{F} to the diagram below to the left, we get the diagram given below to the right, which commutes as well since functors preserve commuting diagrams by general functor property. Since \mathcal{F} preserves coproducts by assumption, we have that $(\mathcal{F}(\mathbf{L}_1 + \mathbf{L}_2), \mathcal{F}(\mathbf{i}_1) : \mathcal{F}(\mathbf{L}_1) \rightarrow \mathcal{F}(\mathbf{L}_1 + \mathbf{L}_2), \mathcal{F}(\mathbf{i}_2) : \mathcal{F}(\mathbf{L}_2) \rightarrow \mathcal{F}(\mathbf{L}_1 + \mathbf{L}_2))$ is a binary coproduct of $(\mathcal{F}(\mathbf{L}_1), \mathcal{F}(\mathbf{L}_2))$ in $(\mathbf{C}_2, \mathcal{M}_2)$ with induced coproduct morphism $\mathcal{F}(\mathbf{f}) : \mathcal{F}(\mathbf{L}_1 + \mathbf{L}_2) \rightarrow \mathcal{F}(\mathbf{G})$. Moreover according to Definition 10, $\mathcal{F}(\mathbf{L}_1 + \mathbf{L}_2) \xrightarrow{\mathcal{F}(\mathbf{e})} \mathcal{F}(\mathbf{K}) \xrightarrow{\mathcal{F}(\mathbf{m})} \mathcal{F}(\mathbf{G})$ is an $\mathcal{E}_2 - \mathcal{M}_2$ -factorization of the morphism $\mathcal{F}(\mathbf{f})$ with $\mathcal{F}(\mathbf{e}) \in \mathcal{E}_2$ and $\mathcal{F}(\mathbf{m}) \in \mathcal{M}_2$, because

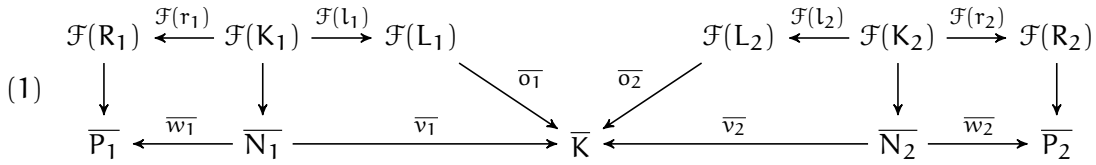
\mathcal{F} preserves monomorphisms and epimorphisms by assumption. Thus using Lemma 1, we get that $((\mathcal{F}(o_1) = \mathcal{F}(e) \circ \mathcal{F}(i_1), \mathcal{F}(o_2) = \mathcal{F}(e) \circ \mathcal{F}(i_2)), \mathcal{F}(m))$ is an $\mathcal{E}'_2 - \mathcal{M}_2$ pair factorization of $(\mathcal{F}(m_1), \mathcal{F}(m_2))$.



□

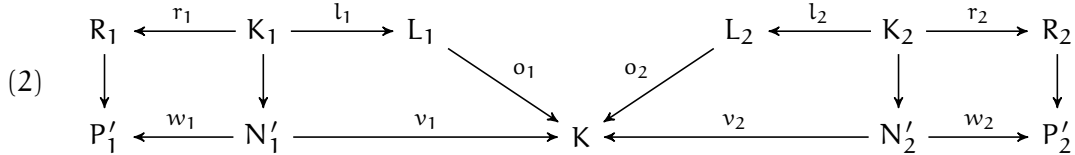
Lemma 17: (\mathcal{F} -Reachable Critical Pairs, see page 103)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathcal{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathcal{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, and an \mathcal{M} -functor $\mathcal{F} : (\mathcal{C}_1, \mathcal{M}_1) \rightarrow (\mathcal{C}_2, \mathcal{M}_2)$ that creates (direct) transformations according to Theorem 1 and morphisms. Then each critical pair of rules $\mathcal{F}(\rho_1)$ and $\mathcal{F}(\rho_2)$ with overlapping $\bar{K} \cong \mathcal{F}(K)$ in AS_2 is already \mathcal{F} -reachable (up to isomorphism).

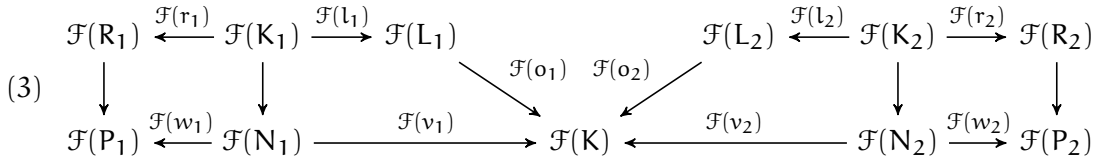


Proof.

Consider a critical pair (1) with $\bar{K} = \mathcal{F}(K)$. This implies by creation of (direct) transformations with $\mathcal{F}(o_i) = \bar{o}_i$ for $i \in \{1, 2\}$ the following diagram (2)



with $\mathcal{F}((2)) \cong (1)$. The uniqueness of (direct) transformations with given match morphisms implies that $\mathcal{F}((2)) \cong (3)$ and hence $(1) \cong (3)$. This means that (1) is an \mathcal{F} -reachable critical pair.



□

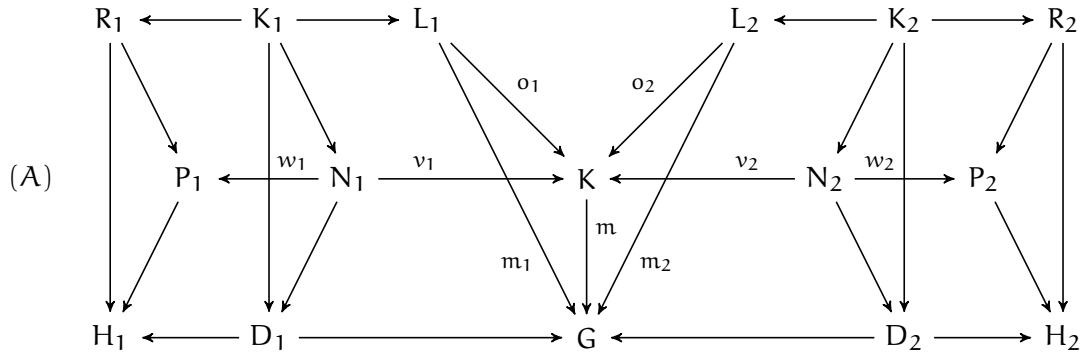
Lemma 18: (Completeness of \mathcal{F} -Reachable Critical Pair, see page 103)

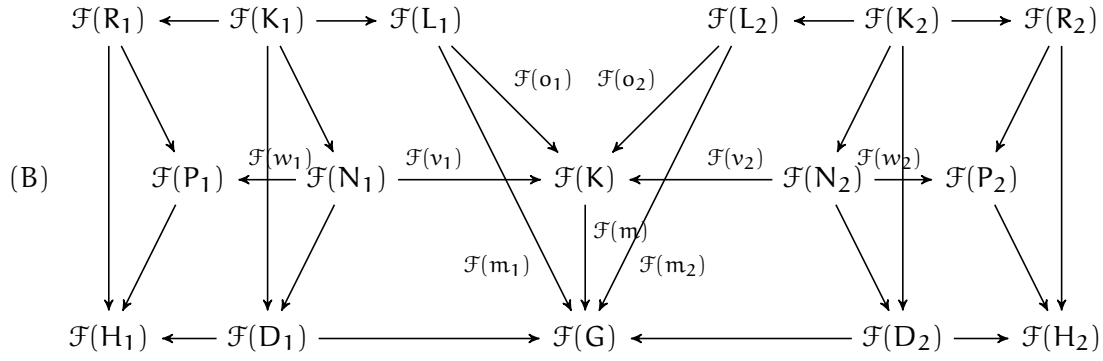
Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that is compatible with pair factorization, and a translated parallel dependent transformation span $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$. Then there is an \mathcal{F} -reachable critical pair $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ of rules $\mathcal{F}(\rho_1)$, $\mathcal{F}(\rho_2)$ and an embedding given below.

$$\begin{array}{ccccc}
 \mathcal{F}(P_1) & \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} & \mathcal{F}(K) & \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} & \mathcal{F}(P_2) \\
 \downarrow & & \downarrow \mathcal{F}(m) & & \downarrow \\
 \mathcal{F}(H_1) & \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} & \mathcal{F}(G) & \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} & \mathcal{F}(H_2)
 \end{array}
 \quad \begin{array}{c} (1) \quad (2) \end{array}$$

Proof.

The translated parallel dependent transformation span $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$ in AS_2 is generated by parallel dependent transformation span $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$ in AS_1 . Because of the existence of an $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization in AS_1 , we have completeness of critical pairs in AS_1 using Fact 2 and therefore a critical pair $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ in AS_1 with the corresponding embedding (see diagram (A) below), where $m \circ o_1 = m_1$ and $m \circ o_2 = m_2$ is an $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization with $(o_1, o_2) \in \mathcal{E}'_1$ and $m \in \mathcal{M}_1$. By assumption that \mathcal{F} is compatible with pair factorization, we have that $\mathcal{F}(m) \circ \mathcal{F}(o_1) = \mathcal{F}(m_1)$ and $\mathcal{F}(m) \circ \mathcal{F}(o_2) = \mathcal{F}(m_2)$ is an $\mathcal{E}'_2 - \mathcal{M}_2$ pair factorization with $(\mathcal{F}(o_1), \mathcal{F}(o_2)) \in \mathcal{E}'_2$ and $\mathcal{F}(m) \in \mathcal{M}_2$. This implies by construction of critical pairs in [88] and by application of Lemma 17 that $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ is an \mathcal{F} -reachable critical pair (see diagram (B) below). Applying \mathcal{F} to the diagram (A) we obtain also an embedding into the given translated parallel dependent transformation span $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$, because $\mathcal{F}(m) \in \mathcal{M}_2$. Now the lower part of the diagram (B) corresponds to the extension diagrams (1) and (2) since \mathcal{F} preserves pushouts along \mathcal{M} -morphisms using the proof of the Completeness of Critical Pairs Lemma in [88].





□

Lemma 19: (Creation of Compatibility of Nested Application Conditions, see page 110)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (\mathcal{M}) -morphisms. Then \mathcal{F} creates compatibility of nested application conditions, i.e.,

$$(\mathcal{F}(\text{ac}_P) \Rightarrow \mathcal{F}(\text{ac}'_P)) \text{ implies } (\text{ac}_P \Rightarrow \text{ac}'_P).$$

Proof.

We have to show that $\text{ac}_P \Rightarrow \text{ac}'_P$, i.e., according to Definition 27 that $\forall p : P \rightarrow G. ((p \models \text{ac}_P) \Rightarrow (p \models \text{ac}'_P))$.

Fix $p : P \rightarrow G. p \models \text{ac}_P$. Then by Lemma 11 we have that $\mathcal{F}(p) \models \mathcal{F}(\text{ac}_P)$. Using the assumption that $\mathcal{F}(\text{ac}_P) \Rightarrow \mathcal{F}(\text{ac}'_P)$ and Definition 27 we also have that $\mathcal{F}(p) \models \mathcal{F}(\text{ac}'_P)$. This implies by Lemma 11 that $p \models \text{ac}'_P$. □

Lemma 20: ($\mathcal{F}(\text{AC})$ -Compatibility Implies AC-Compatibility, see page 111)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms and creates (\mathcal{M}) -morphisms. Then two AC-disregarding transformations are AC-compatible if they are $\mathcal{F}(\text{AC})$ -compatible.

Proof.

Transformations $P_1 \xleftarrow{\rho_1, \sigma_1} K \xrightarrow{\rho_2, \sigma_2} P_2$ are $\mathcal{F}(\text{AC})$ -compatible

$$\stackrel{\text{Def. 58}}{\Leftrightarrow} (\mathcal{F}(\text{ac}_K) \wedge \mathcal{F}(\text{ac}_K^*)) \Rightarrow (\mathcal{F}(\text{ac}(t_1)) \wedge \mathcal{F}(\text{ac}(t_2))), \text{ where } t_i \triangleq K \xrightarrow{\rho_i, \sigma_i} P_i \xrightarrow{t'_i} K'$$

are extended AC-disregarding transformations with derived nested application conditions $\text{ac}(t_i)$ on K for $i \in \{1, 2\}$

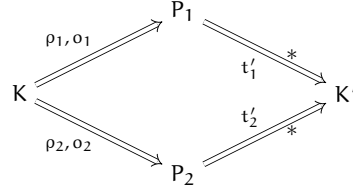
$$\stackrel{\text{Def. 47}}{\Leftrightarrow} \mathcal{F}(\text{ac}_K \wedge \text{ac}_K^*) \Rightarrow \mathcal{F}(\text{ac}(t_1) \wedge \text{ac}(t_2)), \text{ where } t_i \triangleq K \xrightarrow{\rho_i, \sigma_i} P_i \xrightarrow{t'_i} K' \text{ are}$$

extended AC-disregarding transformations with derived nested application conditions $\text{ac}(t_i)$ on K for $i \in \{1, 2\}$

$$\stackrel{\text{Lem. 19}}{\Rightarrow} (\text{ac}_K \wedge \text{ac}_K^*) \Rightarrow (\text{ac}(t_1) \wedge \text{ac}(t_2)), \text{ where } t_i \triangleq K \xrightarrow{\rho_i, \sigma_i} P_i \xrightarrow{t'_i} K' \text{ are}$$

extended AC-disregarding transformations with derived nested application conditions $\text{ac}(t_i)$ on K for $i \in \{1, 2\}$

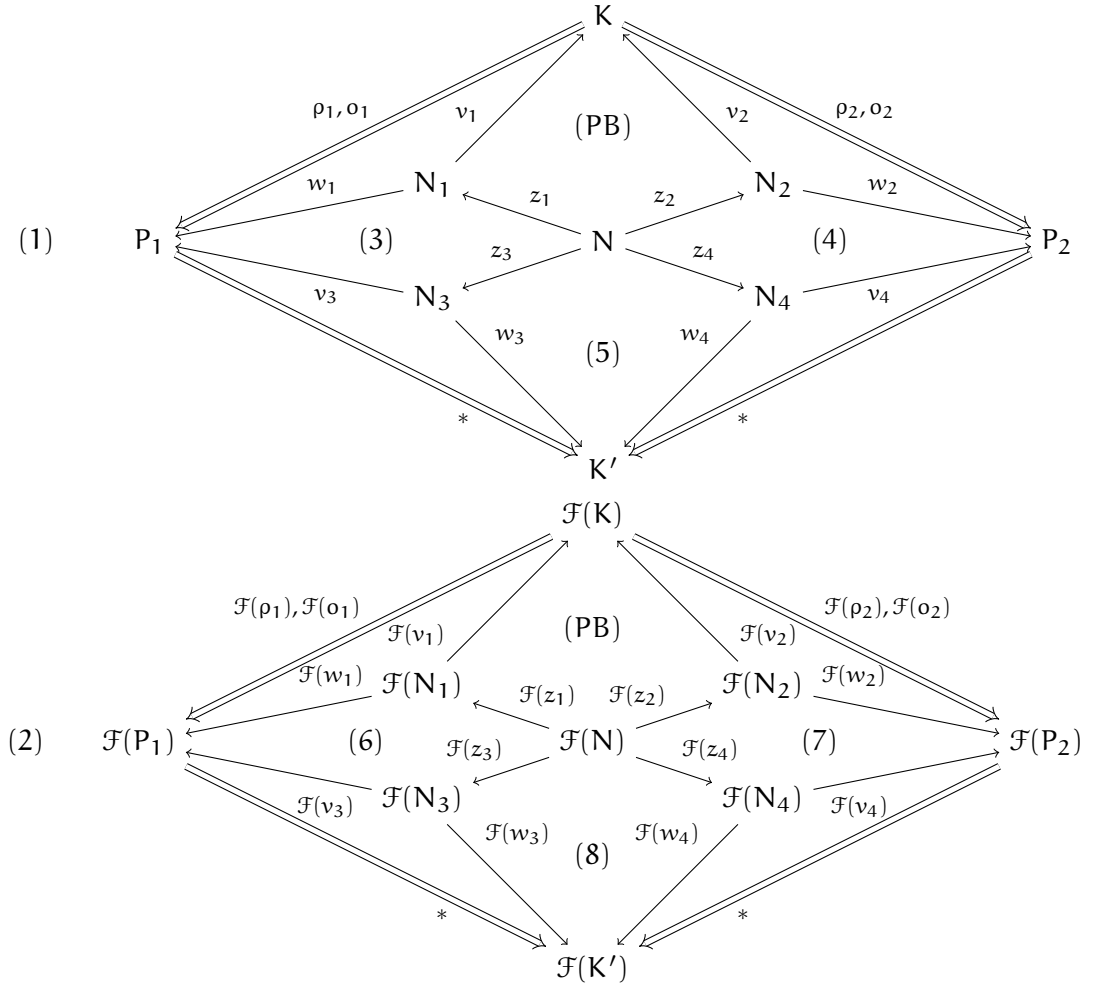
$$\stackrel{\text{Def. 28}}{\Leftrightarrow} \text{Transformations } P_1 \xleftarrow{\rho_1, \sigma_1} K \xrightarrow{\rho_2, \sigma_2} P_2 \text{ are AC-compatible}$$



□

Lemma 21: (Preservation and Creation of Plain Strict Confluence, see [page 112](#))

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves pullbacks of \mathcal{M} -morphisms as well as creates morphisms, and local confluence diagrams (1) in AS_1 and (2) in AS_2 with $\mathcal{F}(1) = (2)$. Then the diagram (1) is plain strictly confluent iff the diagram (2) is plain strictly confluent.



Proof.

• (\Rightarrow) :

Let (1) be a strict confluence diagram with a pullback (N, z_1, z_2) in diagram (1) with $z_1, z_2 \in \mathcal{M}_1^1$ leading to the corresponding pullback $(\mathcal{F}(N), \mathcal{F}(z_1), \mathcal{F}(z_2))$ in diagram (2) with $\mathcal{F}(z_1), \mathcal{F}(z_2) \in \mathcal{M}_2$, because \mathcal{F} preserves pullbacks of \mathcal{M} -morphisms by assumption. If

(1) is strictly confluent, then we have according to Definition 14 that there exist morphisms $z_3 : N \rightarrow N_3$ and $z_4 : N \rightarrow N_4$, such that subdiagrams (3), (4) and (5) commute. Now, by application of functor \mathcal{F} to the morphisms z_3 and z_4 from diagram (1) we get the morphisms $\mathcal{F}(z_3)$ and $\mathcal{F}(z_4)$ in diagram (2), such that the corresponding subdiagrams (6), (7) and (8) commute as well, because \mathcal{F} preserves commuting diagrams by general functor property. This implies that (2) is a strict confluence diagram.

- (\Leftarrow):

Let (2) be a strict confluence diagram. Construct a pullback (N, z_1, z_2) in diagram (1) with $z_1, z_2 \in \mathcal{M}_1^1$ leading to the corresponding pullback $(\mathcal{F}(N), \mathcal{F}(z_1), \mathcal{F}(z_2))$ in diagram (2) with $\mathcal{F}(z_1), \mathcal{F}(z_2) \in \mathcal{M}_2$, because \mathcal{F} preserves pullbacks of \mathcal{M} -morphisms by assumption. Strict confluence of diagram (2) means according to Definition 14, that there exist morphisms $z'_3 : \mathcal{F}(N) \rightarrow \mathcal{F}(N_3)$ and $z'_4 : \mathcal{F}(N) \rightarrow \mathcal{F}(N_4)$, such that subdiagrams (6), (7) and (8) commute. Applying morphism creation property given by assumption, we get for every morphism from diagram (2) the corresponding morphism in diagram (1), such that \mathcal{F} applied to every morphism in diagram (1) equals to the corresponding given morphism in diagram (2). Furthermore, the uniqueness of morphism creation implies commutativity of every subdiagram of (1) and hence strict confluence of the whole diagram (1). \square

Lemma 22: (\mathcal{F} is Compatible with Shift-Transformation, see page 114)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that preserves and creates \mathcal{E}' -instances according to Definition 61. Then \mathcal{F} is compatible with the **Shift**-transformation, i.e.,

$$\forall ac_P, b : P \rightarrow P'. \mathcal{F}(\mathbf{Shift}(b, ac_P)) = \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(ac_P)).$$

Proof (by Induction over the Depth of a Nested Application Condition).

Fix ac_P over the object P and $b : P \rightarrow P'$.

Basis :

Let $ac_P = \text{true}$. Then we have:

$$\begin{aligned} \mathcal{F}(\mathbf{Shift}(b, \text{true})) &\stackrel{\text{Rem. 4}}{=} \mathcal{F}(\text{true}) \stackrel{\text{Def. 47}}{=} \text{true} \\ &\stackrel{\text{Rem. 4}}{=} \mathbf{Shift}(\mathcal{F}(b), \text{true}) \stackrel{\text{Def. 47}}{=} \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\text{true})) \end{aligned}$$

Induction Step :

It remains to show under the corresponding Induction Hypothesis:

1. $\forall a : P \rightarrow C. \mathcal{F}(\mathbf{Shift}(b, \exists(a, ac_C))) = \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\exists(a, ac_C)))$,
2. $\mathcal{F}(\mathbf{Shift}(b, \neg ac_P)) = \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\neg ac_P))$,
3. $\mathcal{F}(\mathbf{Shift}(b, \bigwedge_{i \in J} ac_{P,i})) = \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\bigwedge_{i \in J} ac_{P,i}))$,
4. $\mathcal{F}(\mathbf{Shift}(b, \bigvee_{i \in J} ac_{P,i})) = \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\bigvee_{i \in J} ac_{P,i}))$.

¹ Obviously it holds that $l_1, l_2 \in \mathcal{M}_1$ and thus also the opposite morphisms v_1, v_2 in the corresponding pushout diagrams from the weak critical pair definition are in \mathcal{M}_1 . Similarly, z_1, z_2 are the opposite morphisms of v_1, v_2 in the pullback diagram (PB) in diagram (1) and hence $z_1, z_2 \in \mathcal{M}_1$.

$$\begin{array}{ccc}
\text{ac}_P \triangleright P & \xrightarrow{b} & P' \\
\downarrow a & (1) & \downarrow a' \\
\text{ac}_C \triangleright C & \xrightarrow{b'} & C'
\end{array}
\qquad
\begin{array}{ccc}
\mathcal{F}(\text{ac}_P) \triangleright \mathcal{F}(P) & \xrightarrow{\mathcal{F}(b)} & \mathcal{F}(P') \\
\downarrow \mathcal{F}(a) & (2) & \downarrow a'' \\
\mathcal{F}(\text{ac}_C) \triangleright \mathcal{F}(C) & \xrightarrow{b''} & C'' = \mathcal{F}(C')
\end{array}$$

1. Fix $a : P \rightarrow C$.

- **Case 1:** $F = \{(a', b') \in \mathcal{E}'_1 \mid b' \in \mathcal{M}_1 \wedge (1) \text{ commutes}\} = \emptyset$:

$$\begin{aligned}
\mathcal{F}(\mathbf{Shift}(b, \exists(a, \text{ac}_C))) &\stackrel{\text{Rem. 4}}{=} \mathcal{F}(\text{false}) \equiv \mathcal{F}(\neg \text{true}) \\
&\stackrel{\text{Def. 47}}{=} \neg(\mathcal{F}(\text{true})) \stackrel{\text{Def. 47}}{=} \neg \text{true} \equiv \text{false} \\
&\stackrel{(*)}{=} \mathbf{Shift}(\mathcal{F}(b), \exists(\mathcal{F}(a), \mathcal{F}(\text{ac}_C))) \\
&\stackrel{\text{Def. 47}}{=} \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\exists(a, \text{ac}_C)))
\end{aligned}$$

Where $(*)$ holds by Remark 4, because $F' = \{(a'', b'') \in \mathcal{E}'_2 \mid b'' \in \mathcal{M}_2 \wedge (2) \text{ commutes}\} = \emptyset$ since \mathcal{F} creates \mathcal{E}'_1 -instances from $\overline{\mathcal{E}}'_2$ -instances.

- **Case 2:** $F = \{(a', b') \in \mathcal{E}'_1 \mid b' \in \mathcal{M}_1 \wedge (1) \text{ commutes}\} \neq \emptyset$:

$$\begin{aligned}
&\mathcal{F}(\mathbf{Shift}(b, \exists(a, \text{ac}_C))) \\
&\stackrel{\text{Rem. 4}}{=} \mathcal{F}\left(\bigvee_{(a', b') \in F} \exists(a', \mathbf{Shift}(b', \text{ac}_C))\right) \\
&\stackrel{\text{Def. 47}}{=} \bigvee_{(a', b') \in F} \mathcal{F}(\exists(a', \mathbf{Shift}(b', \text{ac}_C))) \\
&\stackrel{\text{Def. 47}}{=} \bigvee_{(a', b') \in F} \exists(\mathcal{F}(a'), \mathcal{F}(\mathbf{Shift}(b', \text{ac}_C))) \\
&\stackrel{(*_1)}{=} \bigvee_{(a', b') \in F} \exists(\mathcal{F}(a'), \mathbf{Shift}(\mathcal{F}(b'), \mathcal{F}(\text{ac}_C))) \\
&\stackrel{\text{Def. F}}{=} \bigvee_{(a', b') \in \{(a', b') \in \mathcal{E}'_1 \mid b' \in \mathcal{M}_1 \wedge a' \circ b = b' \circ a\}} \exists(\mathcal{F}(a'), \mathbf{Shift}(\mathcal{F}(b'), \mathcal{F}(\text{ac}_C))) \\
&\stackrel{\text{Index Shifting}}{=} \bigvee_{(a'', b'') \in \{(a'', b'') \in \mathcal{F}(\mathcal{E}'_1) \mid b'' \in \mathcal{F}(\mathcal{M}_1) \wedge a'' \circ \mathcal{F}(b) = b'' \circ \mathcal{F}(a)\}} \exists(a'', \mathbf{Shift}(b'', \mathcal{F}(\text{ac}_C))) \\
&\stackrel{\text{Def. 61 (1,2)} + \mathcal{M}\text{-funct. prop.}}{=} \bigvee_{(a'', b'') \in \{(a'', b'') \in \overline{\mathcal{E}}'_2 \mid b'' \in \mathcal{M}_2 \wedge a'' \circ \mathcal{F}(b) = b'' \circ \mathcal{F}(a)\}} \exists(a'', \mathbf{Shift}(b'', \mathcal{F}(\text{ac}_C))) \\
&\stackrel{\text{Def. F'}}{=} \bigvee_{(a'', b'') \in F'} \exists(a'', \mathbf{Shift}(b'', \mathcal{F}(\text{ac}_C))) \\
&\stackrel{\text{Rem. 4} + (*_2)}{=} \mathbf{Shift}(\mathcal{F}(b), \exists(\mathcal{F}(a), \mathcal{F}(\text{ac}_C))) \\
&\stackrel{\text{Def. 47}}{=} \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\exists(a, \text{ac}_C)))
\end{aligned}$$

$(*_1)$: Is equal by induction hypothesis, which states that for some ac_C over the object C holds: $\forall b' : C \rightarrow C'. \mathcal{F}(\mathbf{Shift}(b', \text{ac}_C)) = \mathbf{Shift}(\mathcal{F}(b'), \mathcal{F}(\text{ac}_C))$.

($*_2$): $F' \neq \emptyset$, because each \mathcal{E}'_1 -instance from $F \neq \emptyset$, as given by assumption of the considered case, can be translated into the corresponding $\bar{\mathcal{E}}'_2$ -instance since $\bar{\mathcal{E}}'_2 = \mathcal{F}(\mathcal{E}'_1)$ by Definition 61.

Furthermore, index shifting means the following:

$$\bigvee_{(i_1, i_2) \in A} G(\mathcal{F}(i_1), \mathcal{F}(i_2)) = \bigvee_{(j_1, j_2) \in \mathcal{F}(A)} G(j_1, j_2),$$

where $G(x, y) \triangleq \exists(x, \mathbf{Shift}(y, \mathcal{F}(\text{ac}_C)))$.

2.

$$\begin{aligned} \mathcal{F}(\mathbf{Shift}(b, \neg \text{ac}_P)) &\stackrel{\text{Rem. 4}}{=} \mathcal{F}(\neg(\mathbf{Shift}(b, \text{ac}_P))) \\ &\stackrel{\text{Def. 47}}{=} \neg(\mathcal{F}(\mathbf{Shift}(b, \text{ac}_P))) \stackrel{(*_3)}{=} \neg(\mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\text{ac}_P))) \\ &\stackrel{\text{Rem. 4}}{=} \mathbf{Shift}(\mathcal{F}(b), \neg(\mathcal{F}(\text{ac}_P))) \stackrel{\text{Def. 47}}{=} \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\neg \text{ac}_P)) \end{aligned}$$

($*_3$): Is equal by induction hypothesis, which states that for some ac_P over the object P holds: $\forall b : P \rightarrow P'. \mathcal{F}(\mathbf{Shift}(b, \text{ac}_P)) = \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\text{ac}_P))$.

3.

$$\begin{aligned} \mathcal{F}(\mathbf{Shift}(b, \bigwedge_{i \in \mathcal{I}} \text{ac}_{P,i})) &\stackrel{\text{Rem. 4}}{=} \mathcal{F}(\bigwedge_{i \in \mathcal{I}} \mathbf{Shift}(b, \text{ac}_{P,i})) \\ &\stackrel{\text{Def. 47}}{=} \bigwedge_{i \in \mathcal{I}} \mathcal{F}(\mathbf{Shift}(b, \text{ac}_{P,i})) \stackrel{(*_4)}{=} \bigwedge_{i \in \mathcal{I}} \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\text{ac}_{P,i})) \\ &\stackrel{\text{Rem. 4}}{=} \mathbf{Shift}(\mathcal{F}(b), \bigwedge_{i \in \mathcal{I}} \mathcal{F}(\text{ac}_{P,i})) \stackrel{\text{Def. 47}}{=} \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\bigwedge_{i \in \mathcal{I}} \text{ac}_{P,i})) \end{aligned}$$

4.

$$\begin{aligned} \mathcal{F}(\mathbf{Shift}(b, \bigvee_{i \in \mathcal{I}} \text{ac}_{P,i})) &\stackrel{\text{Rem. 4}}{=} \mathcal{F}(\bigvee_{i \in \mathcal{I}} \mathbf{Shift}(b, \text{ac}_{P,i})) \\ &\stackrel{\text{Def. 47}}{=} \bigvee_{i \in \mathcal{I}} \mathcal{F}(\mathbf{Shift}(b, \text{ac}_{P,i})) \stackrel{(*_4)}{=} \bigvee_{i \in \mathcal{I}} \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\text{ac}_{P,i})) \\ &\stackrel{\text{Rem. 4}}{=} \mathbf{Shift}(\mathcal{F}(b), \bigvee_{i \in \mathcal{I}} \mathcal{F}(\text{ac}_{P,i})) \stackrel{\text{Def. 47}}{=} \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\bigvee_{i \in \mathcal{I}} \text{ac}_{P,i})) \end{aligned}$$

($*_4$): Is equal by induction hypothesis, which states that for some $\text{ac}_{P,i}$ ($i \in \mathcal{I}$) over the object P holds: $\forall b : P \rightarrow P'. \mathcal{F}(\mathbf{Shift}(b, \text{ac}_{P,i})) = \mathbf{Shift}(\mathcal{F}(b), \mathcal{F}(\text{ac}_{P,i}))$.

□

Lemma 23: (\mathcal{F} is Compatible with L-Transformation, see page 114)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that translates and creates rule applicability according to Theorem 3 as well as preserves pullbacks of \mathcal{M} -morphisms. Then \mathcal{F} is compatible with the L-transformation, i.e.,

$$\forall \text{ac}_R, \rho \in P \cup P^*. \mathcal{F}(\mathbf{L}(\rho, \text{ac}_R)) = \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\text{ac}_R))$$

where P^* is a set of derived rules in AS_1 .

Proof (by Induction over the Depth of a Nested Application Condition).

Fix ac_R over the object R and $\rho \in P \cup P^*$.

Basis :

Let $\text{ac}_R = \text{true}$. Then we have:

$$\begin{aligned} \mathcal{F}(\mathbf{L}(\rho, \text{true})) &\stackrel{\text{Rem. 5}}{=} \mathcal{F}(\text{true}) \stackrel{\text{Def. 47}}{=} \text{true} \\ &\stackrel{\text{Rem. 5}}{=} \mathbf{L}(\mathcal{F}(\rho), \text{true}) \stackrel{\text{Def. 47}}{=} \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\text{true})) \end{aligned}$$

Induction Step :

It remains to show under the corresponding Induction Hypothesis:

1. $\forall a : R \rightarrow H. \mathcal{F}(\mathbf{L}(\rho, \exists(a, \text{ac}_H))) = \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\exists(a, \text{ac}_H)))$
2. $\mathcal{F}(\mathbf{L}(\rho, \neg \text{ac}_R)) = \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\neg \text{ac}_R))$
3. $\mathcal{F}(\mathbf{L}(\rho, \bigwedge_{i \in \mathcal{I}} \text{ac}_{R,i})) = \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\bigwedge_{i \in \mathcal{I}} \text{ac}_{R,i}))$
4. $\mathcal{F}(\mathbf{L}(\rho, \bigvee_{i \in \mathcal{I}} \text{ac}_{R,i})) = \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\bigvee_{i \in \mathcal{I}} \text{ac}_{R,i}))$

$$\begin{array}{ccccc} \rho : & \begin{array}{ccccc} & & \text{ac}_R & & \\ & & \nabla & & \\ L & \xleftarrow{l} & K & \xrightarrow{r} & R \end{array} & \mathcal{F}(\rho) : & \begin{array}{ccccc} & & \mathcal{F}(\text{ac}_R) & & \\ & & \nabla & & \\ \mathcal{F}(L) & \xleftarrow{\mathcal{F}(l)} & \mathcal{F}(K) & \xrightarrow{\mathcal{F}(r)} & \mathcal{F}(R) \end{array} \\ & \begin{array}{ccc} \downarrow b & (2) & \downarrow \\ \rho^* : & G & \xleftarrow{l^*} D \end{array} & \begin{array}{ccc} \downarrow a & (1) & \downarrow \\ & H & \end{array} & \begin{array}{ccc} \downarrow \mathcal{F}(b) & (4) & \downarrow \\ \mathcal{F}(\rho^*) : & \mathcal{F}(G) & \xleftarrow{\mathcal{F}(l^*)} \mathcal{F}(D) \end{array} & \begin{array}{ccc} \downarrow \mathcal{F}(a) & (3) & \downarrow \\ & \mathcal{F}(H) & \end{array} \\ & \begin{array}{ccc} \Delta & & \Delta \\ \mathbf{L}(\rho^*, \text{ac}_H) & & \text{ac}_H \end{array} & & \begin{array}{ccc} \Delta & & \Delta \\ \mathbf{L}(\mathcal{F}(\rho^*), \mathcal{F}(\text{ac}_H)) & & \mathcal{F}(\text{ac}_H) \end{array} \end{array}$$

1. Fix $a : R \rightarrow H$.

- **Case 1:** (r, a) has no pushout complement in (1):

$$\begin{aligned} \mathcal{F}(\mathbf{L}(\rho, \exists(a, \text{ac}_H))) &\stackrel{\text{Rem. 5}}{=} \mathcal{F}(\text{false}) \equiv \mathcal{F}(\neg \text{true}) \\ &\stackrel{\text{Def. 47}}{=} \neg(\mathcal{F}(\text{true})) \stackrel{\text{Def. 47}}{=} \neg \text{true} \equiv \text{false} \\ &\stackrel{(*_1)}{=} \mathbf{L}(\mathcal{F}(\rho), \exists(\mathcal{F}(a), \mathcal{F}(\text{ac}_H))) \\ &\stackrel{\text{Def. 47}}{=} \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\exists(a, \text{ac}_H))) \end{aligned}$$

Where $(*_1)$ holds because $(\mathcal{F}(r), \mathcal{F}(a))$ has no pushout complement in (3) since \mathcal{F} creates rule applicability.

- **Case 2:** (r, a) has a pushout complement in (1) and $\rho^* = (G \leftarrow D \rightarrow H)$ is the derived rule by constructing the pushout (2):

$$\begin{aligned} \mathcal{F}(\mathbf{L}(\rho, \exists(a, \text{ac}_H))) &\stackrel{\text{Rem. 5}}{=} \mathcal{F}(\exists(b, \mathbf{L}(\rho^*, \text{ac}_H))) \\ &\stackrel{\text{Def. 47}}{=} \exists(\mathcal{F}(b), \mathcal{F}(\mathbf{L}(\rho^*, \text{ac}_H))) \\ &\stackrel{(*_2)}{=} \exists(\mathcal{F}(b), \mathbf{L}(\mathcal{F}(\rho^*), \mathcal{F}(\text{ac}_H))) \\ &\stackrel{\text{Rem. 5} + (*_3)}{=} \mathbf{L}(\mathcal{F}(\rho), \exists(\mathcal{F}(a), \mathcal{F}(\text{ac}_H))) \\ &\stackrel{\text{Def. 47}}{=} \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\exists(a, \text{ac}_H))) \end{aligned}$$

Where $(*_2)$ holds by induction hypothesis, which states that for some ac_H over the object H holds: $\forall \rho^* \in P^*. \mathcal{F}(\mathbf{L}(\rho^*, \text{ac}_H) = \mathbf{L}(\mathcal{F}(\rho^*), \mathcal{F}(\text{ac}_H)))$ and $(*_3)$ holds, because $(\mathcal{F}(r), \mathcal{F}(a))$ has a pushout complement in (3) since \mathcal{F} translates rule applicability and $\mathcal{F}(\rho^*)$ is the derived rule of the translated direct transformation.

2.

$$\begin{aligned} \mathcal{F}(\mathbf{L}(\rho, \neg \text{ac}_R)) &\stackrel{\text{Rem. 5}}{=} \mathcal{F}(\neg(\mathbf{L}(\rho, \text{ac}_R))) \\ &\stackrel{\text{Def. 47}}{=} \neg(\mathcal{F}(\mathbf{L}(\rho, \text{ac}_R))) \stackrel{(*_4)}{=} \neg(\mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\text{ac}_R))) \\ &\stackrel{\text{Rem. 5}}{=} \mathbf{L}(\mathcal{F}(\rho), \neg(\mathcal{F}(\text{ac}_R))) \stackrel{\text{Def. 47}}{=} \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\neg \text{ac}_R)) \end{aligned}$$

Where $(*_4)$ holds by induction hypothesis, which states that for some ac_R over the object R holds: $\forall \rho \in P. \mathcal{F}(\mathbf{L}(\rho, \text{ac}_R)) = \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\text{ac}_R))$.

3.

$$\begin{aligned} \mathcal{F}(\mathbf{L}(\rho, \bigwedge_{i \in \mathcal{I}} \text{ac}_{R,i})) &\stackrel{\text{Rem. 5}}{=} \mathcal{F}(\bigwedge_{i \in \mathcal{I}} \mathbf{L}(\rho, \text{ac}_{R,i})) \\ &\stackrel{\text{Def. 47}}{=} \bigwedge_{i \in \mathcal{I}} \mathcal{F}(\mathbf{L}(\rho, \text{ac}_{R,i})) \stackrel{(*_5)}{=} \bigwedge_{i \in \mathcal{I}} \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\text{ac}_{R,i})) \\ &\stackrel{\text{Rem. 5}}{=} \mathbf{L}(\mathcal{F}(\rho), \bigwedge_{i \in \mathcal{I}} \mathcal{F}(\text{ac}_{R,i})) \stackrel{\text{Def. 47}}{=} \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\bigwedge_{i \in \mathcal{I}} \text{ac}_{R,i})) \end{aligned}$$

4.

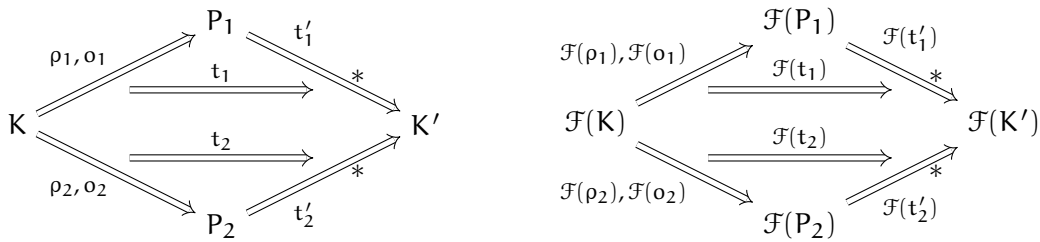
$$\begin{aligned} \mathcal{F}(\mathbf{L}(\rho, \bigvee_{i \in \mathcal{I}} \text{ac}_{R,i})) &\stackrel{\text{Rem. 5}}{=} \mathcal{F}(\bigvee_{i \in \mathcal{I}} \mathbf{L}(\rho, \text{ac}_{R,i})) \\ &\stackrel{\text{Def. 47}}{=} \bigvee_{i \in \mathcal{I}} \mathcal{F}(\mathbf{L}(\rho, \text{ac}_{R,i})) \stackrel{(*_5)}{=} \bigvee_{i \in \mathcal{I}} \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\text{ac}_{R,i})) \\ &\stackrel{\text{Rem. 5}}{=} \mathbf{L}(\mathcal{F}(\rho), \bigvee_{i \in \mathcal{I}} \mathcal{F}(\text{ac}_{R,i})) \stackrel{\text{Def. 47}}{=} \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\bigvee_{i \in \mathcal{I}} \text{ac}_{R,i})) \end{aligned}$$

Where $(*_5)$ holds by induction hypothesis, which states that for some $\text{ac}_{R,i}$ ($i \in \mathcal{I}$) over the object R holds: $\forall \rho \in P. \mathcal{F}(\mathbf{L}(\rho, \text{ac}_{R,i})) = \mathbf{L}(\mathcal{F}(\rho), \mathcal{F}(\text{ac}_{R,i}))$.

□

Lemma 24: (AC(\mathcal{F})-Compatibility Implies \mathcal{F} (AC)-Compatibility, see page 115)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates morphisms and is compatible with **Shift**- and **L**-transformations according to Lemmas 22 and 23. Then two AC-disregarding transformations $P_1 \xrightarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2 \xrightarrow{t'_2} K'$ and $\mathcal{F}(P_1) \xrightarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2) \xrightarrow{\mathcal{F}(t'_2)} \mathcal{F}(K')$ are \mathcal{F} (AC)-compatible if their corresponding translations by the \mathcal{M} -functor \mathcal{F} are AC(\mathcal{F})-compatible.



Proof.

We have to show the following three steps which lead in Step 4 to the desired result:

- **Step 1:** $\mathcal{F}(\text{ac}_K) = \text{ac}_{\mathcal{F}(K)}$

$$\begin{aligned} \mathcal{F}(\text{ac}_K) &\stackrel{\text{Def. 25}}{=} \mathcal{F}(\mathbf{Shift}(o_1, \text{ac}_{L_1}) \wedge \mathbf{Shift}(o_2, \text{ac}_{L_2})) \\ &\stackrel{\text{Lem. 22}}{=} \mathbf{Shift}(\mathcal{F}(o_1), \mathcal{F}(\text{ac}_{L_1})) \wedge \mathbf{Shift}(\mathcal{F}(o_2), \mathcal{F}(\text{ac}_{L_2})) \\ &\stackrel{\text{Def. 56}}{=} \text{ac}_{\mathcal{F}(K)} \end{aligned}$$

- **Step 2:** $\mathcal{F}(\text{ac}_K^*) = \text{ac}_{\mathcal{F}(K)}^*$

$$\mathcal{F}(\text{ac}_K^*) \stackrel{\text{Def. 25}}{=} \mathcal{F}(\neg(\text{ac}_{K,d_{12}}^* \wedge \text{ac}_{K,d_{21}}^*))$$

- **Case 1:** $\exists d_{12} : L_1 \rightarrow N_2, d_{21} : L_2 \rightarrow N_1. v_2 \circ d_{12} = o_1 \wedge v_1 \circ d_{21} = o_2$

$$\begin{aligned} &\mathcal{F}(\neg(\text{ac}_{K,d_{12}}^* \wedge \text{ac}_{K,d_{21}}^*)) \\ &\stackrel{\text{Def. 25}}{=} \mathcal{F}(\neg(\mathbf{L}(p_2^*, \mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1})) \wedge \text{ac}_{K,d_{21}}^*)) \\ &\stackrel{\text{Def. 25}}{=} \mathcal{F}(\neg(\mathbf{L}(p_2^*, \mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1})) \\ &\quad \wedge \mathbf{L}(p_1^*, \mathbf{Shift}(w_1 \circ d_{21}, \text{ac}_{L_2})))) \\ &\stackrel{\text{Def. 47}}{=} \neg(\mathcal{F}(\mathbf{L}(p_2^*, \mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1})) \\ &\quad \wedge \mathbf{L}(p_1^*, \mathbf{Shift}(w_1 \circ d_{21}, \text{ac}_{L_2})))) \\ &\stackrel{\text{Def. 47}}{=} \neg(\mathcal{F}(\mathbf{L}(p_2^*, \mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1})) \\ &\quad \wedge \mathcal{F}(\mathbf{L}(p_1^*, \mathbf{Shift}(w_1 \circ d_{21}, \text{ac}_{L_2})))) \\ &\stackrel{\text{Lem. 23}}{=} \neg(\mathbf{L}(\mathcal{F}(p_2^*), \mathcal{F}(\mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1}))) \\ &\quad \wedge \mathbf{L}(\mathcal{F}(p_1^*), \mathcal{F}(\mathbf{Shift}(w_1 \circ d_{21}, \text{ac}_{L_2})))) \\ &\stackrel{\text{Lem. 22}}{=} \neg(\mathbf{L}(\mathcal{F}(p_2^*), \mathbf{Shift}(\mathcal{F}(w_2 \circ d_{12}), \mathcal{F}(\text{ac}_{L_1}))) \\ &\quad \wedge \mathbf{L}(\mathcal{F}(p_1^*), \mathbf{Shift}(\mathcal{F}(w_1 \circ d_{21}), \mathcal{F}(\text{ac}_{L_2})))) \\ &\stackrel{\text{funct. prop.}}{=} \neg(\mathbf{L}(\mathcal{F}(p_2^*), \mathbf{Shift}(\mathcal{F}(w_2) \circ \mathcal{F}(d_{12}), \mathcal{F}(\text{ac}_{L_1}))) \\ &\quad \wedge \mathbf{L}(\mathcal{F}(p_1^*), \mathbf{Shift}(\mathcal{F}(w_1) \circ \mathcal{F}(d_{21}), \mathcal{F}(\text{ac}_{L_2})))) \\ &\stackrel{\text{Def. 56} + (*_1)}{=} \neg(\text{ac}_{\mathcal{F}(K), d'_{12}}^* \\ &\quad \wedge \mathbf{L}(\mathcal{F}(p_1^*), \mathbf{Shift}(\mathcal{F}(w_1) \circ \mathcal{F}(d_{21}), \mathcal{F}(\text{ac}_{L_2})))) \\ &\stackrel{\text{Def. 56} + (*_1)}{=} \neg(\text{ac}_{\mathcal{F}(K), d'_{12}}^* \wedge \text{ac}_{\mathcal{F}(K), d'_{21}}^*) \\ &\stackrel{\text{Def. 56}}{=} \text{ac}_{\mathcal{F}(K)}^* \end{aligned}$$

Where $(*_1)$ holds since there are morphisms $d'_{12} : \mathcal{F}(L_1) \rightarrow \mathcal{F}(N_2)$ and $d'_{21} : \mathcal{F}(L_2) \rightarrow \mathcal{F}(N_1)$ with $d'_{12} = \mathcal{F}(d_{12})$, $d'_{21} = \mathcal{F}(d_{21})$ such that $\mathcal{F}(v_2) \circ \mathcal{F}(d_{12}) = \mathcal{F}(o_1)$ and $\mathcal{F}(v_1) \circ \mathcal{F}(d_{21}) = \mathcal{F}(o_2)$, because $v_2 \circ d_{12} = o_1$ and $v_1 \circ d_{21} = o_2$ by assumption and functors preserve commuting diagrams.

- **Case 2:** $\nexists d_{12} : L_1 \rightarrow N_2. v_2 \circ d_{12} = o_1 \wedge \nexists d_{21} : L_2 \rightarrow N_1. v_1 \circ d_{21} = o_2$

$$\mathcal{F}(\neg(\text{ac}_{K,d_{12}}^* \wedge \text{ac}_{K,d_{21}}^*))$$

$$\begin{aligned}
& \stackrel{\text{Def. } 25}{=} \mathcal{F}(\neg(\text{false} \wedge \text{ac}_{\mathcal{K}, d_{21}}^*)) \\
& \stackrel{\text{Def. } 25}{=} \mathcal{F}(\neg(\text{false} \wedge \text{false})) \\
& \stackrel{\text{Def. } 47}{=} \neg(\mathcal{F}(\text{false} \wedge \text{false})) \\
& \stackrel{\text{Def. } 47}{=} \neg(\mathcal{F}(\text{false}) \wedge \mathcal{F}(\text{false})) \\
& \stackrel{\text{Def. } 47}{=} \neg(\text{false} \wedge \text{false}) \\
& \stackrel{\text{Def. } 56 + (*_2)}{=} \neg(\text{ac}_{\mathcal{F}(\mathcal{K}), d'_{12}}^* \wedge \text{false}) \\
& \stackrel{\text{Def. } 56 + (*_2)}{=} \neg(\text{ac}_{\mathcal{F}(\mathcal{K}), d'_{12}}^* \wedge \text{ac}_{\mathcal{F}(\mathcal{K}), d'_{21}}^*) \\
& \stackrel{\text{Def. } 56}{=} \text{ac}_{\mathcal{F}(\mathcal{K})}^*
\end{aligned}$$

Where $(*_2)$ holds since there are no morphisms $d'_{12} : \mathcal{F}(L_1) \rightarrow \mathcal{F}(N_2)$ and $d'_{21} : \mathcal{F}(L_2) \rightarrow \mathcal{F}(N_1)$ such that $\mathcal{F}(v_2) \circ d'_{12} = \mathcal{F}(o_1)$ and $\mathcal{F}(v_1) \circ d'_{21} = \mathcal{F}(o_2)$, because assuming that there are such morphisms and using the fact that \mathcal{F} creates morphisms we get the existence of morphisms $d_{12} : L_1 \rightarrow N_2$, $d_{21} : L_2 \rightarrow N_1$ with $d'_{12} = \mathcal{F}(d_{12})$ and $d'_{21} = \mathcal{F}(d_{21})$ such that $v_2 \circ d_{12} = o_1$ and $v_1 \circ d_{21} = o_2$, which is a contradiction to our assumption.

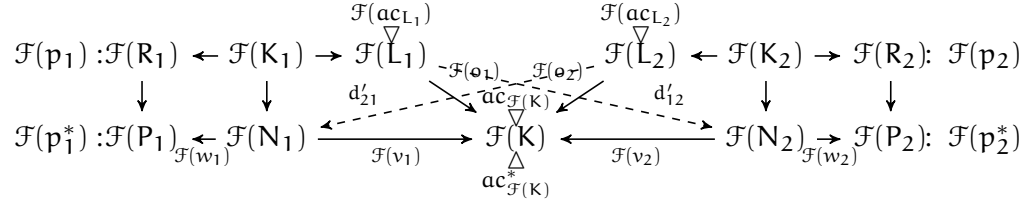
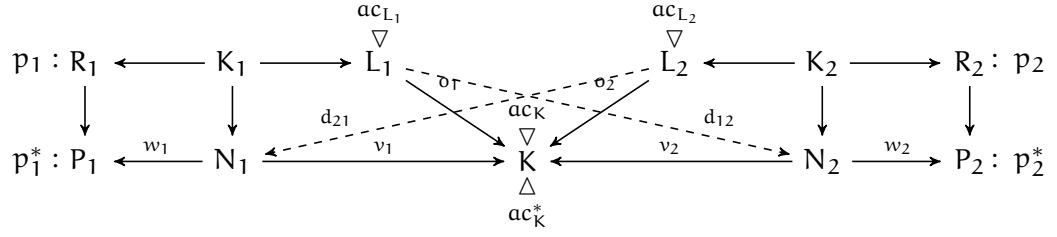
– **Case 3:** $\exists d_{12} : L_1 \rightarrow N_2. v_2 \circ d_{12} = o_1 \wedge \nexists d_{21} : L_2 \rightarrow N_1. v_1 \circ d_{21} = o_2$

$$\begin{aligned}
& \mathcal{F}(\neg(\text{ac}_{\mathcal{K}, d_{12}}^* \wedge \text{ac}_{\mathcal{K}, d_{21}}^*)) \\
& \stackrel{\text{Def. } 25}{=} \mathcal{F}(\neg(\mathbf{L}(p_2^*, \mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1})) \wedge \text{ac}_{\mathcal{K}, d_{21}}^*)) \\
& \stackrel{\text{Def. } 25}{=} \mathcal{F}(\neg(\mathbf{L}(p_2^*, \mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1})) \wedge \text{false})) \\
& \stackrel{\text{Def. } 47}{=} \neg(\mathcal{F}(\mathbf{L}(p_2^*, \mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1})) \wedge \text{false})) \\
& \stackrel{\text{Def. } 47}{=} \neg(\mathcal{F}(\mathbf{L}(p_2^*, \mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1}))) \wedge \mathcal{F}(\text{false})) \\
& \stackrel{\text{Def. } 47}{=} \neg(\mathcal{F}(\mathbf{L}(p_2^*, \mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1}))) \wedge \text{false}) \\
& \stackrel{\text{Lem. } 23}{=} \neg(\mathbf{L}(\mathcal{F}(p_2^*), \mathcal{F}(\mathbf{Shift}(w_2 \circ d_{12}, \text{ac}_{L_1}))) \wedge \text{false}) \\
& \stackrel{\text{Lem. } 22}{=} \neg(\mathbf{L}(\mathcal{F}(p_2^*), \mathbf{Shift}(\mathcal{F}(w_2 \circ d_{12}), \mathcal{F}(\text{ac}_{L_1}))) \wedge \text{false}) \\
& \stackrel{\text{funct. prop.}}{=} \neg(\mathbf{L}(\mathcal{F}(p_2^*), \mathbf{Shift}(\mathcal{F}(w_2) \circ \mathcal{F}(d_{12}), \mathcal{F}(\text{ac}_{L_1}))) \wedge \text{false}) \\
& \stackrel{\text{Def. } 56 + (*_3)}{=} \neg(\text{ac}_{\mathcal{F}(\mathcal{K}), d'_{12}}^* \wedge \text{false}) \\
& \stackrel{\text{Def. } 56 + (*_3)}{=} \neg(\text{ac}_{\mathcal{F}(\mathcal{K}), d'_{12}}^* \wedge \text{ac}_{\mathcal{F}(\mathcal{K}), d'_{21}}^*) \\
& \stackrel{\text{Def. } 56}{=} \text{ac}_{\mathcal{F}(\mathcal{K})}^*
\end{aligned}$$

Where $(*_3)$ holds since there is a morphism $d'_{12} : \mathcal{F}(L_1) \rightarrow \mathcal{F}(N_2)$ with $d'_{12} = \mathcal{F}(d_{12})$ such that $\mathcal{F}(v_2) \circ \mathcal{F}(d_{12}) = \mathcal{F}(o_1)$, because $v_2 \circ d_{12} = o_1$ by assumption and functors preserve commuting diagrams. Furthermore, there is no morphism $d'_{21} : \mathcal{F}(L_2) \rightarrow \mathcal{F}(N_1)$ such that $\mathcal{F}(v_1) \circ d'_{21} = \mathcal{F}(o_2)$, because assuming that there is such a morphism and using the fact that \mathcal{F} creates morphisms we get the existence of the morphism $d_{21} : L_2 \rightarrow N_1$ with $d'_{21} = \mathcal{F}(d_{21})$ such that $v_1 \circ d_{21} = o_2$, which is a contradiction to our assumption.

- **Case 4:** $\exists d_{21} : L_2 \rightarrow N_1.v_1 \circ d_{21} = o_2 \wedge \nexists d_{12} : L_1 \rightarrow N_2.v_2 \circ d_{12} = o_1$

This case works analogously to the Case 3.



- **Step 3:** $\mathcal{F}(\text{ac}(t_i)) = \text{ac}(\mathcal{F}(t_i))$ for $i \in \{1, 2\}$

- **Case 1:** $t_i : G_0 \Rightarrow G_0$ with the length of transformation $|t_i| = 0$

$$\begin{aligned} \mathcal{F}(\text{ac}(t_i)) &\stackrel{\text{Def. 22}}{=} \mathcal{F}(\text{true}) \\ &\stackrel{\text{Def. 47}}{=} \text{true} \stackrel{| \mathcal{F}(t_i) | = 0}{=} \text{ac}(\mathcal{F}(t_i)) \end{aligned}$$

- **Case 2:** $t_i : G_0 \xRightarrow{\rho_1, m_1} G_1$ with the length of transformation $|t_i| = 1$

$$\begin{aligned} \mathcal{F}(\text{ac}(t_i)) &\stackrel{\text{Def. 22}}{=} \mathcal{F}(\text{Shift}(m_1, \text{ac}_{L_1})) \\ &\stackrel{\text{Lem. 22}}{=} \text{Shift}(\mathcal{F}(m_1), \mathcal{F}(\text{ac}_{L_1})) \stackrel{\text{Def. 54}}{=} \text{ac}(\mathcal{F}(t_i)) \end{aligned}$$

- **Case 3:** $t_i : G_0 \xRightarrow{*} G_n \xRightarrow{\rho_{n+1}, m_{n+1}} G_{n+1}$ with the length of transformation $|t_i| \geq 2$

Proof by Induction over the length of transformation:

Basis :

Let $t_i : G_0 \Rightarrow G_1 \Rightarrow G_2$ with the length of transformation $|t_i| = 2$:

$$\begin{aligned} &\mathcal{F}(\text{ac}(t_i : G_0 \Rightarrow G_1 \Rightarrow G_2)) \\ &\stackrel{\text{Def. 22}}{=} \mathcal{F}(\text{ac}(t_1 : G_0 \Rightarrow G_1) \wedge \text{L}(p_1^*, \text{ac}(t_2 : G_1 \Rightarrow G_2))) \\ &\stackrel{|t_1|=1}{=} \mathcal{F}(\text{Shift}(m_1, \text{ac}_{L_1}) \wedge \text{L}(p_1^*, \text{ac}(t_2 : G_1 \Rightarrow G_2))) \\ &\stackrel{|t_2|=1}{=} \mathcal{F}(\text{Shift}(m_1, \text{ac}_{L_1}) \wedge \text{L}(p_1^*, \text{Shift}(m_2, \text{ac}_{L_2}))) \\ &\stackrel{\text{Def. 47}}{=} \mathcal{F}(\text{Shift}(m_1, \text{ac}_{L_1})) \wedge \mathcal{F}(\text{L}(p_1^*, \text{Shift}(m_2, \text{ac}_{L_2}))) \\ &\stackrel{\text{Lem. 23}}{=} \mathcal{F}(\text{Shift}(m_1, \text{ac}_{L_1})) \wedge \text{L}(\mathcal{F}(p_1^*), \mathcal{F}(\text{Shift}(m_2, \text{ac}_{L_2}))) \\ &\stackrel{\text{Lem. 22}}{=} \text{Shift}(\mathcal{F}(m_1), \mathcal{F}(\text{ac}_{L_1})) \wedge \text{L}(\mathcal{F}(p_1^*), \text{Shift}(\mathcal{F}(m_2), \mathcal{F}(\text{ac}_{L_2}))) \\ &\stackrel{\text{Def. 54}}{=} \text{ac}(\mathcal{F}(t_1) : \mathcal{F}(G_0) \Rightarrow \mathcal{F}(G_1)) \\ &\wedge \text{L}(\mathcal{F}(p_1^*), \text{ac}(\mathcal{F}(t_2) : \mathcal{F}(G_1) \Rightarrow \mathcal{F}(G_2))) \\ &\stackrel{\text{Def. 54}}{=} \text{ac}(\mathcal{F}(t_i) : \mathcal{F}(G_0) \Rightarrow \mathcal{F}(G_1) \Rightarrow \mathcal{F}(G_2)) \end{aligned}$$

Induction Hypothesis :

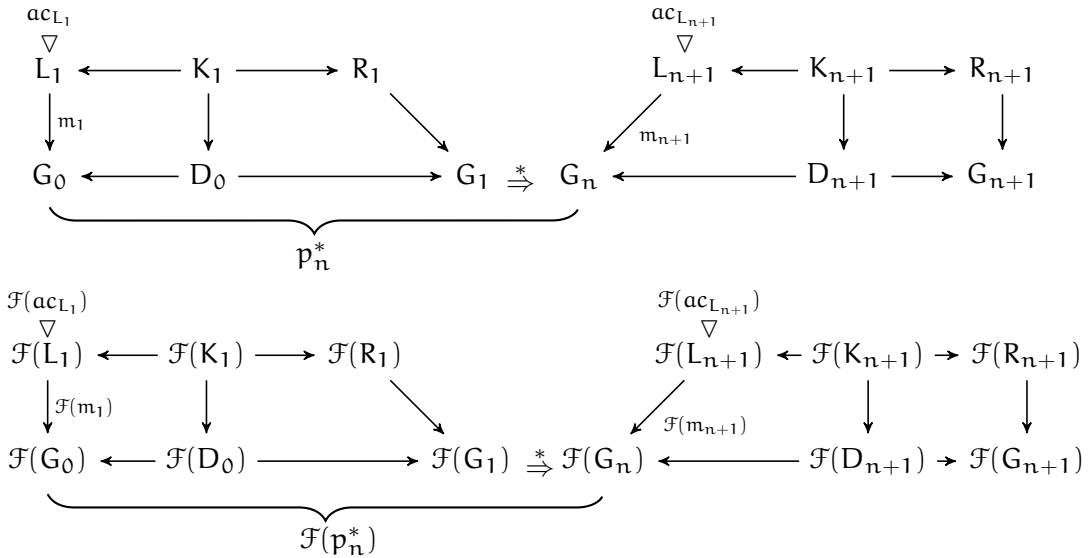
For some $t'_i : G_0 \xrightarrow{*} G_n$ with the length of transformation $|t'_i| = n \geq 2$ holds $\mathcal{F}(\text{ac}(t'_i : G_0 \xrightarrow{*} G_n)) = \text{ac}(\mathcal{F}(t'_i) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n))$.

It remains to show:

For $t_i : G_0 \xrightarrow{*} G_n \Rightarrow G_{n+1}$ with the length of transformation $|t_i| = n + 1 \geq 3$ holds $\mathcal{F}(\text{ac}(t_i : G_0 \xrightarrow{*} G_{n+1})) = \text{ac}(\mathcal{F}(t_i) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_{n+1}))$.

Induction Step :

$$\begin{aligned}
& \mathcal{F}(\text{ac}(t_i : G_0 \xrightarrow{*} G_{n+1})) \\
& \stackrel{\text{Def. 22}}{=} \mathcal{F}(\text{ac}(t'_i : G_0 \xrightarrow{*} G_n) \wedge \mathbf{L}(p_n^*, \text{ac}(t_{n+1} : G_n \Rightarrow G_{n+1}))) \\
& \stackrel{\text{Def. 47}}{=} \mathcal{F}(\text{ac}(t'_i : G_0 \xrightarrow{*} G_n)) \wedge \mathcal{F}(\mathbf{L}(p_n^*, \text{ac}(t_{n+1} : G_n \Rightarrow G_{n+1}))) \\
& \stackrel{\text{Ind. H.}}{=} \text{ac}(\mathcal{F}(t'_i) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n)) \wedge \mathcal{F}(\mathbf{L}(p_n^*, \text{ac}(t_{n+1} : G_n \Rightarrow G_{n+1}))) \\
& \stackrel{\text{Lem. 23}}{=} \text{ac}(\mathcal{F}(t'_i) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n)) \wedge \mathbf{L}(\mathcal{F}(p_n^*), \mathcal{F}(\text{ac}(t_{n+1} : G_n \Rightarrow G_{n+1}))) \\
& \stackrel{|t_{n+1}|=1}{=} \text{ac}(\mathcal{F}(t'_i) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n)) \wedge \mathbf{L}(\mathcal{F}(p_n^*), \mathcal{F}(\text{Shift}(m_{n+1}, \text{ac}_{L_{n+1}}))) \\
& \stackrel{\text{Lem. 22}}{=} \text{ac}(\mathcal{F}(t'_i) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n)) \wedge \mathbf{L}(\mathcal{F}(p_n^*), \text{Shift}(\mathcal{F}(m_{n+1}), \mathcal{F}(\text{ac}_{L_{n+1}}))) \\
& \stackrel{\text{Def. 54}}{=} \text{ac}(\mathcal{F}(t'_i) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_n)) \\
& \wedge \mathbf{L}(\mathcal{F}(p_n^*), \text{ac}(\mathcal{F}(t_{n+1}) : \mathcal{F}(G_n) \Rightarrow \mathcal{F}(G_{n+1}))) \\
& \stackrel{\text{Def. 54}}{=} \text{ac}(\mathcal{F}(t_i) : \mathcal{F}(G_0) \xrightarrow{*} \mathcal{F}(G_{n+1}))
\end{aligned}$$



- **Step 4:** Then we have the following.

Translated transformations $\mathcal{F}(P_1) \xrightarrow{\mathcal{F}(\rho_1), \mathcal{F}(\sigma_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(\sigma_2)} \mathcal{F}(P_2)$ are AC(\mathcal{F})-compatible

$\stackrel{\text{Def. 59}}{\Leftrightarrow} (\text{ac}_{\mathcal{F}(K)} \wedge \text{ac}_{\mathcal{F}(K)}^*) \Rightarrow (\text{ac}(\mathcal{F}(t_1)) \wedge \text{ac}(\mathcal{F}(t_2)))$, where $\mathcal{F}(t_i) \triangleq \mathcal{F}(K)$

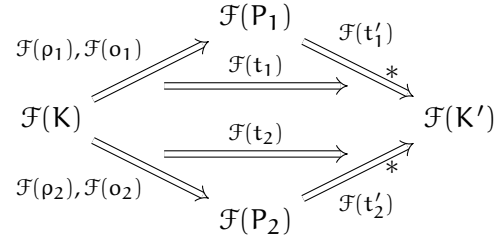
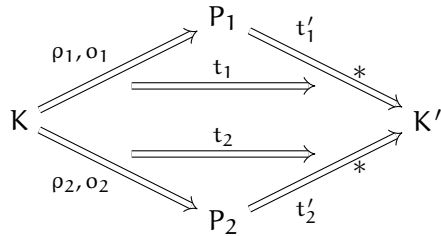
$\mathcal{F}(\rho_i), \mathcal{F}(\sigma_i) \xrightarrow{\mathcal{F}(t'_i)} \mathcal{F}(P_i) \xrightarrow{\mathcal{F}(t'_i)^*} \mathcal{F}(K')$ are translated extended AC-disregarding transformations with translated derived nested application conditions $\text{ac}(\mathcal{F}(t_i))$ on $\mathcal{F}(K)$

for $i \in \{1, 2\}$

$\xRightarrow{\text{Steps } 1, 2, 3} (\mathcal{F}(\text{ac}_K) \wedge \mathcal{F}(\text{ac}_K^*)) \Rightarrow (\mathcal{F}(\text{ac}(t_1)) \wedge \mathcal{F}(\text{ac}(t_2))),$ where $t_i \triangleq K \xRightarrow{\rho_i, \sigma_i} P_i$

$\xRightarrow{t'_i *} K'$ are extended AC-disregarding transformations with derived nested application conditions $\text{ac}(t_i)$ on K for $i \in \{1, 2\}$

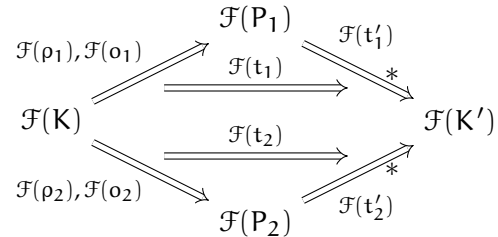
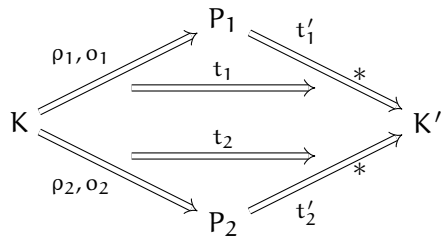
$\xLeftrightarrow{\text{Def. 58}} \text{Transformations } P_1 \xLeftrightarrow{\rho_1, \sigma_1} K \xRightarrow{\rho_2, \sigma_2} P_2 \text{ are } \mathcal{F}(\text{AC})\text{-compatible}$



□

Lemma 25: (AC(\mathcal{F})-Compatibility Implies AC-Compatibility [213], see page 115)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (\mathcal{M} -)morphisms and is compatible with **Shift**- and **L**-transformations according to Lemmas 22 and 23. Then two AC-disregarding transformations $P_1 \xLeftrightarrow{\rho_1, \sigma_1} K \xRightarrow{\rho_2, \sigma_2} P_2$ are AC-compatible if their corresponding translations by the \mathcal{M} -functor \mathcal{F} are AC(\mathcal{F})-compatible.



Proof.

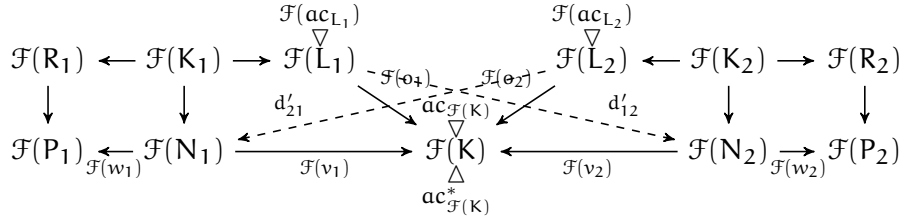
Assume that two translated AC-disregarding transformations $\mathcal{F}(P_1) \xLeftrightarrow{\mathcal{F}(\rho_1), \mathcal{F}(\sigma_1)} \mathcal{F}(K) \xRightarrow{\mathcal{F}(\rho_2), \mathcal{F}(\sigma_2)} \mathcal{F}(P_2)$ are AC(\mathcal{F})-compatible. Then by Lemma 24 we also have that the transformations $P_1 \xLeftrightarrow{\rho_1, \sigma_1} K \xRightarrow{\rho_2, \sigma_2} P_2$ are $\mathcal{F}(\text{AC})$ -compatible. This implies the AC-compatibility of AC-disregarding transformations $P_1 \xLeftrightarrow{\rho_1, \sigma_1} K \xRightarrow{\rho_2, \sigma_2} P_2$ by application of Lemma 20. □

Lemma 26: (Preservation of (Weak) Critical Pairs [213], see page 116)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$, where P are rules with nested application conditions, a (weak) critical pair $P_1 \xLeftrightarrow{\rho_1, \sigma_1} K \xRightarrow{\rho_2, \sigma_2} P_2$ in AS_1 , and an \mathcal{M} -functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$ that creates (\mathcal{M} -)morphisms and is compatible with $\mathcal{E}' - \mathcal{M}$ pair factorization as well as with **Shift**- and **L**-transformations. Then $\mathcal{F}(P_1) \xLeftrightarrow{\mathcal{F}(\rho_1), \mathcal{F}(\sigma_1)} \mathcal{F}(K) \xRightarrow{\mathcal{F}(\rho_2), \mathcal{F}(\sigma_2)} \mathcal{F}(P_2)$ is the corresponding \mathcal{F} -reachable (weak) critical pair in AS_2 .

Proof.

- Let $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ be a weak critical pair in AS_1 . This means according to Definition 25, that $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ is a pair of AC-disregarding transformations with $(o_1, o_2) \in \mathcal{E}'_1$. Applying the assumption that \mathcal{F} translates rule applicability, we get the corresponding AC-disregarding pair of transformations $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ in AS_2 by constructing for each transformation within the double pushout diagram first the corresponding pushout complement, because \mathcal{F} preserves pushouts along \mathcal{M} -morphisms according to the \mathcal{M} -functor property from Definition 41, and subsequently the second pushout diagram, because of the same reason as mentioned before (for more details see the proof of Theorem 1). This pair of transformations in AS_2 (see the picture below) is an \mathcal{F} -reachable critical pair in the sense of the plain case, because all objects and morphisms within the constructed double pushout diagrams are \mathcal{F} -images and we also have $(\mathcal{F}(o_1), \mathcal{F}(o_2)) \in \mathcal{E}'_2$, because \mathcal{F} is compatible with pair factorization by assumption. So we get that $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ is an \mathcal{F} -reachable weak critical pair.

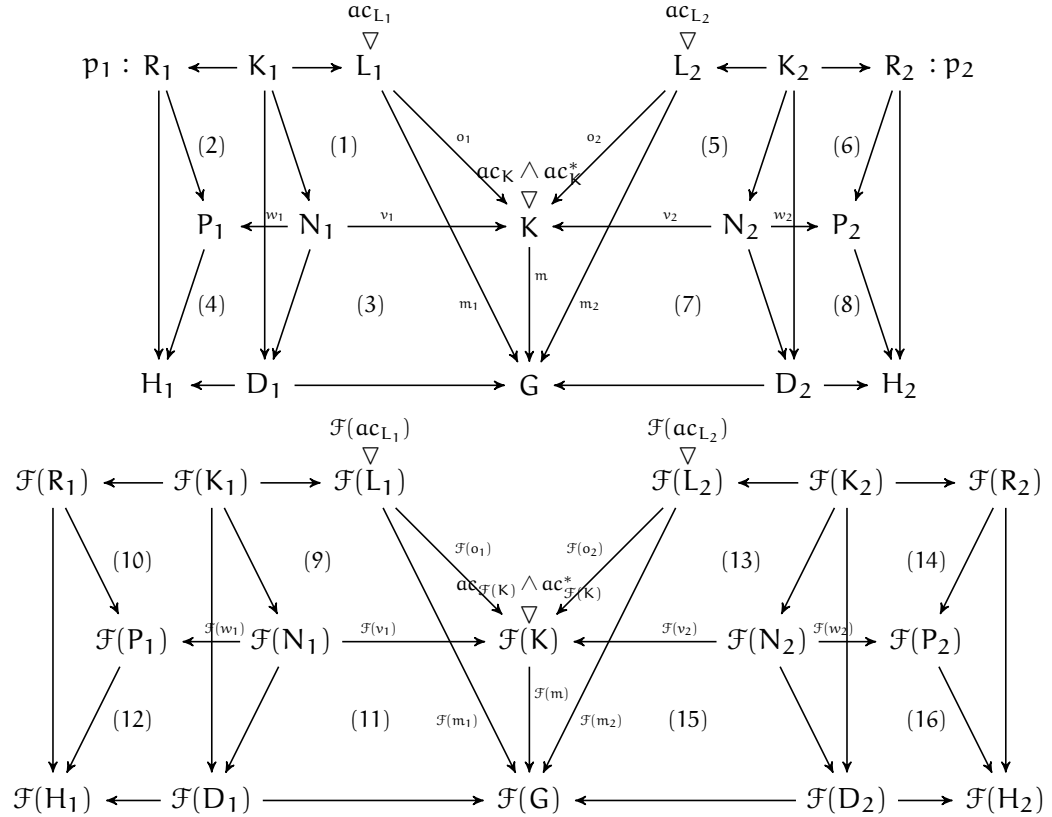


- Let $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ be a critical pair in AS_1 . This means according to Definition 26, that $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$ is a weak critical pair with induced extension and conflict-inducing nested application conditions ac_K and ac_K^* , respectively, and it holds additionally that if we embed the considered weak critical pair into a pair of AC-regarding transformations $H_1 \xleftarrow{\rho_1, m_1} G \xrightarrow{\rho_2, m_2} H_2$, then there is an \mathcal{M}_1 -morphism $m : K \rightarrow G$, such that $m \models ac_K \wedge ac_K^*$ and morphisms $m_i = m \circ o_i$ for $i \in \{1, 2\}$ satisfy the gluing condition, i.e., m_i has a pushout complement D_i with respect to the plain derived rule p_i for $i \in \{1, 2\}$ (see the picture below, where diagrams (1)-(8) are pushouts). Now we can construct the corresponding pair of translated AC-regarding transformations $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$, because \mathcal{F} translates rule applicability, i.e., for both transformations we have pushout complements $\mathcal{F}(D_1)$, $\mathcal{F}(D_2)$ and can subsequently construct the respective second pushout completing the corresponding double pushout diagrams (for details see the proof of Theorem 1). Furthermore, applying \mathcal{F} to the weak critical pair $P_1 \xleftarrow{\rho_1, o_1} K \xrightarrow{\rho_2, o_2} P_2$, we get the \mathcal{F} -reachable weak critical pair $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ with induced translated extension and conflict-inducing nested application conditions on $\mathcal{F}(K)$, $ac_{\mathcal{F}(K)}$ and $ac_{\mathcal{F}(K)}^*$, respectively, according to the previous part of the proof. Because \mathcal{F} is compatible with pair factorization, we have that there is the morphism $\mathcal{F}(m) : \mathcal{F}(K) \rightarrow \mathcal{F}(G)$ such that $\mathcal{F}(m) \circ \mathcal{F}(o_i) = \mathcal{F}(m_i)$ for $i \in \{1, 2\}$ and $\mathcal{F}(m) \in \mathcal{M}_2$. Moreover, because \mathcal{F} translates pushouts along \mathcal{M} -morphisms according to the \mathcal{M} -functor property, we get that also the diagrams (9)-(16) are pushouts obtaining the corresponding embedding of the \mathcal{F} -reachable weak critical pair $\mathcal{F}(P_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ into the considered pair of translated transformations $\mathcal{F}(H_1) \xleftarrow{\mathcal{F}(\rho_1), \mathcal{F}(m_1)} \mathcal{F}(G) \xrightarrow{\mathcal{F}(\rho_2), \mathcal{F}(m_2)} \mathcal{F}(H_2)$.

It remains to show, that also $\mathcal{F}(m) \models \text{ac}_{\mathcal{F}(K)} \wedge \text{ac}_{\mathcal{F}(K)}^*$ or equivalently $\mathcal{F}(m) \models \mathcal{F}(\text{ac}_K \wedge \text{ac}_K^*)$, because the following holds:

$$\text{ac}_{\mathcal{F}(K)} \wedge \text{ac}_{\mathcal{F}(K)}^* \stackrel{\text{Proof of Lem. 24}}{=} \mathcal{F}(\text{ac}_K) \wedge \mathcal{F}(\text{ac}_K^*) \stackrel{\text{Def. 47}}{=} \mathcal{F}(\text{ac}_K \wedge \text{ac}_K^*)$$

According to Definition 26 as given before, we already know that $m \models \text{ac}_K \wedge \text{ac}_K^*$. This implies using Lemma 11 that $\mathcal{F}(m) \models \mathcal{F}(\text{ac}_K \wedge \text{ac}_K^*)$, which was to be shown. Hence $\mathcal{F}(P_1) \xrightarrow{\mathcal{F}(p_1), \mathcal{F}(o_1)} \mathcal{F}(K) \xrightarrow{\mathcal{F}(p_2), \mathcal{F}(o_2)} \mathcal{F}(P_2)$ is the corresponding \mathcal{F} -reachable critical pair in AS_2 .



□

Lemma 58: (\mathcal{F}_C Satisfies Required Properties, see page 215)

Consider \mathcal{M} -adhesive transformation systems $\text{AS}_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $\text{AS}_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}(P))$ such that $(\mathbf{C}_1, \mathcal{M}_1)$ has \mathcal{E}'_1 - \mathcal{M}_1 pair factorization and initial pushouts, a functor $\mathcal{F} : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}_2, \mathcal{M}_2)$, and functors $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \rightarrow (\mathbf{C}'_2, \mathcal{M}_2^*)$, $\mathcal{F}_C^{-1} : (\mathbf{C}'_2, \mathcal{M}_2^*) \rightarrow (\mathbf{C}_1, \mathcal{M}_1)$ building a category equivalence $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{C}'_2, \mathcal{M}_2^*)$ where $(\mathbf{C}'_2, \mathcal{M}_2^*)$ is the subcategory of $(\mathbf{C}_2, \mathcal{M}_2)$, in which all non- \mathcal{F} -images have been removed, and where the functor \mathcal{F}_C is the restriction of \mathcal{F} . Then the functor \mathcal{F}_C satisfies the properties listed in Definition 69.

Proof.

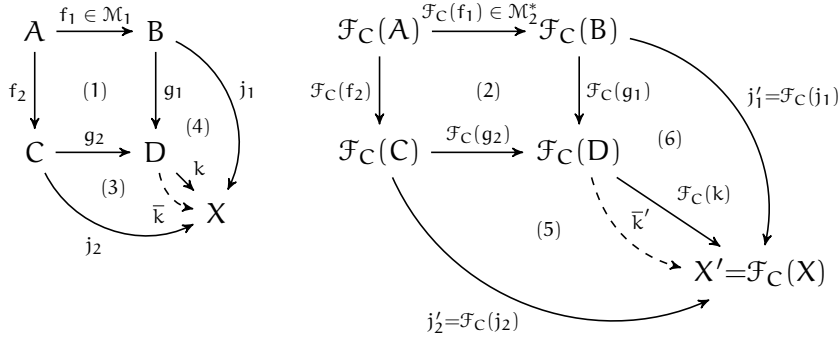
1. \mathcal{F}_C preserves monomorphisms, i.e., $\mathcal{F}_C(\mathcal{M}_1) \subseteq \mathcal{M}_2^*$:

Since we have by assumption that the functor \mathcal{F}_C is a restriction of \mathcal{F} , it holds that $\mathcal{F}_C(\mathcal{M}_1) = \mathcal{F}(\mathcal{M}_1) = \mathcal{M}_2^*$, which directly implies that $\mathcal{F}_C(\mathcal{M}_1) \subseteq \mathcal{M}_2^*$.

2. \mathcal{F}_C preserves pushouts along \mathcal{M} -morphisms:

Let (1) be a pushout along \mathcal{M}_1 -morphisms in \mathbf{C}_1 as given below with $f_1 \in \mathcal{M}_1$. Applying

the functor \mathcal{F}_C to the diagram (1), we get the diagram (2), which obviously commutes, because functors preserve commuting diagrams by general functor property. Moreover, $\mathcal{F}_C(f_1) \in \mathcal{M}_2^*$ because the functor \mathcal{F}_C preserves monomorphisms as shown before. It remains to show the universal property for the diagram (2). Let X' be a comparison object with $j'_1 : \mathcal{F}_C(B) \rightarrow X'$, $j'_2 : \mathcal{F}_C(C) \rightarrow X'$ such that $j'_2 \circ \mathcal{F}_C(f_2) = j'_1 \circ \mathcal{F}_C(f_1)$. Since \mathbf{C}'_2 has only \mathcal{F}_C -images, there are X , $j_1 : B \rightarrow X$ and $j_2 : C \rightarrow X$ such that $\mathcal{F}_C(j_1) = j'_1$, $\mathcal{F}_C(j_2) = j'_2$ and $\mathcal{F}_C(X) = X'$. Obviously, $j_1 \circ f_1 = j_2 \circ f_2$, which implies that there is a unique $k : D \rightarrow X$ such the triangles (3), (4) commute. Applying the functor \mathcal{F}_C to k , we get the morphism $\mathcal{F}_C(k)$ making triangles (5) and (6) commute as well. Let $\bar{k}' : \mathcal{F}_C(D) \rightarrow \mathcal{F}_C(X)$ be a morphism which makes the triangles (5) and (6) commute. Since \mathbf{C}'_2 has only \mathcal{F}_C -images, there is $\bar{k} : D \rightarrow X$ such that $\mathcal{F}_C(\bar{k}) = \bar{k}'$. By uniqueness of k , we have that $k = \bar{k}$ because \bar{k} makes the triangles (3) and (4) commute as well. Therefore, $\bar{k}' = \mathcal{F}_C(\bar{k}) = \mathcal{F}_C(k)$. Thus, (2) is a pushout along \mathcal{M}_2^* -morphisms in \mathbf{C}'_2 .



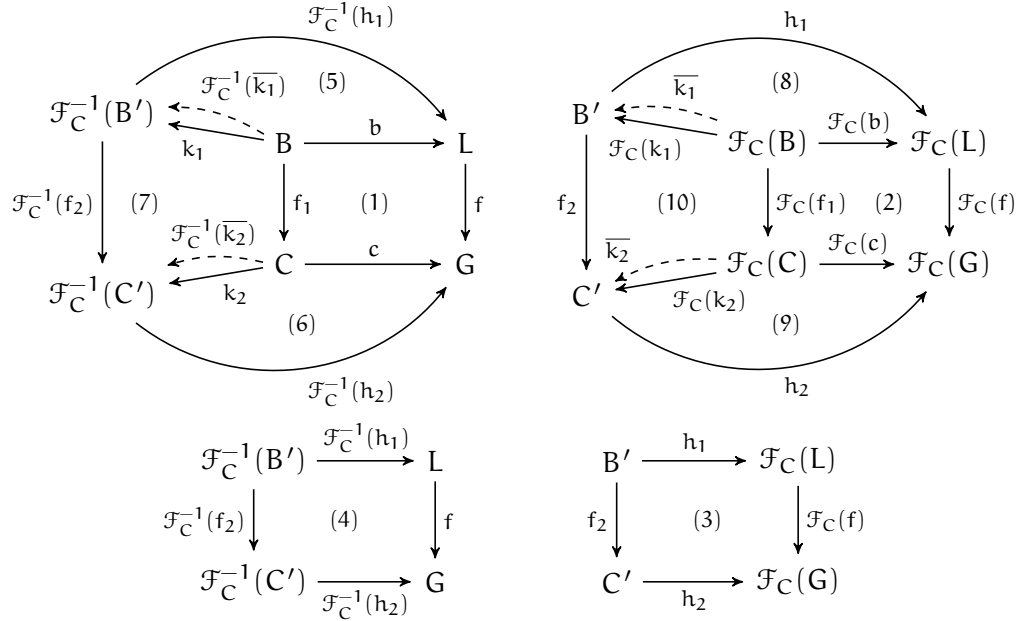
3. \mathcal{F}_C creates morphisms:

Let $f' : \mathcal{F}_C(A) \rightarrow \mathcal{F}_C(B)$ be a morphism in $\text{Mor}_{\mathbf{C}'_2}$. Since \mathbf{C}'_2 has only \mathcal{F}_C -images, there is $f : A \rightarrow B$ such that $\mathcal{F}_C(f) = f'$. Let $g : A \rightarrow B$ be another morphism such that $\mathcal{F}_C(g) = f'$. By injectiveness of \mathcal{F} and hence also of \mathcal{F}_C we get $g = f$. Thus, f is unique.

4. \mathcal{F}_C preserves initial pushouts:

Let (1) be an initial pushout and hence also a pushout in $(\mathbf{C}_1, \mathcal{M}_1)$. Since the functor \mathcal{F}_C preserves pushouts along \mathcal{M} -morphisms, we get that (2) is a pushout in \mathbf{C}'_2 . Consider now another pushout (3) in \mathbf{C}'_2 with $h_1 : B' \rightarrow \mathcal{F}_C(L)$, $h_2 : C' \rightarrow \mathcal{F}_C(G)$ in \mathcal{M}_2^* . Since $\mathcal{F}_C^{-1} : (\mathbf{C}'_2, \mathcal{M}_2^*) \xrightarrow{\sim} (\mathbf{C}_1, \mathcal{M}_1)$ is obviously a category equivalence as well and thus has the same properties as the category equivalence $\mathcal{F}_C : (\mathbf{C}_1, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{C}'_2, \mathcal{M}_2^*)$, we have that also the functor \mathcal{F}_C^{-1} preserves monomorphisms and pushouts along \mathcal{M} -morphisms, which implies that also (4) is a pushout with $\mathcal{F}_C^{-1}(h_1) : \mathcal{F}_C^{-1}(B') \rightarrow L$ and $\mathcal{F}_C^{-1}(h_2) : \mathcal{F}_C^{-1}(C') \rightarrow G$ in \mathcal{M}_1 . By assumption we know that (1) is an initial pushout, which means according to Definition 4 that there are unique morphisms $k_1 : B \rightarrow \mathcal{F}_C^{-1}(B')$, $k_2 : C \rightarrow \mathcal{F}_C^{-1}(C')$ in \mathcal{M}_1 such that triangles (5), (6) commute and (7) is a pushout in \mathbf{C}_1 . Applying now the functor \mathcal{F}_C to k_1 and k_2 , we get morphisms $\mathcal{F}_C(k_1) : \mathcal{F}_C(B) \rightarrow B'$ and $\mathcal{F}_C(k_2) : \mathcal{F}_C(C) \rightarrow C'$ in \mathcal{M}_2^* because the functor \mathcal{F}_C preserves monomorphisms, making the triangles (8) and (9) commute by general functor property. Moreover, we obtain that (10) is a pushout in \mathbf{C}'_2 because the functor \mathcal{F}_C preserves pushouts along \mathcal{M} -morphisms. It remains now to show that the morphisms $\mathcal{F}_C(k_1)$, $\mathcal{F}_C(k_2)$ are unique with properties mentioned before. For this reason assume first that there are other morphisms $\bar{k}_1 : \mathcal{F}_C(B) \rightarrow B'$, $\bar{k}_2 : \mathcal{F}_C(C) \rightarrow C'$ in \mathcal{M}_2^* such that it holds that $h_1 \circ \bar{k}_1 = \mathcal{F}_C(b)$, $h_2 \circ \bar{k}_2 = \mathcal{F}_C(c)$ and $f_2 \circ \bar{k}_1 = \bar{k}_2 \circ \mathcal{F}_C(f_1)$ is a pushout in \mathbf{C}'_2 . Applying now the functor \mathcal{F}_C^{-1} to \bar{k}_1 and \bar{k}_2 ,

we get morphisms $\mathcal{F}_C^{-1}(\overline{k_1}) : B \rightarrow \mathcal{F}_C^{-1}(B')$ and $\mathcal{F}_C^{-1}(\overline{k_2}) : C \rightarrow \mathcal{F}_C^{-1}(C')$ in \mathcal{M}_1 because the functor \mathcal{F}_C^{-1} preserves monomorphisms such that $\mathcal{F}_C^{-1}(h_1) \circ \mathcal{F}_C^{-1}(\overline{k_1}) = b$, $\mathcal{F}_C^{-1}(h_2) \circ \mathcal{F}_C^{-1}(\overline{k_2}) = c$ by general functor property and $\mathcal{F}_C^{-1}(\overline{k_2}) \circ f_1 = \mathcal{F}_C^{-1}(f_2) \circ \mathcal{F}_C^{-1}(\overline{k_1})$ is a pushout in \mathbf{C}_1 because the functor \mathcal{F}_C^{-1} preserves pushouts along \mathcal{M} -morphisms. But since (1) is an initial pushout, we have that the morphisms k_1, k_2 are unique, which directly implies that $k_1 = \mathcal{F}_C^{-1}(\overline{k_1})$ and $k_2 = \mathcal{F}_C^{-1}(\overline{k_2})$. Thus, we get that also $\mathcal{F}_C(k_1) = \overline{k_1}$ and $\mathcal{F}_C(k_2) = \overline{k_2}$. This altogether implies that (2) is an initial pushout in $(\mathbf{C}_2', \mathcal{M}_2^*)$ according to Definition 4.



5. \mathcal{F}_C **preserves epimorphisms, i.e.,** $\mathcal{F}_C(\mathcal{E}_1) \subseteq \mathcal{E}_2^*$:

Let $f : A \rightarrow B$ be an epimorphism in $\text{Mor}_{\mathbf{C}_1}$. We have to show that also $\mathcal{F}_C(f) : \mathcal{F}_C(A) \rightarrow \mathcal{F}_C(B)$ is an epimorphism in $\text{Mor}_{\mathbf{C}_2'}$. Consider morphisms $\mathcal{F}_C(g), \mathcal{F}_C(h) : \mathcal{F}_C(B) \rightarrow \mathcal{F}_C(C)$ such that $\mathcal{F}_C(g) \circ \mathcal{F}_C(f) = \mathcal{F}_C(h) \circ \mathcal{F}_C(f)$ (see the diagram below to the right). Since the functor \mathcal{F}_C creates morphisms uniquely as shown before, we have morphisms $g, h : B \rightarrow C$ with $g \circ f = h \circ f$. Moreover, since f is an epimorphism by assumption, we get that $g = h$ and thus also $\mathcal{F}_C(g) = \mathcal{F}_C(h)$, which was to be shown.

$$A \xrightarrow{f} B \xrightarrow[g]{g} C \quad \mathcal{F}_C(A) \xrightarrow{\mathcal{F}_C(f)} \mathcal{F}_C(B) \xrightarrow[\mathcal{F}_C(h)]{\mathcal{F}_C(g)} \mathcal{F}_C(C)$$

6. \mathcal{F}_C **preserves coproducts:**

Consider a family of objects $(\mathcal{F}_C(A_i))_{i \in I}$ in $\text{Ob}_{\mathbf{C}_2'}$ with an index set I . Since \mathbf{C}_2' has only \mathcal{F}_C -images, there is also a family of objects $(A_i)_{i \in I}$ in $\text{Ob}_{\mathbf{C}_1}$. Construct now a coproduct $(A, (u_i)_{i \in I})$ of $(A_i)_{i \in I}$ in \mathbf{C}_1 for $A \in \text{Ob}_{\mathbf{C}_1}$ and a family of morphisms $(u_i : A_i \rightarrow A)_{i \in I}$ in $\text{Mor}_{\mathbf{C}_1}$. Then there are also an object $\mathcal{F}_C(A) \in \text{Ob}_{\mathbf{C}_2'}$ as well as a family of morphisms $(\mathcal{F}_C(u_i) : \mathcal{F}_C(A_i) \rightarrow \mathcal{F}_C(A))_{i \in I}$ in $\text{Mor}_{\mathbf{C}_2'}$. Take now $B' \in \text{Ob}_{\mathbf{C}_2'}$ as object of comparison together with a family of morphisms $(f'_i : \mathcal{F}_C(A_i) \rightarrow B')_{i \in I}$ in $\text{Mor}_{\mathbf{C}_2'}$. Since \mathbf{C}_2' has only \mathcal{F}_C -images, we have that there are an object of comparison $B \in \text{Ob}_{\mathbf{C}_1}$ together with a family of morphisms $(f_i : A_i \rightarrow B)_{i \in I}$ in $\text{Mor}_{\mathbf{C}_1}$ such that $(f'_i)_{i \in I} = (\mathcal{F}_C(f_i))_{i \in I}$ and $B' = \mathcal{F}_C(B)$. Then by the universal property of coproducts, we have that there is a unique morphism $f : A \rightarrow B$ making the triangle (1) commute. Applying the functor \mathcal{F}_C to the triangle to the left, we get the triangle to the right commuting as well since the functor \mathcal{F}_C

preserves commuting diagrams by general functor property.

Let $\mathcal{F}_C(\bar{f}) : \mathcal{F}_C(A) \rightarrow \mathcal{F}_C(B)$ be a morphism in $\text{Mor}_{\mathcal{C}'_2}$ making the triangle (2) commute as well. Then there is also a morphism $\bar{f} : A \rightarrow B$ in $\text{Mor}_{\mathcal{C}_1}$ and it holds:

$$\mathcal{F}_C(\bar{f}) \circ \mathcal{F}_C(u_i) = \mathcal{F}_C(f_i) \Leftrightarrow \mathcal{F}_C(\bar{f} \circ u_i) = \mathcal{F}_C(f_i) \xrightarrow{\text{Lem. } 10} \bar{f} \circ u_i = f_i$$

This implies by uniqueness of f that $\bar{f} = f$ and thus also $\mathcal{F}_C(\bar{f}) = \mathcal{F}_C(f)$. Then we have that $\mathcal{F}_C(f)$ is a unique morphism making the triangle (2) commute.

$$\begin{array}{ccc} A_i & \xrightarrow{u_i} & A \\ & \searrow f_i & \downarrow f \\ & & B \end{array} \quad (1) \quad \begin{array}{ccc} \mathcal{F}_C(A_i) & \xrightarrow{\mathcal{F}_C(u_i)} & \mathcal{F}_C(A) \\ & \searrow f'_i = \mathcal{F}_C(f_i) & \downarrow \mathcal{F}_C(f) \\ & & \mathcal{F}_C(B) = B' \end{array} \quad (2)$$

7. \mathcal{F}_C creates \mathcal{M} -morphisms:

Let $f' : \mathcal{F}_C(A) \rightarrow \mathcal{F}_C(B)$ be a morphism in \mathcal{M}_2^* . From the fact that the functor \mathcal{F}_C creates morphisms, we get that there is a unique $f : A \rightarrow B$ such that $\mathcal{F}_C(f) = f'$. Furthermore, since $\mathcal{M}_2^* = \mathcal{F}_C(\mathcal{M}_1)$ by construction, we know that there is a morphism $f'' : A \rightarrow B$ from \mathcal{M}_1 such that $\mathcal{F}_C(f'') = f'$. By uniqueness we get that $f'' = f$.

8. \mathcal{F}_C preserves pullbacks of \mathcal{M} -morphisms:

Let (1) be a pullback of \mathcal{M}_1 -morphisms in \mathcal{C}_1 as given below with $g_1, g_2 \in \mathcal{M}_1$. Applying the functor \mathcal{F}_C to the diagram (1), we get the diagram (2), which obviously commutes, because functors preserve commuting diagrams by general functor property. Moreover, $\mathcal{F}_C(g_1), \mathcal{F}_C(g_2) \in \mathcal{M}_2^*$ because the functor \mathcal{F}_C preserves monomorphisms. It remains to show the universal property for the diagram (2). Let X' be a comparison object with $j'_1 : X' \rightarrow \mathcal{F}_C(B)$, $j'_2 : X' \rightarrow \mathcal{F}_C(C)$ such that $\mathcal{F}_C(g_2) \circ j'_2 = \mathcal{F}_C(f_1) \circ j'_1$. Since \mathcal{C}'_2 has only \mathcal{F}_C -images, there are $X, j_1 : X \rightarrow B$ and $j_2 : X \rightarrow C$ such that $\mathcal{F}_C(j_1) = j'_1$, $\mathcal{F}_C(j_2) = j'_2$ and $\mathcal{F}_C(X) = X'$. Obviously, $g_1 \circ j_1 = g_2 \circ j_2$, which implies that there is a unique $k : X \rightarrow A$ such that the triangles (3), (4) commute. Applying the functor \mathcal{F}_C to k , we get the morphism $\mathcal{F}_C(k)$ making triangles (5) and (6) commute as well. Let $\bar{k}' : \mathcal{F}_C(X) \rightarrow \mathcal{F}_C(A)$ be a morphism which makes the triangles (5) and (6) commute. Since \mathcal{C}'_2 has only \mathcal{F}_C -images, there is $\bar{k} : X \rightarrow A$ such that $\mathcal{F}_C(\bar{k}) = \bar{k}'$. By uniqueness of k , we have that $k = \bar{k}$ because \bar{k} makes the triangles (3) and (4) commute as well. Therefore, $\bar{k}' = \mathcal{F}_C(\bar{k}) = \mathcal{F}_C(k)$. Thus, (2) is a pullback of \mathcal{M}_2^* -morphisms in \mathcal{C}'_2 .

$$\begin{array}{ccc} & & j'_1 = \mathcal{F}_C(j_1) \\ & & \curvearrowright \\ X' = \mathcal{F}_C(X) & & \mathcal{F}_C(A) \xrightarrow{\mathcal{F}_C(f_1)} \mathcal{F}_C(B) \\ & \searrow \mathcal{F}_C(k) & \downarrow \mathcal{F}_C(g_1) \in \mathcal{M}_2^* \\ & & \mathcal{F}_C(C) \xrightarrow{\mathcal{F}_C(g_2) \in \mathcal{M}_2^*} \mathcal{F}_C(D) \end{array} \quad (6) \quad \begin{array}{ccc} X & \xrightarrow{j_1} & B \\ \downarrow \bar{k} & \searrow k & \downarrow f_1 \\ A & \xrightarrow{f_1} & B \\ \downarrow f_2 & & \downarrow g_1 \in \mathcal{M}_1 \\ C & \xrightarrow{g_2 \in \mathcal{M}_1} & D \end{array} \quad (1) \quad \begin{array}{ccc} X & \xrightarrow{j_1} & B \\ \downarrow \bar{k} & \searrow k & \downarrow f_1 \\ A & \xrightarrow{f_1} & B \\ \downarrow f_2 & & \downarrow g_1 \in \mathcal{M}_1 \\ C & \xrightarrow{g_2 \in \mathcal{M}_1} & D \end{array} \quad (3) \quad \begin{array}{ccc} X & \xrightarrow{j_1} & B \\ \downarrow \bar{k} & \searrow k & \downarrow f_1 \\ A & \xrightarrow{f_1} & B \\ \downarrow f_2 & & \downarrow g_1 \in \mathcal{M}_1 \\ C & \xrightarrow{g_2 \in \mathcal{M}_1} & D \end{array} \quad (4)$$

9. \mathcal{F}_C preserves \mathcal{E}' -instances:

According to Definition 61, we have to show the following²:

$$\forall (a'_1 : P' \rightarrow C'_1, b'_1 : C \rightarrow C'_1) \in \mathcal{E}'_1.$$

² \mathcal{E}'_2 denotes in the following formula the class of jointly epimorphic morphisms of the subcategory \mathcal{C}'_2 .

$$\begin{aligned} \exists (a'_2 : \mathcal{F}_C(P') \rightarrow C'_2, b'_2 : \mathcal{F}_C(C) \rightarrow C'_2) \in \mathcal{E}_2'' \\ a'_2 = \mathcal{F}_C(a'_1) \wedge b'_2 = \mathcal{F}_C(b'_1) \end{aligned}$$

Consider morphisms $a : P \rightarrow C$ and $b : P \rightarrow P'$ in Mor_{C_1} and let $(a'_1 : P' \rightarrow C'_1, b'_1 : C \rightarrow C'_1)$ be in \mathcal{E}_1' . Since C'_2 contains only \mathcal{F}_C -images and the functor \mathcal{F}_C is a restriction of \mathcal{F} , we have the morphisms $\mathcal{F}_C(a'_1) : \mathcal{F}_C(P') \rightarrow \mathcal{F}_C(C'_1)$ and $\mathcal{F}_C(b'_1) : \mathcal{F}_C(C) \rightarrow \mathcal{F}_C(C'_1)$ in Mor_{C_2} with $C'_2 = \mathcal{F}_C(C'_1)$, $a'_2 = \mathcal{F}_C(a'_1)$ and $b'_2 = \mathcal{F}_C(b'_1)$.

It remains to show that $(a'_2, b'_2) \in \mathcal{E}_2''$. According to Definition 71 in Appendix A, we have to show that arbitrary morphisms $h'_1, h'_2 : C'_2 \rightarrow X'$ satisfy the property that $(h'_1 \circ a'_2 = h'_2 \circ a'_2 \wedge h'_1 \circ b'_2 = h'_2 \circ b'_2) \Rightarrow (h'_1 = h'_2)$. It holds the following:

$$\begin{aligned} h'_1 \circ a'_2 &= h'_2 \circ a'_2 \wedge h'_1 \circ b'_2 = h'_2 \circ b'_2 \\ &\Rightarrow h'_1 \circ \mathcal{F}_C(a'_1) = h'_2 \circ \mathcal{F}_C(a'_1) \wedge h'_1 \circ \mathcal{F}_C(b'_1) = h'_2 \circ \mathcal{F}_C(b'_1) \\ &\stackrel{(*)}{\Rightarrow} \mathcal{F}_C(h_1) \circ \mathcal{F}_C(a'_1) = \mathcal{F}_C(h_2) \circ \mathcal{F}_C(a'_1) \\ &\wedge \mathcal{F}_C(h_1) \circ \mathcal{F}_C(b'_1) = \mathcal{F}_C(h_2) \circ \mathcal{F}_C(b'_1) \\ &\stackrel{\text{Funct. prop.}}{\Rightarrow} \mathcal{F}_C(h_1 \circ a'_1) = \mathcal{F}_C(h_2 \circ a'_1) \wedge \mathcal{F}_C(h_1 \circ b'_1) = \mathcal{F}_C(h_2 \circ b'_1) \\ &\stackrel{\text{Lem. 10} + \mathcal{F}_C \text{ restr. of } \mathcal{F}}{\Rightarrow} h_1 \circ a'_1 = h_2 \circ a'_1 \wedge h_1 \circ b'_1 = h_2 \circ b'_1 \\ &\stackrel{(a'_1, b'_1) \in \mathcal{E}_1'}{\Rightarrow} h_1 = h_2 \\ &\Rightarrow \mathcal{F}_C(h_1) = \mathcal{F}_C(h_2) \\ &\Rightarrow h'_1 = h'_2 \end{aligned}$$

(*): This step is possible since C'_2 contains only \mathcal{F}_C -images and the functor \mathcal{F}_C is a restriction of \mathcal{F} .

Thus, altogether we get that $(a'_2 = \mathcal{F}_C(a'_1), b'_2 = \mathcal{F}_C(b'_1)) \in \mathcal{E}_2''$.

$$\begin{array}{ccc} P & \xrightarrow{b} & P' \\ a \downarrow & & \downarrow a'_1 \\ C & \xrightarrow{b'_1} & C'_1 \xrightarrow[h_2]{h_1} X \end{array} \quad \begin{array}{ccc} \mathcal{F}_C(P) & \xrightarrow{\mathcal{F}_C(b)} & \mathcal{F}_C(P') \\ \mathcal{F}_C(a) \downarrow & & \downarrow a'_2 \\ \mathcal{F}_C(C) & \xrightarrow{b'_2} & C'_2 \xrightarrow[h'_2]{h'_1} X' = \mathcal{F}_C(X) \end{array}$$

10. \mathcal{F}_C creates \mathcal{E}' -instances:

According to Definition 61, we have to show the following:

$$\begin{aligned} \forall (a'_2 : \mathcal{F}_C(P') \rightarrow C'_2, b'_2 : \mathcal{F}_C(C) \rightarrow C'_2) \in \mathcal{F}_C(\mathcal{E}_1'). \quad (1) \text{ commutes} \wedge b'_2 \in \mathcal{M}_2^* \\ \Rightarrow \exists (a'_1 : P' \rightarrow C'_1, b'_1 : C \rightarrow C'_1) \in \mathcal{E}_1'. \quad a'_2 = \mathcal{F}_C(a'_1) \wedge b'_2 = \mathcal{F}_C(b'_1) \\ \wedge (2) \text{ commutes} \wedge b'_1 \in \mathcal{M}_1 \end{aligned}$$

Consider morphisms $a : P \rightarrow C$ and $b : P \rightarrow P'$ in Mor_{C_1} and let $(a'_2 : \mathcal{F}_C(P') \rightarrow C'_2, b'_2 : \mathcal{F}_C(C) \rightarrow C'_2)$ be in $\mathcal{F}_C(\mathcal{E}_1')$, (1) commute and $b'_2 \in \mathcal{M}_2^*$. Since C'_2 contains only \mathcal{F}_C -images and the functor \mathcal{F}_C is a restriction of \mathcal{F} , we have that $a'_2 = \mathcal{F}_C(a'_1)$, $b'_2 = \mathcal{F}_C(b'_1)$ and $C'_2 = \mathcal{F}_C(C'_1)$ for morphisms $a'_1 : P' \rightarrow C'_1$ and $b'_1 : C \rightarrow C'_1$ in Mor_{C_1} . Since $\mathcal{M}_2^* = \mathcal{F}(\mathcal{M}_1)$ and $b'_2 = \mathcal{F}_C(b'_1) = \mathcal{F}(b'_1)$, we also have that $b'_1 \in \mathcal{M}_1$. Furthermore, we have that (2) commutes since:

$$a'_2 \circ \mathcal{F}_C(b) = b'_2 \circ \mathcal{F}_C(a)$$

$$\begin{aligned}
&\Rightarrow \mathcal{F}_C(a'_1) \circ \mathcal{F}_C(b) = \mathcal{F}_C(b'_1) \circ \mathcal{F}_C(a) \\
&\stackrel{\text{Funct. prop.}}{\Rightarrow} \mathcal{F}_C(a'_1 \circ b) = \mathcal{F}_C(b'_1 \circ a) \\
&\stackrel{\text{Lem. 10} + \mathcal{F}_C \text{ restr. of } \mathcal{F}}{\Rightarrow} a'_1 \circ b = b'_1 \circ a
\end{aligned}$$

It remains to show that $(a'_1, b'_1) \in \mathcal{E}'_1$. According to Definition 71 in Appendix A, we have to show that arbitrary morphisms $h_1, h_2 : C'_1 \rightarrow X$ satisfy the property that $(h_1 \circ a'_1 = h_2 \circ a'_1 \wedge h_1 \circ b'_1 = h_2 \circ b'_1) \Rightarrow (h_1 = h_2)$. It holds the following:

$$\begin{aligned}
&h_1 \circ a'_1 = h_2 \circ a'_1 \wedge h_1 \circ b'_1 = h_2 \circ b'_1 \\
&\Rightarrow \mathcal{F}_C(h_1 \circ a'_1) = \mathcal{F}_C(h_2 \circ a'_1) \wedge \mathcal{F}_C(h_1 \circ b'_1) = \mathcal{F}_C(h_2 \circ b'_1) \\
&\stackrel{\text{Funct. prop.}}{\Rightarrow} \mathcal{F}_C(h_1) \circ \mathcal{F}_C(a'_1) = \mathcal{F}_C(h_2) \circ \mathcal{F}_C(a'_1) \\
&\wedge \mathcal{F}_C(h_1) \circ \mathcal{F}_C(b'_1) = \mathcal{F}_C(h_2) \circ \mathcal{F}_C(b'_1) \\
&\stackrel{(a'_1, b'_1) \in \mathcal{F}_C(\mathcal{E}'_1)}{\Rightarrow} \mathcal{F}_C(h_1) = \mathcal{F}_C(h_2) \\
&\stackrel{\text{Lem. 10} + \mathcal{F}_C \text{ restr. of } \mathcal{F}}{\Rightarrow} h_1 = h_2
\end{aligned}$$

Thus, altogether we get that $(a'_1, b'_1) \in \mathcal{E}'_1$.

$$\begin{array}{ccc}
\mathcal{F}_C(P) & \xrightarrow{\mathcal{F}_C(b)} & \mathcal{F}_C(P') \\
\mathcal{F}_C(a) \downarrow & (1) & \downarrow a'_2 \\
\mathcal{F}_C(C) & \xrightarrow{b'_2} & C'_2 \xrightarrow[\mathcal{F}_C(h_2)]{\mathcal{F}_C(h_1)} \mathcal{F}_C(X)
\end{array}
\quad
\begin{array}{ccc}
P & \xrightarrow{b} & P' \\
a \downarrow & (2) & \downarrow a'_1 \\
C & \xrightarrow{b'_1} & C'_1 \xrightarrow[h_2]{h_1} X
\end{array}$$

□

Lemma 59: ($I \circ \mathcal{F}_C$ Satisfies Required Properties, see page 215)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (C_1, \mathcal{M}_1, P)$, $AS_2 = (C_2, \mathcal{M}_2, \mathcal{F}(P))$ such that the underlying \mathcal{M} -adhesive categories (C_i, \mathcal{M}_i) have \mathcal{E}'_i - \mathcal{M}_i pair factorization for $i \in \{1, 2\}$ and (C_1, \mathcal{M}_1) has initial pushouts, a functor $\mathcal{F} : (C_1, \mathcal{M}_1) \rightarrow (C_2, \mathcal{M}_2)$, an inclusion functor $I : (C'_2, \mathcal{M}_2^*) \rightarrow (C_2, \mathcal{M}_2)$, and functors $\mathcal{F}_C : (C_1, \mathcal{M}_1) \rightarrow (C'_2, \mathcal{M}_2^*)$, $\mathcal{F}_C^{-1} : (C'_2, \mathcal{M}_2^*) \rightarrow (C_1, \mathcal{M}_1)$ building a category equivalence $\mathcal{F}_C : (C_1, \mathcal{M}_1) \xrightarrow{\sim} (C'_2, \mathcal{M}_2^*)$ where (C'_2, \mathcal{M}_2^*) is the subcategory of (C_2, \mathcal{M}_2) , in which all non- \mathcal{F} -images have been removed. Then the functor composition $I \circ \mathcal{F}_C : (C_1, \mathcal{M}_1) \rightarrow (C_2, \mathcal{M}_2)$ satisfies the properties listed in Definition 69 if the functors I and \mathcal{F}_C satisfy these properties.

Proof.

1. $I \circ \mathcal{F}_C$ preserves monomorphisms, i.e., $(I \circ \mathcal{F}_C)(\mathcal{M}_1) \subseteq \mathcal{M}_2$:

$$(I \circ \mathcal{F}_C)(\mathcal{M}_1) \stackrel{\text{Def. I}}{=} \mathcal{F}_C(\mathcal{M}_1) \stackrel{(*)}{\subseteq} \mathcal{M}_2^* \subseteq \mathcal{M}_2$$

The step $(*)$ holds, because the functor \mathcal{F}_C preserves monomorphisms by assumption.

2. $I \circ \mathcal{F}_C$ preserves pushouts along \mathcal{M} -morphisms:

Consider a pushout (1) in C_1 with an \mathcal{M}_1 -morphism f_1 . Since the functor \mathcal{F}_C preserves monomorphisms and pushouts along \mathcal{M} -morphisms by assumption, we have that also the diagram (2) is a pushout in C'_2 with an \mathcal{M}_2^* -morphism $\mathcal{F}_C(f_1)$. Since I preserves monomorphisms and pushouts along \mathcal{M} -morphisms by assumption, we have that also the diagram (3) is a pushout in C_2 with an \mathcal{M}_2 -morphism $I(\mathcal{F}_C(f_1)) = \mathcal{F}_C(f_1)$.

$$\begin{array}{ccccc}
A & \xrightarrow{f_1 \in \mathcal{M}_1} & B & \mathcal{F}_C(A) & \xrightarrow{\mathcal{F}_C(f_1) \in \mathcal{M}_2^*} & \mathcal{F}_C(B) & I(\mathcal{F}_C(A)) & \xrightarrow{I(\mathcal{F}_C(f_1)) \in \mathcal{M}_2} & I(\mathcal{F}_C(B)) \\
f_2 \downarrow & (1) & g_1 \downarrow & \mathcal{F}_C(f_2) \downarrow & (2) & \mathcal{F}_C(g_1) \downarrow & I(\mathcal{F}_C(f_2)) \downarrow & (3) & I(\mathcal{F}_C(g_1)) \downarrow \\
C & \xrightarrow{g_2} & D & \mathcal{F}_C(C) & \xrightarrow{\mathcal{F}_C(g_2)} & \mathcal{F}_C(D) & I(\mathcal{F}_C(C)) & \xrightarrow{I(\mathcal{F}_C(g_2))} & I(\mathcal{F}_C(D))
\end{array}$$

3. $I \circ \mathcal{F}_C$ **creates morphisms:**

Since the functor \mathcal{F}_C creates morphisms by assumption, we have that $\forall f' : \mathcal{F}_C(A) \rightarrow \mathcal{F}_C(B)$. $\exists! f : A \rightarrow B$. $\mathcal{F}_C(f) = f'$. Applying now the definition of I , we obtain that $\forall f' : (I \circ \mathcal{F}_C)(A) \rightarrow (I \circ \mathcal{F}_C)(B)$. $\exists! f : A \rightarrow B$. $(I \circ \mathcal{F}_C)(f) = f'$.

4. $I \circ \mathcal{F}_C$ **preserves initial pushouts:**

Consider an initial pushout (1) over a morphism $f : L \rightarrow G$ in $(\mathbf{C}_1, \mathcal{M}_1)$ with boundary object B , context object C and \mathcal{M}_1 -morphisms $b : B \rightarrow L$, $c : C \rightarrow G$. Since the functor \mathcal{F}_C preserves initial pushouts by assumption, we have that also the diagram (2) is an initial pushout over the morphism $\mathcal{F}_C(f) : \mathcal{F}_C(L) \rightarrow \mathcal{F}_C(G)$ in $(\mathbf{C}'_2, \mathcal{M}_2^*)$ with boundary object $\mathcal{F}_C(B)$, context object $\mathcal{F}_C(C)$ and \mathcal{M}_2^* -morphisms $\mathcal{F}_C(b) : \mathcal{F}_C(B) \rightarrow \mathcal{F}_C(L)$, $\mathcal{F}_C(c) : \mathcal{F}_C(C) \rightarrow \mathcal{F}_C(G)$. Since I preserves initial pushouts by assumption, we have that also the diagram (3) is an initial pushout over the morphism $I(\mathcal{F}_C(f)) : I(\mathcal{F}_C(L)) \rightarrow I(\mathcal{F}_C(G))$ in $(\mathbf{C}_2, \mathcal{M}_2)$ with boundary object $I(\mathcal{F}_C(B))$, context object $I(\mathcal{F}_C(C))$ and \mathcal{M}_2 -morphisms $I(\mathcal{F}_C(b)) : I(\mathcal{F}_C(B)) \rightarrow I(\mathcal{F}_C(L))$, $I(\mathcal{F}_C(c)) : I(\mathcal{F}_C(C)) \rightarrow I(\mathcal{F}_C(G))$.

$$\begin{array}{ccccc}
B & \xrightarrow{b \in \mathcal{M}_1} & L & \mathcal{F}_C(B) & \xrightarrow{\mathcal{F}_C(b) \in \mathcal{M}_2^*} & \mathcal{F}_C(L) & I(\mathcal{F}_C(B)) & \xrightarrow{I(\mathcal{F}_C(b)) \in \mathcal{M}_2} & I(\mathcal{F}_C(L)) \\
f_1 \downarrow & (1) & f \downarrow & \mathcal{F}_C(f_1) \downarrow & (2) & \mathcal{F}_C(f) \downarrow & I(\mathcal{F}_C(f_1)) \downarrow & (3) & I(\mathcal{F}_C(f)) \downarrow \\
C & \xrightarrow{c \in \mathcal{M}_1} & G & \mathcal{F}_C(C) & \xrightarrow{\mathcal{F}_C(c) \in \mathcal{M}_2^*} & \mathcal{F}_C(G) & I(\mathcal{F}_C(C)) & \xrightarrow{I(\mathcal{F}_C(c)) \in \mathcal{M}_2} & I(\mathcal{F}_C(G))
\end{array}$$

5. $I \circ \mathcal{F}_C$ **preserves epimorphisms, i.e., $(I \circ \mathcal{F}_C)(\mathcal{E}_1) \subseteq \mathcal{E}_2$:**

$$(I \circ \mathcal{F}_C)(\mathcal{E}_1) \stackrel{\text{Def. } I}{=} I(\mathcal{F}_C(\mathcal{E}_1)) \stackrel{(*)}{\subseteq} \mathcal{E}_2^* \subseteq \mathcal{E}_2$$

The step (*) holds, because the functor \mathcal{F}_C preserves epimorphisms by assumption.

6. $I \circ \mathcal{F}_C$ **preserves coproducts:**

Consider a coproduct $(A, (u_i)_{i \in I})$ of a family of objects $(A_i)_{i \in I}$ in \mathbf{C}_1 for $A \in \text{Ob}_{\mathbf{C}_1}$ and a family of morphisms $(u_i : A_i \rightarrow A)_{i \in I}$ in $\text{Mor}_{\mathbf{C}_1}$ with an index set I . Since the functor \mathcal{F}_C preserves coproducts by assumption, we have that also $(\mathcal{F}_C(A), (\mathcal{F}_C(u_i))_{i \in I})$ is a coproduct of the family of translated objects $(\mathcal{F}_C(A_i))_{i \in I}$ in \mathbf{C}'_2 for $\mathcal{F}_C(A) \in \text{Ob}_{\mathbf{C}'_2}$ and the family of translated morphisms $(\mathcal{F}_C(u_i) : \mathcal{F}_C(A_i) \rightarrow \mathcal{F}_C(A))_{i \in I}$ in $\text{Mor}_{\mathbf{C}'_2}$. Since I preserves coproducts by assumption, we have that also $(I(\mathcal{F}_C(A)), (I(\mathcal{F}_C(u_i)))_{i \in I})$ is a coproduct of the family of translated objects $(I(\mathcal{F}_C(A_i)))_{i \in I}$ in \mathbf{C}_2 for $I(\mathcal{F}_C(A)) \in \text{Ob}_{\mathbf{C}_2}$ and the family of translated morphisms $(I(\mathcal{F}_C(u_i)) : I(\mathcal{F}_C(A_i)) \rightarrow I(\mathcal{F}_C(A)))_{i \in I}$ in $\text{Mor}_{\mathbf{C}_2}$.

$$A_i \xrightarrow{u_i} A \quad \mathcal{F}_C(A_i) \xrightarrow{\mathcal{F}_C(u_i)} \mathcal{F}_C(A) \quad I(\mathcal{F}_C(A_i)) \xrightarrow{I(\mathcal{F}_C(u_i))} I(\mathcal{F}_C(A))$$

7. $I \circ \mathcal{F}_C$ **creates \mathcal{M} -morphisms:**

Since the functor \mathcal{F}_C creates \mathcal{M} -morphisms by assumption, it holds that $\forall f' : \mathcal{F}_C(A) \rightarrow \mathcal{F}_C(B)$. $f' \in \mathcal{M}_2^* \Rightarrow (\exists! f \in \mathcal{M}_1. \mathcal{F}_C(f) = f')$. Applying now the definition of I , we obtain that $\forall f' : (I \circ \mathcal{F}_C)(A) \rightarrow (I \circ \mathcal{F}_C)(B)$. $f' \in \mathcal{M}_2 \Rightarrow (\exists! f \in \mathcal{M}_1. (I \circ \mathcal{F}_C)(f) = f')$

8. $I \circ \mathcal{F}_C$ **preserves pullbacks of \mathcal{M} -morphisms:**

Consider a pullback (1) in \mathbf{C}_1 with \mathcal{M}_1 -morphisms g_1 and g_2 . Since the functor \mathcal{F}_C preserves monomorphisms and pullbacks of \mathcal{M} -morphisms by assumption, we have that

also the diagram (2) is a pullback in \mathbf{C}'_2 with \mathcal{M}_2^* -morphisms $\mathcal{F}_C(g_1)$ and $\mathcal{F}_C(g_2)$. Since I preserves monomorphisms and pullbacks of \mathcal{M} -morphisms by assumption, we have that also the diagram (3) is a pullback in \mathbf{C}_2 with \mathcal{M}_2 -morphisms $I(\mathcal{F}_C(g_1)) = \mathcal{F}_C(g_1)$ and $I(\mathcal{F}_C(g_2)) = \mathcal{F}_C(g_2)$.

$$\begin{array}{ccccc}
 A & \xrightarrow{f_1} & B & \xrightarrow{\mathcal{F}_C(f_1)} & \mathcal{F}_C(B) & \xrightarrow{I(\mathcal{F}_C(f_1))} & I(\mathcal{F}_C(B)) \\
 f_2 \downarrow & (1) & \downarrow g_1 \in \mathcal{M}_1 & \downarrow \mathcal{F}_C(f_2) & \downarrow \mathcal{F}_C(g_1) \in \mathcal{M}_2^* & \downarrow I(\mathcal{F}_C(f_2)) & \downarrow I(\mathcal{F}_C(g_1)) \in \mathcal{M}_2 \\
 C & \xrightarrow{g_2 \in \mathcal{M}_1} & D & \xrightarrow{\mathcal{F}_C(g_2) \in \mathcal{M}_2^*} & \mathcal{F}_C(D) & \xrightarrow{I(\mathcal{F}_C(g_2)) \in \mathcal{M}_2} & I(\mathcal{F}_C(D))
 \end{array}$$

(2) (3)

9. $I \circ \mathcal{F}_C$ preserves \mathcal{E}' -instances:

According to Definition 61, we have to show the following:

$$\begin{aligned}
 & \forall (a'_1 : P' \rightarrow C'_1, b'_1 : C \rightarrow C'_1) \in \mathcal{E}'_1. \\
 & \exists (a'_3 : (I \circ \mathcal{F}_C)(P') \rightarrow C'_3, b'_3 : (I \circ \mathcal{F}_C)(C) \rightarrow C'_3) \in \mathcal{E}'_2. \\
 & a'_3 = (I \circ \mathcal{F}_C)(a'_1) \wedge b'_3 = (I \circ \mathcal{F}_C)(b'_1)
 \end{aligned}$$

Consider morphisms $a : P \rightarrow C$ and $b : P \rightarrow P'$ in $\text{Mor}_{\mathbf{C}_1}$ and fix $a'_1 : P' \rightarrow C'_1$, $b'_1 : C \rightarrow C'_1$ such that $(a'_1, b'_1) \in \mathcal{E}'_1$. Since the functor \mathcal{F}_C preserves \mathcal{E}' -instances by assumption, it holds that

$$\exists (a'_2 : \mathcal{F}_C(P') \rightarrow C'_2, b'_2 : \mathcal{F}_C(C) \rightarrow C'_2) \in \mathcal{E}'_2. \quad a'_2 = \mathcal{F}_C(a'_1) \wedge b'_2 = \mathcal{F}_C(b'_1)$$

Since I preserves \mathcal{E}' -instances by assumption as well, we have that

$$\begin{aligned}
 & \exists (a'_3 : I(\mathcal{F}_C(P')) \rightarrow C'_3, b'_3 : I(\mathcal{F}_C(C)) \rightarrow C'_3) \in \mathcal{E}'_2. \\
 & a'_3 = I(a'_2) = I(\mathcal{F}_C(a'_1)) = (I \circ \mathcal{F}_C)(a'_1) \\
 & \wedge b'_3 = I(b'_2) = I(\mathcal{F}_C(b'_1)) = (I \circ \mathcal{F}_C)(b'_1)
 \end{aligned}$$

$$\begin{array}{ccccc}
 P & \xrightarrow{b} & P' & \xrightarrow{\mathcal{F}_C(b)} & \mathcal{F}_C(P') & \xrightarrow{I(\mathcal{F}_C(b))} & I(\mathcal{F}_C(P')) \\
 a \downarrow & & \downarrow a'_1 & \downarrow \mathcal{F}_C(a) & \downarrow a'_2 & \downarrow I(\mathcal{F}_C(a)) & \downarrow a'_3 \\
 C & \xrightarrow{b'_1} & C'_1 & \xrightarrow{b'_2} & C'_2 & \xrightarrow{b'_3} & C'_3
 \end{array}$$

10. $I \circ \mathcal{F}_C$ creates \mathcal{E}' -instances:

According to Definition 61, we have to show the following:

$$\begin{aligned}
 & \forall (a'_3 : (I \circ \mathcal{F}_C)(P') \rightarrow C'_3, b'_3 : (I \circ \mathcal{F}_C)(C) \rightarrow C'_3) \in (I \circ \mathcal{F}_C)(\mathcal{E}'_1). \\
 & (1) \text{ commutes} \wedge b'_3 \in \mathcal{M}_2 \\
 & \Rightarrow \exists (a'_1 : P' \rightarrow C'_1, b'_1 : C \rightarrow C'_1) \in \mathcal{E}'_1. \quad a'_3 = (I \circ \mathcal{F}_C)(a'_1) \wedge b'_3 = (I \circ \mathcal{F}_C)(b'_1) \\
 & \wedge (3) \text{ commutes} \wedge b'_1 \in \mathcal{M}_1
 \end{aligned}$$

Consider morphisms $a : P \rightarrow C$ and $b : P \rightarrow P'$ in $\text{Mor}_{\mathbf{C}_1}$ and let $(a'_3 : (I \circ \mathcal{F}_C)(P') \rightarrow C'_3, b'_3 : (I \circ \mathcal{F}_C)(C) \rightarrow C'_3)$ be in $(I \circ \mathcal{F}_C)(\mathcal{E}'_1)$, (1) commute and $b'_3 \in \mathcal{M}_2$. Since I creates \mathcal{E}' -instances by assumption, it holds that

$$\begin{aligned}
 & \exists (a'_2 : \mathcal{F}_C(P') \rightarrow C'_2, b'_2 : \mathcal{F}_C(C) \rightarrow C'_2) \in \mathcal{F}_C(\mathcal{E}'_1). \\
 & a'_3 = I(a'_2) \wedge b'_3 = I(b'_2) \wedge (2) \text{ commutes} \wedge b'_2 \in \mathcal{M}_2^*
 \end{aligned}$$

Since the functor \mathcal{F}_C creates \mathcal{E}' -instances by assumption as well and the premise of the corresponding statement for the subcategory $(\mathbf{C}'_2, \mathcal{M}_2^*)$ is fulfilled, we have that

$$\begin{aligned} \exists (a'_1 : P' \rightarrow C'_1, b'_1 : C \rightarrow C'_1) \in \mathcal{E}'_1. a'_2 = \mathcal{F}_C(a'_1) \wedge b'_2 = \mathcal{F}_C(b'_1) \\ \wedge (3) \text{ commutes} \wedge b'_1 \in \mathcal{M}_1 \end{aligned}$$

Furthermore, it holds that $a'_3 = I(a'_2) = I(\mathcal{F}_C(a'_1)) = (I \circ \mathcal{F}_C)(a'_1)$ and $b'_3 = I(b'_2) = I(\mathcal{F}_C(b'_1)) = (I \circ \mathcal{F}_C)(b'_1)$.

$$\begin{array}{ccccc} (I \circ \mathcal{F}_C)(P) & \xrightarrow{I(\mathcal{F}_C(b))} & (I \circ \mathcal{F}_C)(P') & \mathcal{F}_C(P) & \xrightarrow{\mathcal{F}_C(b)} & \mathcal{F}_C(P') & P & \xrightarrow{b} & P' \\ I(\mathcal{F}_C(a)) \downarrow & (1) & a'_3 \downarrow & \mathcal{F}_C(a) \downarrow & (2) & a'_2 \downarrow & a \downarrow & (3) & a'_1 \downarrow \\ (I \circ \mathcal{F}_C)(C) & \xrightarrow{b'_3} & C'_3 & \mathcal{F}_C(C) & \xrightarrow{b'_2} & C'_2 & C & \xrightarrow{b'_1} & C'_1 \end{array}$$

□

Lemma 60: (\mathcal{F}_{RC} Satisfies Required Properties, see page 218)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}_R(P))$ such that $(\mathbf{C}_1, \mathcal{M}_1)$ has \mathcal{E}'_1 - \mathcal{M}_1 pair factorization and initial pushouts, a restricted functor $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$, and restricted functors $\mathcal{F}_{RC} : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}'_2|_{\mathcal{M}_2^*}$, $\mathcal{F}_{RC}^{-1} : \mathbf{C}'_2|_{\mathcal{M}_2^*} \rightarrow \mathbf{C}_1|_{\mathcal{M}_1}$ building a category equivalence $\mathcal{F}_{RC} : \mathbf{C}_1|_{\mathcal{M}_1} \xrightarrow{\sim} \mathbf{C}'_2|_{\mathcal{M}_2^*}$ such that $\mathbf{C}'_2|_{\mathcal{M}_2^*}$ is the subcategory of $\mathbf{C}_2|_{\mathcal{M}_2}$, in which all non- \mathcal{F}_R -images have been removed, and where the restricted functor \mathcal{F}_{RC} is the restriction of \mathcal{F}_R . Then the restricted functor \mathcal{F}_{RC} satisfies the properties listed in Definition 70.

Proof.

1. \mathcal{F}_{RC} **preserves monomorphisms, i.e.,** $\mathcal{F}_{RC}(\mathcal{M}_1) \subseteq \mathcal{M}_2^*$:
The proof works analogously to Case 1 from the proof of Lemma 58.
2. \mathcal{F}_{RC} **preserves pushouts of \mathcal{M} -morphisms:**
The proof works analogously to Case 2 from the proof of Lemma 58.
3. \mathcal{F}_{RC} **creates \mathcal{M} -morphisms:**
The proof works analogously to Case 7 from the proof of Lemma 58.
4. \mathcal{F}_{RC} **preserves initial pushouts over \mathcal{M} -morphisms:**
The proof works analogously to Case 4 from the proof of Lemma 58.
5. \mathcal{F}_{RC} **preserves epimorphisms, i.e.,** $\mathcal{F}_{RC}(\mathcal{E}_1) \subseteq \mathcal{E}_2^*$:
The proof works analogously to Case 5 from the proof of Lemma 58.
6. \mathcal{F}_{RC} **preserves coproducts of \mathcal{M} -morphisms:**
The proof works analogously to Case 6 from the proof of Lemma 58.
7. \mathcal{F}_{RC} **preserves pullbacks of \mathcal{M} -morphisms:**
The proof works analogously to Case 8 from the proof of Lemma 58.
8. \mathcal{F}_{RC} **preserves \mathcal{E}' -instances:**
The proof works analogously to Case 9 from the proof of Lemma 58.
9. \mathcal{F}_{RC} **creates \mathcal{E}' -instances:**
The proof works analogously to Case 10 from the proof of Lemma 58.

□

Lemma 61: ($I_R \circ \mathcal{F}_{RC}$ Satisfies Required Properties, see page 218)

Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{C}_1, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{C}_2, \mathcal{M}_2, \mathcal{F}_R(P))$ such that the underlying \mathcal{M} -adhesive categories $(\mathbf{C}_i, \mathcal{M}_i)$ have \mathcal{E}'_i - \mathcal{M}_i pair factorization for $i \in \{1, 2\}$ and $(\mathbf{C}_1, \mathcal{M}_1)$ has initial pushouts, a restricted functor $\mathcal{F}_R : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$, a restricted inclusion functor $I_R : \mathbf{C}'_2|_{\mathcal{M}_2^*} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$, and restricted functors $\mathcal{F}_{RC} : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}'_2|_{\mathcal{M}_2^*}$, $\mathcal{F}_{RC}^{-1} : \mathbf{C}'_2|_{\mathcal{M}_2^*} \rightarrow \mathbf{C}_1|_{\mathcal{M}_1}$ building a category equivalence $\mathcal{F}_{RC} : \mathbf{C}_1|_{\mathcal{M}_1} \xrightarrow{\sim} \mathbf{C}'_2|_{\mathcal{M}_2^*}$

such that $\mathbf{C}'_2|_{\mathcal{M}_2^*}$ is the subcategory of $\mathbf{C}_2|_{\mathcal{M}_2}$, in which all non- \mathcal{F}_R -images have been removed. Then the functor composition $I_R \circ \mathcal{F}_{RC} : \mathbf{C}_1|_{\mathcal{M}_1} \rightarrow \mathbf{C}_2|_{\mathcal{M}_2}$ satisfies the properties listed in Definition 70 if the restricted functors I_R and \mathcal{F}_{RC} satisfy these properties.

Proof.

1. $I_R \circ \mathcal{F}_{RC}$ **preserves monomorphisms, i.e.,** $(I_R \circ \mathcal{F}_{RC})(\mathcal{M}_1) \subseteq \mathcal{M}_2$:
The proof works analogously to Case 1 from the proof of Lemma 59.
2. $I_R \circ \mathcal{F}_{RC}$ **preserves pushouts of \mathcal{M} -morphisms:**
The proof works analogously to Case 2 from the proof of Lemma 59.
3. $I_R \circ \mathcal{F}_{RC}$ **creates \mathcal{M} -morphisms:**
The proof works analogously to Case 7 from the proof of Lemma 59.
4. $I_R \circ \mathcal{F}_{RC}$ **preserves initial pushouts over \mathcal{M} -morphisms:**
The proof works analogously to Case 4 from the proof of Lemma 59.
5. $I_R \circ \mathcal{F}_{RC}$ **preserves epimorphisms, i.e.,** $(I_R \circ \mathcal{F}_{RC})(\mathcal{E}_1) \subseteq \mathcal{E}_2$:
The proof works analogously to Case 5 from the proof of Lemma 59.
6. $I_R \circ \mathcal{F}_{RC}$ **preserves coproducts of \mathcal{M} -morphisms:**
The proof works analogously to Case 6 from the proof of Lemma 59.
7. $I_R \circ \mathcal{F}_{RC}$ **preserves pullbacks of \mathcal{M} -morphisms:**
The proof works analogously to Case 8 from the proof of Lemma 59.
8. $I_R \circ \mathcal{F}_{RC}$ **preserves \mathcal{E}' -instances:**
The proof works analogously to Case 9 from the proof of Lemma 59.
9. $I_R \circ \mathcal{F}_{RC}$ **creates \mathcal{E}' -instances:**
The proof works analogously to Case 10 from the proof of Lemma 59.

□

DETAILED PROOFS FOR HYPERGRAPH TRANSFORMATION SYSTEMS

In this appendix, we give the detailed proofs for indicated lemmas from the main part of this work concerning the hypergraph application.

Lemma 2: (Characterization of Hypergraph Morphisms, see page 49)

1. Consider a hypergraph morphism $f = (f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H)$ according to Definition 34. Then the following two properties hold:
 - a) The number of incoming resp. outgoing arrows for each edge remains equal, i.e.,
 - i. $\forall e \in E_G, \forall n \in \mathbb{N}. (|s_G(e)|=n) \Leftrightarrow (|s_H(f_E(e))|=n),$
 - ii. $\forall e \in E_G, \forall n \in \mathbb{N}. (|t_G(e)|=n) \Leftrightarrow (|t_H(f_E(e))|=n)$ and
 - b) The morphism preserves/reflects the source and target components of every edge, i.e.,
 - i. $\forall v \in V_G, \forall e \in E_G, \forall n \leq |s_G(e)|. (s_G^n(e)=v) \Rightarrow (s_H^n(f_E(e))=f_V(v)),$
 - ii. $\forall v \in V_G, \forall e \in E_G, \forall n \leq |s_G(e)|.$
 $(s_G^n(e)=v) \Leftarrow (s_H^n(f_E(e))=f_V(v))$ if f_V is injective,
 - iii. $\forall v \in V_G, \forall e \in E_G, \forall n \leq |t_G(e)|. (t_G^n(e)=v) \Rightarrow (t_H^n(f_E(e))=f_V(v)),$
 - iv. $\forall v \in V_G, \forall e \in E_G, \forall n \leq |t_G(e)|.$
 $(t_G^n(e)=v) \Leftarrow (t_H^n(f_E(e))=f_V(v))$ if f_V is injective.
2. According to Definition 34, $f = (f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H)$ is a hypergraph morphism if the properties 1(b)i and 1(b)iii hold.

Proof.

1. Assume that $f = (f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H)$ is a hypergraph morphism, i.e., $f_V^*(s_G(e)) = s_H(f_E(e)).$

To show: Properties 1(a)i-1(a)ii and 1(b)i-1(b)iv.

Property 1(a)i:

- a) (\Rightarrow) : Given $e \in E_G$ and $n \in \mathbb{N}$ with $|s_G(e)| = n$. Then we have:

$$\begin{aligned} |s_H(f_E(e))| &= |f_V^*(s_G(e))| = |s_G(e)| = n \\ \Rightarrow |s_H(f_E(e))| &= n \end{aligned}$$

- b) (\Leftarrow) : Given $e \in E_G$ and $n \in \mathbb{N}$ with $|s_H(f_E(e))| = n$. Then we have:

$$\begin{aligned} |s_G(e)| &= |f_V^*(s_G(e))| = |s_H(f_E(e))| = n \\ \Rightarrow |s_G(e)| &= n \end{aligned}$$

Property 1(a)ii: Similar to the proof of Property 1(a)i replacing s by t .

Property 1(b)i: Given $v \in V_G$ and $e \in E_G$ with $s_G^n(e) = v = v_n$, where $s_G(e) = v_1 \dots v_n \dots v_m$, $n < |s_G(e)|$ and $m = |s_G(e)|$. Then we have:

$$\begin{aligned} s_H(f_E(e)) &= f_V^*(s_G(e)) = f_V(v_1) \dots f_V(v_n) \dots f_V(v_m) \\ \Rightarrow s_H^n(f_E(e)) &= f_V(v_n) = f_V(v) \end{aligned}$$

Property **1(b)ii**: Given $v \in V_G$ and $e \in E_G$ with $s_H^n(f_E(e)) = f_V(v)$ and $n \leq |s_G(e)|$. Furthermore, let f_V be injective. Then we have:

$$f_V(s_G^n(e)) = [f_V^*(s_G(e))]_n = [s_H(f_E(e))]_n = s_H^n(f_E(e)) = f_V(v)$$

$$\stackrel{f_V \text{ inj.}}{\Rightarrow} s_G^n(e) = v, \text{ where } [\]_n \text{ denotes } n\text{-th component of } [\]$$

Property **1(b)iii**: Similar to the proof of Property **1(b)i** replacing s by t .

Property **1(b)iv**: Similar to the proof of Property **1(b)ii** replacing s by t .

2. Assume that Properties **1(b)i** and **1(b)iii** hold.

To show: $f = (f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H)$ is a hypergraph morphism, i.e.,

$$a) f_V^*(s_G(e)) = s_H(f_E(e)) \text{ and}$$

$$b) f_V^*(t_G(e)) = t_H(f_E(e)).$$

Part **2a**:

$$f_V^*(s_G(e)) = s_H(f_E(e))$$

$$\Leftrightarrow \forall n \leq |s_G(e)|. [f_V^*(s_G(e))]_n = [s_H(f_E(e))]_n$$

$$\Leftrightarrow \forall n \leq |s_G(e)|. f_V(s_G^n(e)) = s_H^n(f_E(e))$$

$$\text{Let } s_G^n(e) = v \stackrel{\text{1(b)i}}{\Rightarrow} s_H^n(f_E(e)) = f_V(v)$$

$$\Rightarrow \forall n \leq |s_G(e)|. f_V(s_G^n(e)) = f_V(v) = s_H^n(f_E(e))$$

$$\Rightarrow f_V^*(s_G(e)) = s_H(f_E(e))$$

Part **2b**: Similar to Part **2a** replacing s by t and using Property **1(b)iii**. □

Lemma 3: (Pushout in HyperGraphs, see page 50)

Consider the pushout construction from Definition 36. Then the diagram given below is a hypergraph pushout.

$$\begin{array}{ccc} A & \xrightarrow{b} & B \\ c \downarrow & (1) & \downarrow g \\ C & \xrightarrow{f} & D \end{array}$$

Proof.

Consider hypergraph morphisms $b = (b_V, b_E)$, $c = (c_V, c_E)$, $f = (f_V, f_E)$ and $g = (g_V, g_E)$. We have to show that the diagram (1) above commutes and that the universal property for pushouts holds for (1).

1. Commutation:

We have to show that $g_i \circ b_i = f_i \circ c_i$ for $i \in \{V, E\}$.

$$a) g_V \circ b_V = f_V \circ c_V :$$

Let x be an arbitrary hypergraph node in V_A . Then we have the following:

$$(g_V \circ b_V)(x) = g_V(b_V(x)) \stackrel{\text{Def. } g_V}{=} [b_V(x)]_{\equiv_V}$$

$$\stackrel{(*)}{=} [c_V(x)]_{\equiv_V} \stackrel{\text{Def. } f_V}{=} f_V(c_V(x)) = (f_V \circ c_V)(x)$$

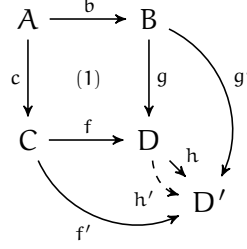
The step $(*)$ holds, because by definition of \equiv_V (see Definition 36) we have that $((b_V(x), c_V(x)) \in \equiv_V) \Rightarrow ([b_V(x)]_{\equiv_V} = [c_V(x)]_{\equiv_V})$.

b) $g_E \circ b_E = f_E \circ c_E :$

The proof for this case works analogously to the case above.

2. **Universal property for pushouts:**

Let $D' = (V_{D'}, E_{D'}, s_{D'}, t_{D'})$ be an object of comparison and $g' : B \rightarrow D'$, $f' : C \rightarrow D'$ with $g' = (g'_V, g'_E)$, $f' = (f'_V, f'_E)$ be hypergraph morphisms such that holds $g' \circ b = f' \circ c$. We have to show: $\exists! h : D \rightarrow D'$. $h \circ f = f' \wedge h \circ g = g'$.



a) **Existence:**

Due to the pushout construction in **Sets** for each component, implying that $h_V \circ f_V = f'_V$, $h_E \circ f_E = f'_E$, $h_V \circ g_V = g'_V$, $h_E \circ g_E = g'_E$, we have that $h = (h_V, h_E)$ exists. It remains to show that h is a hypergraph morphism, i.e., $s_{D'} \circ h_E = h_V^* \circ s_D$ and $t_{D'} \circ h_E = h_V^* \circ t_D$.

$$\begin{array}{ccc} E_D & \xrightarrow{s_D} & V_D^* \\ h_E \downarrow & & \downarrow h_V^* \\ E_{D'} & \xrightarrow{s_{D'}} & V_{D'}^* \end{array}$$

i. $s_{D'} \circ h_E = h_V^* \circ s_D :$

Fix $e \in E_D$. Then by definition of E_D we have that $\exists e' \in (E_B \cap e)$. $e = g_E(e')$ or $\exists e' \in (E_C \cap e)$. $e = f_E(e')$.

- Let $e' \in (E_B \cap e)$ such that holds: $e = g_E(e')$:

We have the following:

$$\begin{aligned} (s_{D'} \circ h_E)(e) &= s_{D'}(h_E(e)) \stackrel{e=g_E(e')}{=} s_{D'}(h_E(g_E(e'))) \\ &= s_{D'}((h_E \circ g_E)(e')) \stackrel{h_E \circ g_E = g'_E}{=} s_{D'}(g'_E(e')) \\ &\stackrel{g' \in \text{Mor}_{\text{HyperGraphs}}}{=} g_V^*(s_B(e')) \stackrel{g'_V = h_V \circ g_V}{=} (h_V \circ g_V)^*(s_B(e')) \\ &= h_V^*(g_V^*(s_B(e'))) \stackrel{g \in \text{Mor}_{\text{HyperGraphs}}}{=} h_V^*(s_D(g_E(e'))) \\ &\stackrel{g_E(e')=e}{=} h_V^*(s_D(e)) = (h_V^* \circ s_D)(e) \end{aligned}$$

$$\begin{array}{ccc} E_B & \xrightarrow{s_B} & V_B^* \\ g'_E \downarrow & & \downarrow g_V^* \\ E_{D'} & \xrightarrow{s_{D'}} & V_{D'}^* \end{array} \quad \begin{array}{ccc} E_B & \xrightarrow{s_B} & V_B^* \\ g_E \downarrow & & \downarrow g_V^* \\ E_D & \xrightarrow{s_D} & V_D^* \end{array}$$

- Let $e' \in (E_C \cap e)$ such that holds: $e = f_E(e')$:

The proof for this case works analogously to the case above.

ii. $t_{D'} \circ h_E = h_V^* \circ t_D :$

The proof for this case works analogously to the case above.

b) **Uniqueness:**

Let $h' : D \rightarrow D'$ be another morphism with $h' = (h'_V, h'_E)$, $h'_V \circ f_V = f'_V$, $h'_E \circ f_E = f'_E$, $h'_V \circ g_V = g'_V$, $h'_E \circ g_E = g'_E$. This implies, using the universal property for the two pushouts in **Sets** for the V- and E-components, that $h'_V = h_V$, $h'_E = h_E$ and hence $h' = h$.

□

Lemma 4: (Pullback in HyperGraphs, see page 52)

Consider the pullback construction from Definition 37. Then the diagram given below is a hypergraph pullback.

$$\begin{array}{ccc} A & \xrightarrow{b} & B \\ c \downarrow & (1) & \downarrow g \\ C & \xrightarrow{f} & D \end{array}$$

Proof.

Consider hypergraph morphisms $b = (b_V, b_E)$, $c = (c_V, c_E)$, $f = (f_V, f_E)$ and $g = (g_V, g_E)$. We have to show that the diagram (1) above commutes and that the universal property for pullbacks holds for (1).

1. **Commutation:**

We have to show that $g_i \circ b_i = f_i \circ c_i$ for $i \in \{V, E\}$.

a) $g_V \circ b_V = f_V \circ c_V$:

Let (x, y) be an arbitrary hypergraph node in V_A . Then we have the following:

$$\begin{aligned} (g_V \circ b_V)(x, y) &= g_V(b_V(x, y)) \stackrel{\text{Def. } b_V}{=} g_V(x) \\ &\stackrel{\text{Def. } V_A}{=} f_V(y) \stackrel{\text{Def. } c_V}{=} f_V(c_V(x, y)) = (f_V \circ c_V)(x, y) \end{aligned}$$

b) $g_E \circ b_E = f_E \circ c_E$:

The proof for this case works analogously to the case above.

2. **Universal property for pullbacks:**

Let $A' = (V_{A'}, E_{A'}, s_{A'}, t_{A'})$ be an object of comparison and $b' : A' \rightarrow B$, $c' : A' \rightarrow C$ with $b' = (b'_V, b'_E)$, $c' = (c'_V, c'_E)$ be hypergraph morphisms such that holds $g \circ b' = f \circ c'$.

We have to show: $\exists! h : A' \rightarrow A$. $b \circ h = b' \wedge c \circ h = c'$.

$$\begin{array}{ccccc} & & & b' & \\ & & & \curvearrowright & \\ A' & & & & B \\ & \searrow h & & & \downarrow g \\ & A & \xrightarrow{b} & & \\ & \downarrow c & (1) & & \\ & C & \xrightarrow{f} & & D \\ & \nearrow c' & & & \end{array}$$

a) **Existence:**

Due to the pullback construction in **Sets** for each component, implying that $b_V \circ h_V = b'_V$, $b_E \circ h_E = b'_E$, $c_V \circ h_V = c'_V$, $c_E \circ h_E = c'_E$, we have that $h = (h_V, h_E)$ exists. It remains to show that h is a hypergraph morphism, i.e., $s_A \circ h_E = h_V^* \circ s_{A'}$ and $t_A \circ h_E = h_V^* \circ t_{A'}$.

$$\begin{array}{ccc}
E_{A'} & \xrightarrow[t_{A'}]{s_{A'}} & V_{A'}^* \\
h_E \downarrow & & \downarrow h_V^* \\
E_A & \xrightarrow[t_A]{s_A} & V_A^*
\end{array}$$

i. $s_A \circ h_E = h_V^* \circ s_{A'} :$

Fix $e \in E_{A'}$. Then we have the following:

$$\begin{aligned}
(s_A \circ h_E)(e) &= (h_V^* \circ s_{A'})(e) \\
&\Leftarrow s_A(h_E(e)) = h_V^*(s_{A'}(e)) \\
&\stackrel{\text{Def. } b_V, c_V}{\Leftarrow} b_V^*(s_A(h_E(e))) = b_V^*(h_V^*(s_{A'}(e))) \\
&\wedge c_V^*(s_A(h_E(e))) = c_V^*(h_V^*(s_{A'}(e))) \\
&\stackrel{b, c \in \text{Mor}^{\text{HyperGraphs}}}{\Leftarrow} s_B(b_E(h_E(e))) = b_V^*(h_V^*(s_{A'}(e))) \\
&\wedge s_C(c_E(h_E(e))) = c_V^*(h_V^*(s_{A'}(e))) \\
&\Leftarrow s_B((b_E \circ h_E)(e)) = (b_V \circ h_V)^*(s_{A'}(e)) \\
&\wedge s_C((c_E \circ h_E)(e)) = (c_V \circ h_V)^*(s_{A'}(e)) \\
&\stackrel{b_E \circ h_E = b'_E}{\Leftarrow} s_B(b'_E(e)) = (b_V \circ h_V)^*(s_{A'}(e)) \\
&\wedge s_C((c_E \circ h_E)(e)) = (c_V \circ h_V)^*(s_{A'}(e)) \\
&\stackrel{c_E \circ h_E = c'_E}{\Leftarrow} s_B(b'_E(e)) = (b_V \circ h_V)^*(s_{A'}(e)) \\
&\wedge s_C(c'_E(e)) = (c_V \circ h_V)^*(s_{A'}(e)) \\
&\stackrel{b_V \circ h_V = b'_V}{\Leftarrow} s_B(b'_E(e)) = b_V'^*(s_{A'}(e)) \\
&\wedge s_C(c'_E(e)) = (c_V \circ h_V)^*(s_{A'}(e)) \\
&\stackrel{c_V \circ h_V = c'_V}{\Leftarrow} s_B(b'_E(e)) = b_V'^*(s_{A'}(e)) \\
&\wedge s_C(c'_E(e)) = c_V'^*(s_{A'}(e)) \\
&\stackrel{b', c' \in \text{Mor}^{\text{HyperGraphs}}}{\Leftarrow} s_B(b'_E(e)) = s_B(b'_E(e)) \\
&\wedge s_C(c'_E(e)) = s_C(c'_E(e))
\end{aligned}$$

$$\begin{array}{ccc}
E_A & \xrightarrow[t_A]{s_A} & V_A^* \\
b_E \downarrow & = & \downarrow b_V^* \\
E_B & \xrightarrow[t_B]{s_B} & V_B^*
\end{array}
\quad
\begin{array}{ccc}
E_A & \xrightarrow[t_A]{s_A} & V_A^* \\
c_E \downarrow & = & \downarrow c_V^* \\
E_C & \xrightarrow[t_C]{s_C} & V_C^*
\end{array}$$

$$\begin{array}{ccc}
E_{A'} & \xrightarrow[t_{A'}]{s_{A'}} & V_{A'}^* \\
b'_E \downarrow & = & \downarrow b_V'^* \\
E_B & \xrightarrow[t_B]{s_B} & V_B^*
\end{array}
\quad
\begin{array}{ccc}
E_{A'} & \xrightarrow[t_{A'}]{s_{A'}} & V_{A'}^* \\
c'_E \downarrow & = & \downarrow c_V'^* \\
E_C & \xrightarrow[t_C]{s_C} & V_C^*
\end{array}$$

ii. $t_A \circ h_E = h_V^* \circ t_{A'} :$

The proof for this case works analogously to the case above.

b) **Uniqueness:**

Let $h' : A' \rightarrow A$ be another morphism with $h' = (h'_V, h'_E)$, $b_V \circ h'_V = b'_V$, $b_E \circ$

$h'_E = b'_E$, $c_V \circ h'_V = c'_V$, $c_E \circ h'_E = c'_E$. This implies, using the universal property for the two pullbacks in **Sets** for the V- and E-components, that $h'_V = h_V$, $h'_E = h_E$ and hence $h' = h$.

□

Lemma 5: (Boundary Object in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, see page 53)

Consider two hypergraphs $L = (V_L, E_L, s_L, t_L)$ and $G = (V_G, E_G, s_G, t_G)$. The boundary object $B \subseteq L$ of the initial pushout over a general morphism $f : L \rightarrow G$ in the \mathcal{M} -adhesive category $(\mathbf{HyperGraphs}, \mathcal{M})$ can be constructed as follows with an inclusion $b : B \rightarrow L$:

$B = (V_B, E_B, s_B, t_B)$ where

$V_B = DP_V \cup IP_V \cup IP_{VE}$ with dangling points

$$DP_V = \{v \in V_L \mid \exists e \in E_G \setminus f_E(E_L). (f_V(v) \in s_G(e)) \vee (f_V(v) \in t_G(e))\}$$

where $x \in w \Leftrightarrow \exists w_1, w_2. w = w_1 \cdot x \cdot w_2$ and identification points

$$IP_V = \{v \in V_L \mid \exists v' \neq v. v' \in V_L \wedge f_V(v) = f_V(v')\},$$

$$IP_{VE} = \{\bar{v} \in V_L \mid \exists e \in IP_E. \bar{v} \in s_L(e) \vee \bar{v} \in t_L(e)\},$$

$$E_B = IP_E = \{e \in E_L \mid \exists e' \neq e. e' \in E_L \wedge f_E(e) = f_E(e')\},$$

$$s_B(e) = s_L(e),$$

$$t_B(e) = t_L(e).$$

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ & & \downarrow f \\ & & G \end{array}$$

Proof.

1. To show: $b : B \rightarrow L$ is an inclusion.

$b : B \rightarrow L$ is an inclusion, because functions s_B and t_B are restrictions of the respective functions s_L and t_L .

2. To show: $s_B, t_B : E_B \rightarrow V_B^*$ are well-defined.

Consider $e \in E_B$ with $v \in s_B(e)$. Then we have that $v \in s_L(e)$, because b_V is an inclusion and hence $v \in V_L$. Then there is $e \in IP_E$ with $v \in s_L(e)$, which implies that $v \in IP_{VE} \subseteq V_B$.

The proof for t_B works analogously replacing s by t .

□

Lemma 6: (Context Object in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, see page 54)

Consider a hypergraph morphism $f : L \rightarrow G$ and the boundary object B constructed according to Lemma 5 above. Then the context object C can be constructed in the \mathcal{M} -adhesive category $(\mathbf{HyperGraphs}, \mathcal{M})$ as follows with inclusion $c : C \rightarrow G$:

$C = (V_C, E_C, s_C, t_C)$ with

$$V_C = (V_G \setminus f_V(V_L)) \cup f_V(b_V(V_B)),$$

$$E_C = (E_G \setminus f_E(E_L)) \cup f_E(b_E(E_B)),$$

$$s_C(e) = s_G(e),$$

$$t_C(e) = t_G(e).$$

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ \downarrow & & \downarrow f \\ C & \xrightarrow{c} & G \end{array}$$

Proof.

1. To show: $c : C \rightarrow G$ is an inclusion.

$c : C \rightarrow G$ is an inclusion, because functions s_C and t_C are restrictions of the respective functions s_G and t_G .

2. To show: $s_C, t_C : E_C \rightarrow V_C^*$ are well-defined.

a) Let $e \in E_C$.

To show: $s_C(e) \in V_C^*$.

$$\begin{array}{ccc} E_C & \xrightarrow{s_C} & V_C^* \\ c_E \downarrow & = & \downarrow c_V^* \\ E_G & \xrightarrow{s_G} & V_G^* \end{array}$$

Consider some $v \in s_C(e)$. According to definition of s_C we get that $v \in s_G(e)$ and hence also $v \in V_G$.

It remains to show: $v \in V_C$.

Case 1: Let $e \in E_G \setminus (f_E(E_L))$: (1)

Case 1.1: Let $v \in f_V(V_L) \Rightarrow \exists v' \in V_L. f_V(v') = v$: (2)

$$\begin{aligned} & v \in s_G(e) \\ & \stackrel{(1), (2)}{\Rightarrow} \exists e \in E_G \setminus (f_E(E_L)). f_V(v') \in s_G(e) \\ & \stackrel{\text{Def. } DP_V}{\Rightarrow} v' \in DP_V \text{ (see Lemma 5)} \\ & \stackrel{\text{Def. } V_B}{\Rightarrow} v' \in V_B \\ & \stackrel{b_V \text{ incl.}}{\Rightarrow} v' \in b_V(V_B) \\ & \stackrel{(2)}{\Rightarrow} v \in f_V(b_V(V_B)) \\ & \stackrel{\text{Def. } V_C}{\Rightarrow} v \in V_C \end{aligned}$$

Case 1.2: Let $v \notin f_V(V_L) \Rightarrow v \in V_G \setminus (f_V(V_L)) \stackrel{\text{Def. } V_C}{\Rightarrow} v \in V_C$

Case 2: Let $e \in f_E(b_E(E_B)) \Rightarrow \exists e' \in E_B. e = f_E(b_E(e')) \stackrel{b_E \text{ incl.}}{=} f_E(e')$: (3)

$$\begin{aligned} & f_V^* \circ s_L = s_G \circ f_E \\ & \Rightarrow f_V^*(s_L(e')) = s_G(f_E(e')) \\ & \stackrel{(3)}{\Rightarrow} f_V^*(s_L(e')) = s_G(e) \\ & \stackrel{v \in s_G(e)}{\Rightarrow} v \in f_V^*(s_L(e')) \\ & \Rightarrow \exists x \in s_L(e'). v = f_V(x) \\ & \stackrel{(*)}{\Rightarrow} x \in V_B \wedge v = f_V(x) \\ & \stackrel{b_V \text{ incl.}}{\Rightarrow} x \in V_B \wedge v = f_V(b_V(x)) \\ & \Rightarrow v \in f_V(b_V(V_B)) \\ & \stackrel{\text{Def. } V_C}{\Rightarrow} v \in V_C \end{aligned}$$

(*):

$$\begin{aligned} s_L(e') &\stackrel{b_E \text{ incl.}}{=} s_L(b_E(e')) \stackrel{(D1) \text{ comm.}}{=} b_V^*(s_B(e')) \\ &\stackrel{b_V \text{ incl.}}{=} s_B(e') \in V_B^* \end{aligned}$$

$$\begin{array}{ccc} E_L & \xrightarrow{s_L} & V_L^* \\ f_E \downarrow & = & \downarrow f_V^* \\ E_G & \xrightarrow{s_G} & V_G^* \end{array} \quad \begin{array}{ccc} E_B & \xrightarrow{s_B} & V_B^* \\ b_E \downarrow & (D1) & \downarrow b_V^* \\ E_L & \xrightarrow{s_L} & V_L^* \end{array}$$

b) The proof for t_C works analogously replacing s by t .

□

Lemma 7: (Initial Pushout in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, see page 54)

Consider a hypergraph morphism $f : L \rightarrow G$, a boundary object B constructed according to Lemma 5, a context object C constructed according to Lemma 6, and inclusions $b : B \rightarrow L$, $c : C \rightarrow G$. Then the diagram (1) given below is an initial pushout in $(\mathbf{HyperGraphs}, \mathcal{M})$ with the hypergraph morphism $g : B \rightarrow C$ defined as $g = f|_B$.

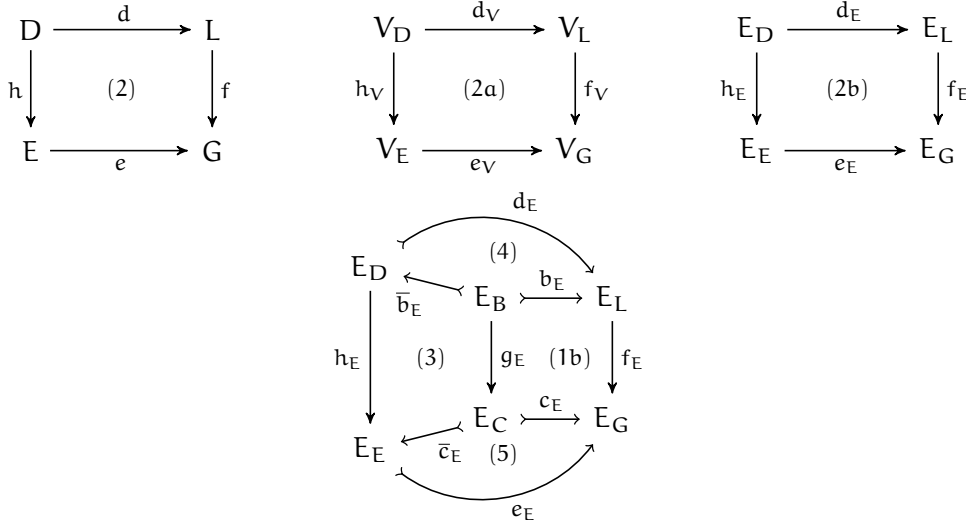
$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ g \downarrow & (1) & \downarrow f \\ C & \xrightarrow{c} & G \end{array}$$

Proof.

In this proof, we show first that (1) is a pushout in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$. According to Definition 36 and Lemma 3, we know that pushouts in $\mathbf{HyperGraphs}$ are constructed componentwise. Then using Lemma A.6 from [224], which concerns the construction of pushout complements in **Sets**, we get that the diagrams (1a) and (1b) are pushouts in **Sets**, because V_C and E_C are pushout complements in **Sets**. This implies directly that also (1) is a pushout in $\mathbf{HyperGraphs}$.

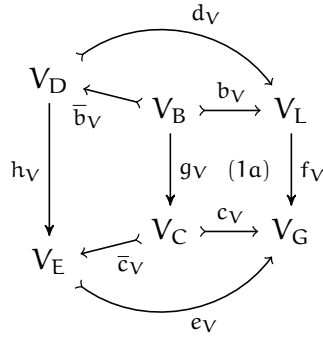
$$\begin{array}{ccc} V_B & \xrightarrow{b_V} & V_L \\ g_V \downarrow & (1a) & \downarrow f_V \\ V_C & \xrightarrow{c_V} & V_G \end{array} \quad \begin{array}{ccc} E_B & \xrightarrow{b_E} & E_L \\ g_E \downarrow & (1b) & \downarrow f_E \\ E_C & \xrightarrow{c_E} & E_G \end{array}$$

In the next step we show that the diagram (1b) is an initial pushout in **Sets**. For this reason assume first that the diagram (2) given below to the left is also a pushout with $d : D \rightarrow L$, $e : E \rightarrow G$ in \mathcal{M}_1 . Since pushouts in $\mathbf{HyperGraphs}$ are constructed componentwise, we have that the diagrams (2a) and (2b) are pushouts in **Sets**. The components E_B and E_C in the diagram (1b) are exactly the boundary over $f_E : E_L \rightarrow E_G$ in **Sets** and the context in **Sets**, respectively (see Definition A.5 and Fact A.6 in [224]). So we get that the pushout (1b) is initial, which implies using the fact that (2b) is a pushout that there are unique morphisms $\bar{b}_E : E_B \rightarrow E_D$, $\bar{c}_E : E_C \rightarrow E_E$ in \mathcal{M}_1 such that the diagram (3) is a pushout and the triangles (4), (5) commute.



Now it remains to show that also the diagram (1a) is an initial pushout in **(HyperGraphs, \mathcal{M}_1)**. For this reason, we define the function $\bar{c}_V : V_C \rightarrow V_E$ given in the picture below with $\bar{c}_V(x) = y$ and $e_V(y) = c_V(x)$ for $x \in V_C$ and $y \in V_E$.

We have to show first that \bar{c}_V given in this way is well-defined, i.e., for every $x \in V_C$ there is a unique $y \in V_E$ such that $e_V(y) = c_V(x)$.



Let $x \in V_C$. As we already know the diagram (2a) is a pushout in **Sets**. So we have the following case distinction for the existence of $y \in V_E$:

1. $x \in \mathbf{V}_G \setminus \mathbf{f}_V(\mathbf{V}_L)$

Since (2a) is a pushout, the functions $f_V : V_L \rightarrow V_G$ and $e_V : V_E \rightarrow V_G$ are jointly surjective. So $x \notin f_V(V_L)$ implies that $x \in e_V(V_E)$ and hence there is $y \in V_E$ with $e_V(y) = x \stackrel{c_V \text{ incl.}}{=} c_V(x)$.

2. $x \in \mathbf{f}_V(\mathbf{b}_V(\mathbf{V}_B))$

In this case, we know that there is $z \in V_B$ with $f_V(b_V(z)) = x \stackrel{c_V \text{ incl.}}{=} c_V(x)$ for $V_B = \mathbf{DP}_V \cup \mathbf{IP}_V \cup \mathbf{IP}_{V_E}$ according to Lemma 5.

a) $z \in \mathbf{DP}_V$

- i. $z \in \mathbf{V}_L, e \in \mathbf{E}_G \setminus \mathbf{f}_E(\mathbf{E}_L)$ s.t. $\mathbf{f}_V(z) \in \mathbf{s}_G(e)$

Since (2b) is a pushout, the functions $f_E : E_L \rightarrow E_G$ and $e_E : E_E \rightarrow E_G$ are jointly surjective. Since $e \notin f_E(E_L)$ by assumption, we know that $e \in e_E(E_E)$. Thus, there is $e' \in E_E$ such that $e_E(e') = e$. Then it holds the following:

$$c_V(x) \stackrel{c_V \text{ incl.}}{=} x \stackrel{\text{Asm.}}{=} f_V(b_V(z)) \stackrel{b_V \text{ incl.}}{=} f_V(z) \stackrel{(*)}{=} (s_G(e))_i$$

$$e =_{\mathbf{E}} e' \quad (s_{\mathbf{G}}(e_{\mathbf{E}}(e')))_i \stackrel{e \in \text{Mor}_{\text{HyperGraphs}}}{=} (e_{\mathbf{V}}^*(s_{\mathbf{E}}(e')))_i = e_{\mathbf{V}}((s_{\mathbf{E}}(e'))_i)$$

(*): Where $f_{\mathbf{V}}(z)$ occurs at position i in $s_{\mathbf{G}}(e)$ according to assumption that $f_{\mathbf{V}}(z) \in s_{\mathbf{G}}(e)$.

$$\begin{array}{ccc} \mathbf{E}_{\mathbf{E}} & \xrightarrow{s_{\mathbf{E}}} & \mathbf{V}_{\mathbf{E}}^* \\ e_{\mathbf{E}} \downarrow & \begin{array}{c} t_{\mathbf{E}} \\ = \\ \end{array} & \downarrow e_{\mathbf{V}}^* \\ \mathbf{E}_{\mathbf{G}} & \xrightarrow{s_{\mathbf{G}}} & \mathbf{V}_{\mathbf{G}}^* \\ & t_{\mathbf{G}} & \end{array}$$

Thus, we have that there is $y = (s_{\mathbf{E}}(e'))_i$ in $\mathbf{V}_{\mathbf{E}}$ such that $e_{\mathbf{V}}(y) = e_{\mathbf{V}}((s_{\mathbf{E}}(e'))_i) = c_{\mathbf{V}}(x)$, which was to be shown.

ii. $\mathbf{z} \in \mathbf{V}_{\mathbf{L}}, \mathbf{e} \in \mathbf{E}_{\mathbf{G}} \setminus \mathbf{f}_{\mathbf{E}}(\mathbf{E}_{\mathbf{L}})$ s.t. $\mathbf{f}_{\mathbf{V}}(\mathbf{z}) \in t_{\mathbf{G}}(\mathbf{e})$

The proof for this case works analogously to the proof in the case above.

b) $\mathbf{z} \in \mathbf{IP}_{\mathbf{V}}$

This means according to Lemma 5 that there is $z' \in \mathbf{V}_{\mathbf{L}}$ such that $z' \neq z$ and $f_{\mathbf{V}}(z) = f_{\mathbf{V}}(z')$. Since the diagram (2a) is a pushout in **Sets**, we have that z, z' are $d_{\mathbf{V}}$ -images because of the pushout construction in **Sets**. Thus, there is $y' \in \mathbf{V}_{\mathbf{D}}$ such that $d_{\mathbf{V}}(y') = z$ implying that $f_{\mathbf{V}}(d_{\mathbf{V}}(y')) = f_{\mathbf{V}}(z)$. Then it holds the following:

$$\begin{aligned} c_{\mathbf{V}}(x) &\stackrel{c_{\mathbf{V}} \text{ incl.}}{=} x \stackrel{x = f_{\mathbf{V}}(b_{\mathbf{V}}(z))}{=} f_{\mathbf{V}}(b_{\mathbf{V}}(z)) \stackrel{b_{\mathbf{V}} \text{ incl.}}{=} f_{\mathbf{V}}(z) \\ f_{\mathbf{V}}(z) &\stackrel{f_{\mathbf{V}}(z) = f_{\mathbf{V}}(d_{\mathbf{V}}(y'))}{=} f_{\mathbf{V}}(d_{\mathbf{V}}(y')) \stackrel{(2a) \text{ comm.}}{=} e_{\mathbf{V}}(h_{\mathbf{V}}(y')) \end{aligned}$$

Thus, we have that there is $y = h_{\mathbf{V}}(y')$ in $\mathbf{V}_{\mathbf{E}}$ such that $e_{\mathbf{V}}(y) = e_{\mathbf{V}}(h_{\mathbf{V}}(y')) = c_{\mathbf{V}}(x)$, which was to be shown.

c) $\mathbf{z} \in \mathbf{IP}_{\mathbf{V}_{\mathbf{E}}}$

i. $\mathbf{e} \in \mathbf{IP}_{\mathbf{E}}$ s.t. $\mathbf{z} \in s_{\mathbf{L}}(\mathbf{e}) \wedge (s_{\mathbf{L}}(\mathbf{e}))_i = \mathbf{z}$

By the characterization of gluing condition in **Sets** (see Fact A.7 in [224]), we have that morphisms $d_{\mathbf{E}} : \mathbf{E}_{\mathbf{D}} \rightarrow \mathbf{E}_{\mathbf{L}}, f_{\mathbf{E}} : \mathbf{E}_{\mathbf{L}} \rightarrow \mathbf{E}_{\mathbf{G}}$ satisfy the gluing condition since (1b) is an initial pushout and there is a morphism $\bar{b}_{\mathbf{E}} : \mathbf{E}_{\mathbf{B}} \rightarrow \mathbf{E}_{\mathbf{D}}$ making the triangle (4) commute as already shown before. This implies that $e \in d_{\mathbf{E}}(\mathbf{E}_{\mathbf{D}})$, i.e., there is $e' \in \mathbf{E}_{\mathbf{D}}$ such that $d_{\mathbf{E}}(e') = e$. Then it holds the following:

$$\begin{aligned} c_{\mathbf{V}}(x) &\stackrel{c_{\mathbf{V}} \text{ incl.}}{=} x \stackrel{x = f_{\mathbf{V}}(b_{\mathbf{V}}(z))}{=} f_{\mathbf{V}}(b_{\mathbf{V}}(z)) \stackrel{b_{\mathbf{V}} \text{ incl.}}{=} f_{\mathbf{V}}(z) \\ z &\stackrel{z = (s_{\mathbf{L}}(e))_i}{=} f_{\mathbf{V}}((s_{\mathbf{L}}(e))_i) \stackrel{e = d_{\mathbf{E}}(e')}{=} f_{\mathbf{V}}((s_{\mathbf{L}}(d_{\mathbf{E}}(e')))_i) \\ &\stackrel{d \in \text{Mor}_{\text{HyperGraphs}}}{=} f_{\mathbf{V}}((d_{\mathbf{V}}^*(s_{\mathbf{D}}(e'))_i) = f_{\mathbf{V}}(d_{\mathbf{V}}((s_{\mathbf{D}}(e'))_i)) \\ &\stackrel{(2a) \text{ comm.}}{=} e_{\mathbf{V}}(h_{\mathbf{V}}((s_{\mathbf{D}}(e'))_i)) \end{aligned}$$

$$\begin{array}{ccc} \mathbf{E}_{\mathbf{D}} & \xrightarrow{s_{\mathbf{D}}} & \mathbf{V}_{\mathbf{D}}^* \\ d_{\mathbf{E}} \downarrow & \begin{array}{c} t_{\mathbf{D}} \\ = \\ \end{array} & \downarrow d_{\mathbf{V}}^* \\ \mathbf{E}_{\mathbf{L}} & \xrightarrow{s_{\mathbf{L}}} & \mathbf{V}_{\mathbf{L}}^* \\ & t_{\mathbf{L}} & \end{array}$$

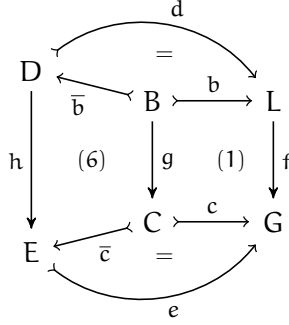
Thus, we have that there is $y = h_{\mathbf{V}}((s_{\mathbf{D}}(e'))_i)$ in $\mathbf{V}_{\mathbf{E}}$ such that $e_{\mathbf{V}}(y) = e_{\mathbf{V}}(h_{\mathbf{V}}((s_{\mathbf{D}}(e'))_i)) = c_{\mathbf{V}}(x)$, which was to be shown.

ii. $\mathbf{e} \in \mathbf{IP}_E$ s.t. $\mathbf{z} \in \mathbf{t}_L(\mathbf{e}) \wedge (\mathbf{t}_L(\mathbf{e}))_i = \mathbf{z}$

The proof for this case works analogously to the proof in the case above.

It remains now to show the uniqueness of the existing $\mathbf{y} \in V_E$ for which holds that $e_V(\mathbf{y}) = c_V(\mathbf{x})$ where $\mathbf{x} \in V_C$. Let us first assume that \mathbf{y} is not unique, i.e., there is $\bar{\mathbf{y}} \in V_E$ with $e_V(\bar{\mathbf{y}}) = c_V(\mathbf{x})$ and $\bar{\mathbf{y}} \neq \mathbf{y}$. Since we already know that $\mathbf{e} \in \mathcal{M}_1$ which means that e_V is injective, we directly get the contradiction $\bar{\mathbf{y}} = \mathbf{y}$ implying altogether that \bar{c}_V is well-defined.

After the functions $\bar{c}_V : V_C \rightarrow V_E$, $\bar{c}_E : E_C \rightarrow E_E$ are given, we now define the morphism $\bar{c} : C \rightarrow E$ by $\bar{c} = (\bar{c}_V, \bar{c}_E)$.



It remains to show that \bar{c} defined in this way is a well-defined hypergraph morphism, i.e., \bar{c} is compatible with source and target functions.

1. $\forall \mathbf{e} \in E_C. \bar{c}_V^*(s_C(\mathbf{e})) = s_E(\bar{c}_E(\mathbf{e}))$

Fix $\mathbf{e} \in E_C$. As we already know, the statements $\bar{c}_V(\mathbf{x}) = \mathbf{y}$ and $c_V(\mathbf{x}) = e_V(\mathbf{y})$ are equivalent for $\mathbf{x} \in V_C$ and $\mathbf{y} \in V_E$. Therefore, obviously, $\bar{c}_V^*(\mathbf{x}) = \mathbf{y}$ and $c_V^*(\mathbf{x}) = e_V^*(\mathbf{y})$ are equivalent as well for $\mathbf{x} \in V_C^*$ and $\mathbf{y} \in V_E^*$. Let $\mathbf{x} = s_C(\mathbf{e})$ and $\mathbf{y} = s_E(\bar{c}_E(\mathbf{e}))$. To get that $\bar{c}_V^*(\mathbf{x}) = \bar{c}_V^*(s_C(\mathbf{e})) = s_E(\bar{c}_E(\mathbf{e})) = \mathbf{y}$ it suffices to show that $c_V^*(\mathbf{x}) = c_V^*(s_C(\mathbf{e})) = e_V^*(s_E(\bar{c}_E(\mathbf{e}))) = e_V^*(\mathbf{y})$. This holds from the following:

c is a hypergraph morphism

$$\Rightarrow c_V^*(s_C(\mathbf{e})) = s_G(c_E(\mathbf{e}))$$

$$\stackrel{(5) \text{ comm.}}{\Rightarrow} c_V^*(s_C(\mathbf{e})) = s_G(e_E(\bar{c}_E(\mathbf{e})))$$

$$\stackrel{\mathbf{e} \in \text{Mor}^{\text{HyperGraphs}}}{\Rightarrow} c_V^*(s_C(\mathbf{e})) = e_V^*(s_E(\bar{c}_E(\mathbf{e})))$$

$$\begin{array}{ccc} E_C \xrightarrow{s_C} V_C^* & E_C \xrightarrow{s_C} V_C^* & E_E \xrightarrow{s_E} V_E^* \\ \bar{c}_E \downarrow \quad \quad \downarrow \bar{c}_V^* & c_E \downarrow \quad \quad \downarrow c_V^* & e_E \downarrow \quad \quad \downarrow e_V^* \\ E_E \xrightarrow{s_E} V_E^* & E_G \xrightarrow{s_G} V_G^* & E_G \xrightarrow{s_G} V_G^* \\ \quad \quad \quad t_E & \quad \quad \quad t_G & \quad \quad \quad t_G \end{array}$$

2. $\forall \mathbf{e} \in E_C. \bar{c}_V^*(t_C(\mathbf{e})) = t_E(\bar{c}_E(\mathbf{e}))$

The proof for this case works analogously to the proof in the case above.

Thus, we have that $\bar{c} : C \rightarrow E$ is a morphism with $c = e \circ \bar{c}$ because $c_E = e_E \circ \bar{c}_E$ by commutativity of the triangle (5) and $c_V = e_V \circ \bar{c}_V$ holds by construction of the function \bar{c}_V . This implies by Morphism-Pushout-Lemma (see Lemma A.1 in [224]) that \bar{c} is unique and there is a unique morphism $\bar{b} : B \rightarrow D$ in \mathcal{M}_1 with $b = d \circ \bar{b}$ such that furthermore holds that (6) is a pushout in **HyperGraphs**. Hence we directly get that (1) is an initial pushout in $(\text{HyperGraphs}, \mathcal{M}_1)$. \square

Lemma 27: (Well-Definedness of Hypergraph Morphism Translation, see page 126)

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, and the functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Then for each hypergraph morphism $f : G_1 \rightarrow G_2$ the corresponding typed attributed graph morphism $\mathcal{F}_{\text{HG}}(f) : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ is well-defined in $\mathbf{AGraphs}_{\text{HGTG}}$. Moreover, \mathcal{F}_{HG} preserves compositionality, injective morphisms, inclusions, and identities.

Proof.

We have to show the following five steps:

1. The components of $\mathcal{F}_{\text{HG}}(f)$ are well-defined w.r.t. the codomain.
2. The components of $\mathcal{F}_{\text{HG}}(f)$ are compatible with the source and target functions.
3. The components of $\mathcal{F}_{\text{HG}}(f)$ are compatible with the typing morphisms.
4. Compositionality axiom holds for \mathcal{F}_{HG} .
5. $f \in \mathcal{M}_1$ (inclusion, identity) implies that $\mathcal{F}_{\text{HG}}(f) \in \mathcal{M}_2$ (inclusion, identity).

Let $\mathcal{F}_{\text{HG}}(f) : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ be defined by $\mathcal{F}_{\text{HG}}(f) = f' = (f'_{V_G}, f'_{V_D} = \text{id}_{\mathbb{N}}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}})$, where in the short notation $\mathcal{F}_{\text{HG}}(G_i) = (V_G^{G_i}, \mathbb{N}, E_G^{G_i}, E_{NA}^{G_i}, E_{EA}^{G_i}, (s_j^{G_i}, t_j^{G_i})_{j \in \{G, NA, EA\}})$ for $i \in \{1, 2\}$.

1. $f'_{V_G}, f'_{V_D}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}}$ are well-defined w.r.t. the codomain.

- a) f'_{V_G} is well-defined, i.e., $f'_{V_G}(x) \in V_G^{G_2}$ for $x \in V_G^{G_1}$:

- Case 1: Let $x \in V_{G_1} \subseteq V_G^{G_1}$.

$f'_{V_G}(x) = f_V(x) \in V_{G_2}$, because $f = (f_V, f_E)$ is a hypergraph morphism
 $\Rightarrow f_V(x) \in V_G^{G_2}$.

- Case 2: Let $x \in E_{G_1} \subseteq V_G^{G_1}$.

$f'_{V_G}(x) = f_E(x) \in E_{G_2}$, because $f = (f_V, f_E)$ is a hypergraph morphism
 $\Rightarrow f_E(x) \in V_G^{G_2}$.

- b) f'_{V_D} is well-defined, i.e., $f'_{V_D}(i) \in V_D^{G_2}$ for $i \in V_D^{G_1}$:

- Let $i \in V_D^{G_1}$ with $V_D^{G_1} = \mathbb{N}$.

$f'_{V_D}(i) = \text{id}_{\mathbb{N}}(i) = i \in \mathbb{N}$
 $\Rightarrow i \in V_D^{G_2}$.

- c) f'_{E_G} is well-defined, i.e., $f'_{E_G}(x, y, n) \in E_G^{G_2}$ for $(x, y, n) \in E_G^{G_1} = E_{n2e}^{G_1} \uplus E_{e2n}^{G_1}$:

- Case 1: Let $(v, e, n) \in E_{n2e}^{G_1}$ with $v \in V_{G_1}, e \in E_{G_1}, n \in \mathbb{N}$, s.t. $s_{G_1}^n(e) = v$.

$f'_{E_G}(v, e, n) = (f_V(v), f_E(e), n)$ with $f_V(v) \in V_{G_2}$ and $f_E(e) \in E_{G_2}$
 $\Rightarrow s_{G_2}^n(f_E(e)) = f_V(v)$, i.e., $f_V(v)$ is one of the source nodes of the hyperedge
 $f_E(e)$ in G_2 , because $f = (f_V, f_E)$ is a hypergraph morphism
 $\Rightarrow (f_V(v), f_E(e), n) \in E_{n2e}^{G_2} \subseteq E_G^{G_2}$.

- Case 2: Let $(e, v, n) \in E_{e2n}^{G_1}$ with $e \in E_{G_1}, v \in V_{G_1}, n \in \mathbb{N}$, s.t. $t_{G_1}^n(e) = v$.

$f'_{E_G}(e, v, n) = (f_E(e), f_V(v), n)$ with $f_E(e) \in E_{G_2}$ and $f_V(v) \in V_{G_2}$
 $\Rightarrow t_{G_2}^n(f_E(e)) = f_V(v)$, i.e., $f_V(v)$ is one of the target nodes of the hyperedge
 $f_E(e)$ in G_2 , because $f = (f_V, f_E)$ is a hypergraph morphism
 $\Rightarrow (f_E(e), f_V(v), n) \in E_{e2n}^{G_2} \subseteq E_G^{G_2}$.

- d) $f'_{E_{NA}}$ is well-defined, i.e., $f'_{E_{NA}}(x, y, z) \in E_{NA}^{G_2}$ for $(x, y, z) \in E_{NA}^{G_1} = E_{in}^{G_1} \uplus E_{out}^{G_1}$:

- Case 1: Let $(e, n, in) \in E_{in}^{G_1}$ with $e \in E_{G_1}, n \in \mathbb{N}$, s.t. $|s_{G_1}(e)| = n$.

$f'_{E_{NA}}(e, n, in) = (f_E(e), n, in)$ with $f_E(e) \in E_{G_2}$
 $\Rightarrow |s_{G_2}(f_E(e))| = |f_V^*(s_{G_1}(e))| = |s_{G_1}(e)| = n$, because $f = (f_V, f_E)$ is a

hypergraph morphism

$$\Rightarrow (f_E(e), n, in) \in E_{in}^{G_2} \subseteq E_{NA}^{G_2}.$$

- **Case 2:** Let $(e, n, out) \in E_{out}^{G_1}$ with $e \in E_{G_1}, n \in \mathbb{N}$, s.t. $|t_{G_1}(e)| = n$.
 $f'_{E_{NA}}(e, n, out) = (f_E(e), n, out)$ with $f_E(e) \in E_{G_2}$
 $\Rightarrow |t_{G_2}(f_E(e))| = |f_V^*(t_{G_1}(e))| = |t_{G_1}(e)| = n$, because $f = (f_V, f_E)$ is a hypergraph morphism

$$\Rightarrow (f_E(e), n, out) \in E_{out}^{G_2} \subseteq E_{NA}^{G_2}.$$

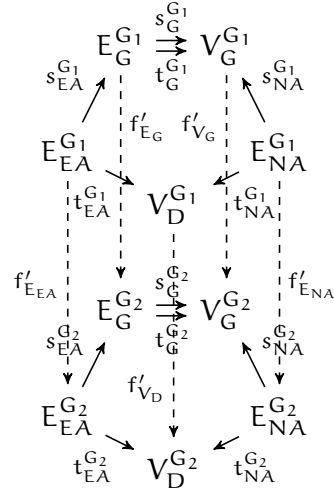
- e) $f'_{E_{EA}}$ is well-defined, i.e., $f'_{E_{EA}}(x, y, z) \in E_{EA}^{G_2}$ for $(x, y, z) \in E_{EA}^{G_1} = E_s^{G_1} \uplus E_t^{G_1}$:

- **Case 1:** Let $(n, v, e) \in E_s^{G_1}$ with $n \in \mathbb{N}, v \in V_{G_1}, e \in E_{G_1}$, s.t. $s_{G_1}^n(e) = v$.
 $f'_{E_{EA}}(n, v, e) = (n, f_V(v), f_E(e))$ with $f_V(v) \in V_{G_2}$ and $f_E(e) \in E_{G_2}$
 $\Rightarrow s_{G_2}^n(f_E(e)) = f_V(v)$, because $f = (f_V, f_E)$ is a hypergraph morphism
 $\Rightarrow (n, f_V(v), f_E(e)) \in E_s^{G_2} \subseteq E_{EA}^{G_2}$.
- **Case 2:** Let $(n, e, v) \in E_t^{G_1}$ with $n \in \mathbb{N}, e \in E_{G_1}, v \in V_{G_1}$, s.t. $t_{G_1}^n(e) = v$.
 $f'_{E_{EA}}(n, e, v) = (n, f_E(e), f_V(v))$ with $f_E(e) \in E_{G_2}$ and $f_V(v) \in V_{G_2}$
 $\Rightarrow t_{G_2}^n(f_E(e)) = f_V(v)$, because $f = (f_V, f_E)$ is a hypergraph morphism
 $\Rightarrow (n, f_E(e), f_V(v)) \in E_t^{G_2} \subseteq E_{EA}^{G_2}$.

2. $f'_{V_G}, f'_{V_D}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}}$ are compatible with the source and target functions.

To show:

- $f'_{V_D} \circ t_{EA}^{G_1} = t_{EA}^{G_2} \circ f'_{E_{EA}}$,
- $t_{NA}^{G_2} \circ f'_{E_{NA}} = f'_{V_D} \circ t_{NA}^{G_1}$,
- $s_{NA}^{G_2} \circ f'_{E_{NA}} = f'_{V_G} \circ s_{NA}^{G_1}$,
- $s_{EA}^{G_2} \circ f'_{E_{EA}} = f'_{E_G} \circ s_{EA}^{G_1}$,
- $s_G^{G_2} \circ f'_{E_G} = f'_{V_G} \circ s_G^{G_1}$,
- $t_G^{G_2} \circ f'_{E_G} = f'_{V_G} \circ t_G^{G_1}$.



Part 2a:

Case 1: Let $(n, v, e) \in E_s^{G_1} \subseteq E_{EA}^{G_1}$ with $n \in \mathbb{N}, v \in V_{G_1}$ and $e \in E_{G_1}$.

$$\begin{aligned} (f'_{V_D} \circ t_{EA}^{G_1})(n, v, e) &= f'_{V_D}(t_{EA}^{G_1}(n, v, e)) = f'_{V_D}(n) = n = t_{EA}^{G_2}(n, f_V(v), f_E(e)) \\ &= t_{EA}^{G_2}(f'_{E_{EA}}(n, v, e)) = (t_{EA}^{G_2} \circ f'_{E_{EA}})(n, v, e) \end{aligned}$$

Case 2: Let $(n, e, v) \in E_t^{G_1} \subseteq E_{EA}^{G_1}$ with $n \in \mathbb{N}, e \in E_{G_1}$ and $v \in V_{G_1}$: similarly to Case 1 replacing v by e and e by v .

Part 2b:

Let $(e, n, x) \in E_{NA}^{G_1}$ with $x \in \{in, out\}$.

$$\begin{aligned} (t_{NA}^{G_2} \circ f'_{E_{NA}})(e, n, x) &= t_{NA}^{G_2}(f'_{E_{NA}}(e, n, x)) = t_{NA}^{G_2}(f_E(e), n, x) = n = f'_{V_D}(n) \\ &= f'_{V_D}(t_{NA}^{G_1}(e, n, x)) = (f'_{V_D} \circ t_{NA}^{G_1})(e, n, x) \end{aligned}$$

Part 2c:

Let $(e, n, x) \in E_{NA}^{G_1}$ with $x \in \{in, out\}$.

$$(s_{NA}^{G_2} \circ f'_{E_{NA}})(e, n, x) = s_{NA}^{G_2}(f'_{E_{NA}}(e, n, x)) = s_{NA}^{G_2}(f_E(e), n, x) = f_E(e) = f'_{V_G}(e)$$

$$= f'_{V_G}(s_{NA}^{G_1}(e, n, x)) = (f'_{V_G} \circ s_{NA}^{G_1})(e, n, x)$$

Part 2d:

Case 1: Let $(n, v, e) \in E_s^{G_1} \subseteq E_{EA}^{G_1}$ with $n \in \mathbb{N}, v \in V_{G_1}$ and $e \in E_{G_1}$.

$$\begin{aligned} (s_{EA}^{G_2} \circ f'_{EA})(n, v, e) &= s_{EA}^{G_2}(f'_{EA}(n, v, e)) = s_{EA}^{G_2}(n, f_V(v), f_E(e)) \\ &= (f_V(v), f_E(e), n) = f'_{EG}(v, e, n) = f'_{EG}(s_{EA}^{G_1}(n, v, e)) \\ &= (f'_{EG} \circ s_{EA}^{G_1})(n, v, e) \end{aligned}$$

Case 2: Let $(n, e, v) \in E_t^{G_1} \subseteq E_{EA}^{G_1}$ with $n \in \mathbb{N}, e \in E_{G_1}$ and $v \in V_{G_1}$: similarly to Case 1 replacing v by e and e by v .

Part 2e:

Case 1: Let $(v, e, n) \in E_{n2e}^{G_1} \subseteq E_G^{G_1}$ with $v \in V_{G_1}, e \in E_{G_1}$ and $n \in \mathbb{N}$.

$$\begin{aligned} (s_G^{G_2} \circ f'_{EG})(v, e, n) &= s_G^{G_2}(f'_{EG}(v, e, n)) = s_G^{G_2}(f_V(v), f_E(e), n) = f_V(v) = f'_{VG}(v) \\ &= f'_{VG}(s_G^{G_1}(v, e, n)) = (f'_{VG} \circ s_G^{G_1})(v, e, n) \end{aligned}$$

Case 2: Let $(e, v, n) \in E_{e2n}^{G_1} \subseteq E_G^{G_1}$ with $e \in E_{G_1}, v \in V_{G_1}$ and $n \in \mathbb{N}$: similarly to Case 1 replacing v by e and e by v .

Part 2f:

Case 1: Let $(v, e, n) \in E_{n2e}^{G_1} \subseteq E_G^{G_1}$ with $v \in V_{G_1}, e \in E_{G_1}$ and $n \in \mathbb{N}$.

$$\begin{aligned} (t_G^{G_2} \circ f'_{EG})(v, e, n) &= t_G^{G_2}(f'_{EG}(v, e, n)) = t_G^{G_2}(f_V(v), f_E(e), n) = f_E(e) = f'_{VG}(e) \\ &= f'_{VG}(t_G^{G_1}(v, e, n)) = (f'_{VG} \circ t_G^{G_1})(v, e, n) \end{aligned}$$

Case 2: Let $(e, v, n) \in E_{e2n}^{G_1} \subseteq E_G^{G_1}$ with $e \in E_{G_1}, v \in V_{G_1}$ and $n \in \mathbb{N}$: similarly to Case 1 replacing v by e and e by v .

3. $f'_{V_G}, f'_{V_D}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}}$ are compatible with the typing morphisms.

Consider $\mathcal{F}_{HG}(G_1) = ((G_1', NAT), type^{G_1})$ and $\mathcal{F}_{HG}(G_2) = ((G_2', NAT), type^{G_2})$.

To show: $type^{G_2} \circ \mathcal{F}_{HG}(f) = type^{G_1}$ with $type^{G_i} = (type_{V_G}^{G_i}, type_{V_D}^{G_i}, type_{E_G}^{G_i}, type_{E_{NA}}^{G_i}, type_{E_{EA}}^{G_i})$ where $i \in \{1, 2\}$ and $\mathcal{F}_{HG}(f) = f' = (f'_{V_G}, f'_{V_D}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}})$.

Or in particular:

- a) $type_{V_G}^{G_2} \circ f'_{V_G} = type_{V_G}^{G_1}$,
- b) $type_{V_D}^{G_2} \circ f'_{V_D} = type_{V_D}^{G_1}$,
- c) $type_{E_G}^{G_2} \circ f'_{E_G} = type_{E_G}^{G_1}$,
- d) $type_{E_{NA}}^{G_2} \circ f'_{E_{NA}} = type_{E_{NA}}^{G_1}$,
- e) $type_{E_{EA}}^{G_2} \circ f'_{E_{EA}} = type_{E_{EA}}^{G_1}$.

$$\begin{array}{ccc} \mathcal{F}_{HG}(G_1) & \xrightarrow{\mathcal{F}_{HG}(f)} & \mathcal{F}_{HG}(G_2) \\ & \searrow \text{type}^{G_1} & \swarrow \text{type}^{G_2} \\ & HGTG & \end{array}$$

Part 3a:

Case 1: Let $v \in V_{G_1} \subseteq V_G^{G_1}$.

$$(type_{V_G}^{G_2} \circ f'_{V_G})(v) = type_{V_G}^{G_2}(f'_{V_G}(v)) = type_{V_G}^{G_2}(f_V(v)) \stackrel{f_V(v) \in V_{G_2} \text{ Node}}{=} type_{V_G}^{G_1}(v)$$

Case 2: Let $e \in E_{G_1} \subseteq V_G^{G_1}$.

$$(type_{V_G}^{G_2} \circ f'_{V_G})(e) = type_{V_G}^{G_2}(f'_{V_G}(e)) = type_{V_G}^{G_2}(f_E(e)) \stackrel{f_E(e) \in E_{G_2} \text{ Edge}}{=} type_{V_G}^{G_1}(e)$$

Part 3b:

Let $i \in \mathbb{N}$.

$$(type_{V_D}^{G_2} \circ f'_{V_D})(i) = type_{V_D}^{G_2}(f'_{V_D}(i)) = type_{V_D}^{G_2}(id_{\mathbb{N}}(i)) = type_{V_D}^{G_2}(i) = nat = type_{V_D}^{G_1}(i)$$

Part 3c:

Case 1: Let $(v, e, n) \in E_{n2e}^{G_1} \subseteq E_{G_1}^{G_1}$ with $v \in V_{G_1}$, $e \in E_{G_1}$ and $n \in \mathbb{N}$.

$$(type_{E_G}^{G_2} \circ f'_{E_G})(v, e, n) = type_{E_G}^{G_2}(f'_{E_G}(v, e, n)) = type_{E_G}^{G_2}(f_V(v), f_E(e), n) \\ \stackrel{f_V(v) \in V_{G_2} \wedge f_E(e) \in E_{G_2}}{=} node2edge = type_{E_G}^{G_1}(v, e, n)$$

Case 2: Let $(e, v, n) \in E_{e2n}^{G_1} \subseteq E_{G_1}^{G_1}$ with $e \in E_{G_1}$, $v \in V_{G_1}$ and $n \in \mathbb{N}$.

$$(type_{E_G}^{G_2} \circ f'_{E_G})(e, v, n) = type_{E_G}^{G_2}(f'_{E_G}(e, v, n)) = type_{E_G}^{G_2}(f_E(e), f_V(v), n) \\ \stackrel{f_E(e) \in E_{G_2} \wedge f_V(v) \in V_{G_2}}{=} edge2node = type_{E_G}^{G_1}(e, v, n)$$

Part 3d:

Case 1: Let $(e, n, in) \in E_{in}^{G_1} \subseteq E_{NA}^{G_1}$ with $e \in E_{G_1}$ and $n \in \mathbb{N}$.

$$(type_{E_{NA}}^{G_2} \circ f'_{E_{NA}})(e, n, in) = type_{E_{NA}}^{G_2}(f'_{E_{NA}}(e, n, in)) = type_{E_{NA}}^{G_2}(f_E(e), n, in) \\ \stackrel{f_E(e) \in E_{G_2}}{=} in = type_{E_{NA}}^{G_1}(e, n, in)$$

Case 2: Let $(e, n, out) \in E_{out}^{G_1} \subseteq E_{NA}^{G_1}$ with $e \in E_{G_1}$ and $n \in \mathbb{N}$: similarly to Case 1 replacing *in* by *out*.

Part 3e:

Case 1: Let $(n, v, e) \in E_s^{G_1} \subseteq E_{EA}^{G_1}$ with $n \in \mathbb{N}$, $v \in V_{G_1}$ and $e \in E_{G_1}$.

$$(type_{E_{EA}}^{G_2} \circ f'_{E_{EA}})(n, v, e) = type_{E_{EA}}^{G_2}(f'_{E_{EA}}(n, v, e)) = type_{E_{EA}}^{G_2}(n, f_V(v), f_E(e)) \\ \stackrel{f_V(v) \in V_{G_2} \wedge f_E(e) \in E_{G_2}}{=} number = type_{E_{EA}}^{G_1}(n, v, e)$$

Case 2: Let $(n, e, v) \in E_t^{G_1} \subseteq E_{EA}^{G_1}$ with $n \in \mathbb{N}$, $e \in E_{G_1}$ and $v \in V_{G_1}$: similarly to Case 1 replacing *v* by *e* and *e* by *v*.

4. **Compositionality axiom holds for \mathcal{F}_{HG} .**

Consider hypergraph morphisms $f : G_2 \rightarrow G_3$ and $g : G_1 \rightarrow G_2$.

To show: $\mathcal{F}_{HG}(f \circ g) = \mathcal{F}_{HG}(f) \circ \mathcal{F}_{HG}(g)$.

- We consider the V_G -component of some typed attributed graph morphism.

Case 1: Let $v \in V_{G_1}$.

$$\begin{aligned} & \mathcal{F}_{HG}(f \circ g)_{V_G}(v) \\ &= ((f \circ g)_V \uplus (f \circ g)_E)(v) \\ &= (f \circ g)_V(v) \\ &\stackrel{Def.}{=} (f_V \circ g_V)(v) \\ &= ((f_V \uplus f_E) \circ (g_V \uplus g_E))(v) \\ &= (\mathcal{F}_{HG}(f)_{V_G} \circ \mathcal{F}_{HG}(g)_{V_G})(v) \\ &\stackrel{Def.}{=} (\mathcal{F}_{HG}(f) \circ \mathcal{F}_{HG}(g))_{V_G}(v) \end{aligned}$$

Case 2: Let $e \in E_{G_1}$. The proof works similar to the Case 1.

- We consider the V_D -component of some typed attributed graph morphism.
Let $x \in \mathbb{N}$.

$$\begin{aligned}
& \mathcal{F}_{HG}(f \circ g)_{V_D}(x) \\
&= id_{\mathbb{N}}(x) \\
&= (id_{\mathbb{N}} \circ id_{\mathbb{N}})(x) \\
&= (\mathcal{F}_{HG}(f)_{V_D} \circ \mathcal{F}_{HG}(g)_{V_D})(x) \\
&= (\mathcal{F}_{HG}(f) \circ \mathcal{F}_{HG}(g))_{V_D}(x)
\end{aligned}$$

- We consider the E_G -component of some typed attributed graph morphism.
Case 1: Let $(v, e, n) \in E_{n2e}^{G_1}$.

$$\begin{aligned}
& \mathcal{F}_{HG}(f \circ g)_{E_G}(v, e, n) \\
&= ((f \circ g)_V(v), (f \circ g)_E(e), n) \\
&\stackrel{Def.}{=} ((f_V \circ g_V)(v), (f_E \circ g_E)(e), n) \\
&= (f_V(g_V(v)), f_E(g_E(e)), n) \\
&= \mathcal{F}_{HG}(f)_{E_G}(g_V(v), g_E(e), n) \\
&= \mathcal{F}_{HG}(f)_{E_G}(\mathcal{F}_{HG}(g)_{E_G}(v, e, n)) \\
&= (\mathcal{F}_{HG}(f)_{E_G} \circ \mathcal{F}_{HG}(g)_{E_G})(v, e, n) \\
&\stackrel{Def.}{=} (\mathcal{F}_{HG}(f) \circ \mathcal{F}_{HG}(g))_{E_G}(v, e, n)
\end{aligned}$$

Case 2: Let $(e, v, n) \in E_{e2n}^{G_1}$. The proof is similar to the Case 1 replacing v by e and e by v .

- We consider the E_{NA} -component of some typed attributed graph morphism.
Let $(e, n, x) \in E_{in}^{G_1} \uplus E_{out}^{G_1}$.

$$\begin{aligned}
& \mathcal{F}_{HG}(f \circ g)_{E_{NA}}(e, n, x) \\
&= ((f \circ g)_E(e), n, x) \\
&\stackrel{Def.}{=} ((f_E \circ g_E)(e), n, x) \\
&= (f_E(g_E(e)), n, x) \\
&= \mathcal{F}_{HG}(f)_{E_{NA}}(g_E(e), n, x) \\
&= \mathcal{F}_{HG}(f)_{E_{NA}}(\mathcal{F}_{HG}(g)_{E_{NA}}(e, n, x)) \\
&= (\mathcal{F}_{HG}(f)_{E_{NA}} \circ \mathcal{F}_{HG}(g)_{E_{NA}})(e, n, x) \\
&\stackrel{Def.}{=} (\mathcal{F}_{HG}(f) \circ \mathcal{F}_{HG}(g))_{E_{NA}}(e, n, x)
\end{aligned}$$

- We consider the E_{EA} -component of some typed attributed graph morphism.
Case 1: Let $(n, v, e) \in E_s^{G_1}$.

$$\begin{aligned}
& \mathcal{F}_{HG}(f \circ g)_{E_{EA}}(n, v, e) \\
&= (n, (f \circ g)_V(v), (f \circ g)_E(e)) \\
&\stackrel{Def.}{=} (n, (f_V \circ g_V)(v), (f_E \circ g_E)(e)) \\
&= (n, f_V(g_V(v)), f_E(g_E(e)))
\end{aligned}$$

$$\begin{aligned}
&= \mathcal{F}_{\text{HG}}(f)_{\text{E}_{\text{EA}}}(\mathbf{n}, g_{\text{V}}(v), g_{\text{E}}(e)) \\
&= \mathcal{F}_{\text{HG}}(f)_{\text{E}_{\text{EA}}}(\mathcal{F}_{\text{HG}}(g)_{\text{E}_{\text{EA}}}(\mathbf{n}, v, e)) \\
&= (\mathcal{F}_{\text{HG}}(f)_{\text{E}_{\text{EA}}} \circ \mathcal{F}_{\text{HG}}(g)_{\text{E}_{\text{EA}}})(\mathbf{n}, v, e) \\
&\stackrel{\text{Def.}}{=} (\mathcal{F}_{\text{HG}}(f) \circ \mathcal{F}_{\text{HG}}(g))_{\text{E}_{\text{EA}}}(\mathbf{n}, v, e)
\end{aligned}$$

Case 2: Let $(\mathbf{n}, e, v) \in \text{E}_{\text{t}}^{\text{G}^1}$. The proof is similar to the Case 1 replacing v by e and e by v .

5. $f \in \mathcal{M}_1$ (**inclusion, identity**) implies that $\mathcal{F}_{\text{HG}}(f) \in \mathcal{M}_2$ (**inclusion, identity**).

a) Let $f = (f_{\text{V}}, f_{\text{E}}) \in \mathcal{M}_1$, i.e., $f_{\text{V}}, f_{\text{E}}$ are injective.

To show: $\mathcal{F}_{\text{HG}}(f) = f' = (f'_{\text{V}_{\text{G}}}, f'_{\text{V}_{\text{D}}}, f'_{\text{E}_{\text{G}}}, f'_{\text{E}_{\text{NA}}}, f'_{\text{E}_{\text{EA}}}) \in \mathcal{M}_2$, i.e., $f'_{\text{V}_{\text{G}}}, f'_{\text{V}_{\text{D}}}, f'_{\text{E}_{\text{G}}}, f'_{\text{E}_{\text{NA}}}$ and $f'_{\text{E}_{\text{EA}}}$ are injective.

i. $f'_{\text{V}_{\text{G}}} = f_{\text{V}} \uplus f_{\text{E}}$ is obviously injective for injective f_{V} and f_{E} .

ii. $f'_{\text{V}_{\text{D}}} = \text{id}_{\text{N}}$ is obviously injective.

iii. Consider arbitrary (x_1, y_1, z_1) and (x_2, y_2, z_2) from $\text{E}_{\text{G}}^{\text{G}^1}$.

A. Let $x_1, x_2 \in \text{V}_{\text{G}}$:

$$\begin{aligned}
&f'_{\text{E}_{\text{G}}}(x_1, y_1, z_1) = f'_{\text{E}_{\text{G}}}(x_2, y_2, z_2) \\
&\Rightarrow (f_{\text{V}}(x_1), f_{\text{E}}(y_1), z_1) = (f_{\text{V}}(x_2), f_{\text{E}}(y_2), z_2) \\
&\Rightarrow f_{\text{V}}(x_1) = f_{\text{V}}(x_2) \text{ and } f_{\text{E}}(y_1) = f_{\text{E}}(y_2) \text{ and } z_1 = z_2 \\
&\stackrel{f_{\text{V}}, f_{\text{E}} \text{ inj.}}{\Rightarrow} (x_1, y_1, z_1) = (x_2, y_2, z_2)
\end{aligned}$$

B. Let $x_1, x_2 \in \text{E}_{\text{G}}$: analogously to the previous case applying f_{E} to the first and f_{V} to the second component of both triples.

C. Let $x_1 \in \text{E}_{\text{G}}$ and $x_2 \in \text{V}_{\text{G}}$:

$$\begin{aligned}
&f'_{\text{E}_{\text{G}}}(x_1, y_1, z_1) = f'_{\text{E}_{\text{G}}}(x_2, y_2, z_2) \\
&\Rightarrow (f_{\text{E}}(x_1), f_{\text{V}}(y_1), z_1) = (f_{\text{V}}(x_2), f_{\text{E}}(y_2), z_2) \\
&\Rightarrow f_{\text{E}}(x_1) = f_{\text{V}}(x_2) \\
&\Rightarrow (x_1, y_1, z_1) = (x_2, y_2, z_2), \text{ because } f_{\text{E}}, f_{\text{V}} \text{ have disjoint codomains}
\end{aligned}$$

D. Let $x_1 \in \text{V}_{\text{G}}$ and $x_2 \in \text{E}_{\text{G}}$: analogously to the previous case applying f_{V} to the first and f_{E} to the second component of the first triple and vice versa for the second triple.

iv. Consider arbitrary (x_1, y_1, z_1) and (x_2, y_2, z_2) from $\text{E}_{\text{NA}}^{\text{G}^1}$.

$$\begin{aligned}
&f'_{\text{E}_{\text{NA}}}(x_1, y_1, z_1) = f'_{\text{E}_{\text{NA}}}(x_2, y_2, z_2) \\
&\Rightarrow (f_{\text{E}}(x_1), y_1, z_1) = (f_{\text{E}}(x_2), y_2, z_2) \\
&\Rightarrow f_{\text{E}}(x_1) = f_{\text{E}}(x_2) \text{ and } y_1 = y_2 \text{ and } z_1 = z_2 \\
&\stackrel{f_{\text{E}} \text{ inj.}}{\Rightarrow} (x_1, y_1, z_1) = (x_2, y_2, z_2)
\end{aligned}$$

v. Consider arbitrary (x_1, y_1, z_1) and (x_2, y_2, z_2) from $\text{E}_{\text{EA}}^{\text{G}^1}$.

A. Let $y_1, y_2 \in \text{V}_{\text{G}}$:

$$\begin{aligned}
&f'_{\text{E}_{\text{EA}}}(x_1, y_1, z_1) = f'_{\text{E}_{\text{EA}}}(x_2, y_2, z_2) \\
&\Rightarrow (x_1, f_{\text{V}}(y_1), f_{\text{E}}(z_1)) = (x_2, f_{\text{V}}(y_2), f_{\text{E}}(z_2))
\end{aligned}$$

$$\Rightarrow x_1 = x_2 \text{ and } f_V(y_1) = f_V(y_2) \text{ and } f_E(z_1) = f_E(z_2)$$

$$\xRightarrow{f_V, f_E \text{ inj.}} (x_1, y_1, z_1) = (x_2, y_2, z_2)$$

B. Let $y_1, y_2 \in E_G$: analogously to the previous case applying f_E to the second and f_V to the third component of both triples.

C. Let $y_1 \in E_G$ and $y_2 \in V_G$:

$$f'_{E_{EA}}(x_1, y_1, z_1) = f'_{E_{EA}}(x_2, y_2, z_2)$$

$$\Rightarrow (x_1, f_E(y_1), f_V(z_1)) = (x_2, f_V(y_2), f_E(z_2))$$

$$\Rightarrow f_E(y_1) = f_V(y_2)$$

$$\Rightarrow (x_1, y_1, z_1) = (x_2, y_2, z_2), \text{ because } f_E, f_V \text{ have disjoint codomains}$$

D. Let $y_1 \in V_G$ and $y_2 \in E_G$: analogously to the previous case applying f_V to the second and f_E to the third component of the first triple and vice versa for the second triple.

b) Consider inclusions (identities) $f_A : A \rightarrow A'$ with $f_A(a) = a$ and $f_B : B \rightarrow B'$ with $f_B(b) = b$.

To show: $f_A \uplus f_B : A \uplus B \rightarrow A' \uplus B'$ with $A \uplus B = (A \times \{1\} \cup B \times \{2\})$ and $A' \uplus B' = (A' \times \{1\} \cup B' \times \{2\})$ is an inclusion (identity).

Case 1: Let $i = 1$ and $x \in A \uplus B$:

$$(f_A \uplus f_B)(x, i) = (f_A \uplus f_B)(x, 1) = (f_A(x), 1) = (x, 1)$$

Case 2: Let $i = 2$ and $x \in A \uplus B$:

$$(f_A \uplus f_B)(x, i) = (f_A \uplus f_B)(x, 2) = (f_B(x), 2) = (x, 2)$$

□

Lemma 28: (\mathcal{F}_{HG} Preserves Pushouts along Injective Morphisms [211], see page 129)

Consider \mathcal{M} -adhesive categories (**HyperGraphs**, \mathcal{M}_1), (**AGraphs**_{HGTG}, \mathcal{M}_2), functor $\mathcal{F}_{HG} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{HGTG}$ introduced in Definition 63, hypergraphs G_i for $i \in \{0, 1, 2, 3\}$ with hypergraph morphisms $b = (b_V, b_E)$, $c = (c_V, c_E)$, $g = (g_V, g_E)$, $h = (h_V, h_E)$, and typed attributed graphs $\mathcal{F}_{HG}(G_i)$ for $i \in \{0, 1, 2, 3\}$ with typed attributed graph morphisms $\mathcal{F}_{HG}(b) = b' = (b'_{V_G}, b'_{V_D}, b'_{E_G}, b'_{E_{NA}}, b'_{E_{EA}})$, $\mathcal{F}_{HG}(c) = c' = (c'_{V_G}, c'_{V_D}, c'_{E_G}, c'_{E_{NA}}, c'_{E_{EA}})$, $\mathcal{F}_{HG}(g) = g' = (g'_{V_G}, g'_{V_D}, g'_{E_G}, g'_{E_{NA}}, g'_{E_{EA}})$, $\mathcal{F}_{HG}(h) = h' = (h'_{V_G}, h'_{V_D}, h'_{E_G}, h'_{E_{NA}}, h'_{E_{EA}})$. If (1) is a pushout in **HyperGraphs** with $b \in \mathcal{M}_1$ then we have that (2) is a pushout in **AGraphs**_{HGTG} with $\mathcal{F}_{HG}(b) \in \mathcal{M}_2$.

$$\begin{array}{ccc} G_0 & \xrightarrow{b} & G_1 \\ c \downarrow & (1) & \downarrow g \\ G_2 & \xrightarrow{h} & G_3 \end{array} \qquad \begin{array}{ccc} \mathcal{F}_{HG}(G_0) & \xrightarrow{\mathcal{F}_{HG}(b)=b'} & \mathcal{F}_{HG}(G_1) \\ \mathcal{F}_{HG}(c)=c' \downarrow & (2) & \downarrow \mathcal{F}_{HG}(g)=g' \\ \mathcal{F}_{HG}(G_2) & \xrightarrow{\mathcal{F}_{HG}(h)=h'} & \mathcal{F}_{HG}(G_3) \end{array}$$

Proof.

Let (1) be a pushout in **HyperGraphs** with $b \in \mathcal{M}_1$, i.e., V- and E-components of (1) are pushouts in **Sets**, because pushouts in **HyperGraphs** are constructed componentwise according to Definition 36.

$$\begin{array}{ccc}
V_0 & \xrightarrow{b_V \text{ inj.}} & V_1 \\
c_V \downarrow & (PO) & \downarrow g_V \\
V_2 & \xrightarrow{h_V} & V_3
\end{array}
\qquad
\begin{array}{ccc}
E_0 & \xrightarrow{b_E \text{ inj.}} & E_1 \\
c_E \downarrow & (PO) & \downarrow g_E \\
E_2 & \xrightarrow{h_E} & E_3
\end{array}$$

To show: V_G -, V_D -, E_G -, E_{NA} - and E_{EA} -components of (2) are pushouts in **Sets** with $\mathcal{F}_{HG}(b) = b' \in \mathcal{M}_2$, because pushouts in **AGraphs_{HGTG}** are constructed componentwise as well.

For the given morphism $b \in \mathcal{M}_1$ we have that $\mathcal{F}_{HG}(b) = b' \in \mathcal{M}_2$ according to Lemma 27.

1. V_G -component of (2) is a pushout in **Sets** (see diagram (3)) with $f'_{V_G} = f_V \uplus f_E$ for $f \in \{b, c, g, h\}$, because pushouts are compatible with coproducts (and coproduct in **Sets** is the disjoint union \uplus).

$$\begin{array}{ccc}
V_G^{G_0} & \xrightarrow{b_V \uplus b_E} & V_G^{G_1} \\
c_V \uplus c_E \downarrow & (3) & \downarrow g_V \uplus g_E \\
V_G^{G_2} & \xrightarrow{h_V \uplus h_E} & V_G^{G_3}
\end{array}$$

2. V_D -component of (2) is obviously a pushout in **Sets** (see diagram (4)) with $f'_{V_D} = \text{id}_{\mathbb{N}}$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
V_D^{G_0} = \mathbb{N} & \xrightarrow{\text{id}_{\mathbb{N}}} & V_D^{G_1} = \mathbb{N} \\
\text{id}_{\mathbb{N}} \downarrow & (4) & \downarrow \text{id}_{\mathbb{N}} \\
V_D^{G_2} = \mathbb{N} & \xrightarrow{\text{id}_{\mathbb{N}}} & V_D^{G_3} = \mathbb{N}
\end{array}$$

3. For the E_G -component we have to show that (5) is a pushout in **Sets** which holds if (5a) and similarly (5b) are pushouts.

$$\begin{array}{ccc}
E_G^{G_0} & \xrightarrow{b'_{E_G}} & E_G^{G_1} \\
c'_{E_G} \downarrow & (5) & \downarrow g'_{E_G} \\
E_G^{G_2} & \xrightarrow{h'_{E_G}} & E_G^{G_3}
\end{array}$$

$$\begin{array}{ccc}
E_{n2e}^{G_0} & \xrightarrow{b_V \times b_E \times \text{id}_{\mathbb{N}}} & E_{n2e}^{G_1} \\
c_V \times c_E \times \text{id}_{\mathbb{N}} \downarrow & (5a) & \downarrow g_V \times g_E \times \text{id}_{\mathbb{N}} \\
E_{n2e}^{G_2} & \xrightarrow{h_V \times h_E \times \text{id}_{\mathbb{N}}} & E_{n2e}^{G_3}
\end{array}
\qquad
\begin{array}{ccc}
E_{e2n}^{G_0} & \xrightarrow{b_E \times b_V \times \text{id}_{\mathbb{N}}} & E_{e2n}^{G_1} \\
c_E \times c_V \times \text{id}_{\mathbb{N}} \downarrow & (5b) & \downarrow g_E \times g_V \times \text{id}_{\mathbb{N}} \\
E_{e2n}^{G_2} & \xrightarrow{h_E \times h_V \times \text{id}_{\mathbb{N}}} & E_{e2n}^{G_3}
\end{array}$$

Diagrams (5a) and (5b) commute, because for each product component we have a pushout in **Sets** by assumption. So it remains to show that (5a) and (5b) are pushouts, because products of pushouts are not necessarily pushouts (the merging morphisms are in general not jointly surjective).

- a) For diagram (5a) we have to show that the diagram (5a') is a pushout in **Sets** with

$$V_i \otimes E_i \otimes \mathbb{N} = \{((v, e), n) \in (V_i \times E_i) \times \mathbb{N} \mid s_i^n(e) = v\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_V \otimes f_E \otimes \text{id}_{\mathbb{N}}$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
V_0 \otimes E_0 \otimes \mathbb{N} & \xrightarrow{b_V \otimes b_E \otimes \text{id}_{\mathbb{N}}} & V_1 \otimes E_1 \otimes \mathbb{N} \\
c_V \otimes c_E \otimes \text{id}_{\mathbb{N}} \downarrow & (5a') & \downarrow g_V \otimes g_E \otimes \text{id}_{\mathbb{N}} \\
V_2 \otimes E_2 \otimes \mathbb{N} & \xrightarrow{h_V \otimes h_E \otimes \text{id}_{\mathbb{N}}} & V_3 \otimes E_3 \otimes \mathbb{N}
\end{array}$$

Since b, c, g, h are hypergraph morphisms, we have that all $f_V \otimes f_E \otimes \text{id}_{\mathbb{N}}$ morphisms for $f \in \{b, c, g, h\}$ are well-defined. Furthermore, we have that the components of $(5a')$ are pushouts and pullbacks in **Sets**, because $b = (b_V, b_E) \in \mathcal{M}_1$ by assumption. Hence, also $(5a')$ is a pullback, because pullbacks are compatible with products and it remains to show that $(h_V \otimes h_E \otimes \text{id}_{\mathbb{N}}, g_V \otimes g_E \otimes \text{id}_{\mathbb{N}})$ are jointly surjective.

Consider $(v_3, e_3, n) \in V_3 \otimes E_3 \otimes \mathbb{N}$. The E -component of $(5a')$ is a pushout in **Sets** s.t. we have $e_1 \in E_1$ with $g_E(e_1) = e_3$ (or we have $e_2 \in E_2$ with $h_E(e_2) = e_3$). Without loss of generality we consider the first case. Let $v_1 = s_1^n(e_1)$ for $n \in \mathbb{N}$, then $(g_V \otimes g_E \otimes \text{id}_{\mathbb{N}})(v_1, e_1, n) = (v_3, e_3, n)$, because $g = (g_V, g_E)$ is a hypergraph morphism, i.e., the compatibility with the corresponding source function holds s.t. we have $s_3^n(e_3) = v_3$ (see the diagram below). Hence, $(5a')$ and $(5a)$ are pushouts.

$$\begin{array}{ccc}
E_1 & \xrightarrow{s_1} & V_1^* \\
g_E \downarrow & \begin{array}{c} t_1 \\ = \\ s_3 \end{array} & \downarrow g_V^* \\
E_3 & \xrightarrow{t_3} & V_3^*
\end{array}$$

b) For diagram $(5b)$ we have to show that the diagram $(5b')$ is a pushout in **Sets** with

$$E_i \otimes V_i \otimes \mathbb{N} = \{(e, v, n) \in (E_i \times V_i) \times \mathbb{N} \mid t_i^n(e) = v\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_E \otimes f_V \otimes \text{id}_{\mathbb{N}}$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
E_0 \otimes V_0 \otimes \mathbb{N} & \xrightarrow{b_E \otimes b_V \otimes \text{id}_{\mathbb{N}}} & E_1 \otimes V_1 \otimes \mathbb{N} \\
c_E \otimes c_V \otimes \text{id}_{\mathbb{N}} \downarrow & (5b') & \downarrow g_E \otimes g_V \otimes \text{id}_{\mathbb{N}} \\
E_2 \otimes V_2 \otimes \mathbb{N} & \xrightarrow{h_E \otimes h_V \otimes \text{id}_{\mathbb{N}}} & E_3 \otimes V_3 \otimes \mathbb{N}
\end{array}$$

The proof for this case is similar to the Case 3a. Hence, $(5b')$ and $(5b)$ are pushouts.

4. For the E_{NA} -component we have to show that (6) is a pushout in **Sets** which holds if $(6a)$ and similarly $(6b)$ are pushouts for $X = \{in\}$ and $Y = \{out\}$.

$$\begin{array}{ccccc}
E_{NA}^{G_0} & \xrightarrow{b'_{E_{NA}}} & E_{NA}^{G_1} & & \\
c'_{E_{NA}} \downarrow & (6) & \downarrow g'_{E_{NA}} & & \\
E_{NA}^{G_2} & \xrightarrow{h'_{E_{NA}}} & E_{NA}^{G_3} & & \\
\begin{array}{ccc}
E_{in}^{G_0} & \xrightarrow{b_E \times \text{id}_{\mathbb{N}} \times \text{id}_X} & E_{in}^{G_1} \\
c_E \times \text{id}_{\mathbb{N}} \times \text{id}_X \downarrow & (6a) & \downarrow g_E \times \text{id}_{\mathbb{N}} \times \text{id}_X \\
E_{in}^{G_2} & \xrightarrow{h_E \times \text{id}_{\mathbb{N}} \times \text{id}_X} & E_{in}^{G_3}
\end{array} & & \begin{array}{ccc}
E_{out}^{G_0} & \xrightarrow{b_E \times \text{id}_{\mathbb{N}} \times \text{id}_Y} & E_{out}^{G_1} \\
c_E \times \text{id}_{\mathbb{N}} \times \text{id}_Y \downarrow & (6b) & \downarrow g_E \times \text{id}_{\mathbb{N}} \times \text{id}_Y \\
E_{out}^{G_2} & \xrightarrow{h_E \times \text{id}_{\mathbb{N}} \times \text{id}_Y} & E_{out}^{G_3}
\end{array}
\end{array}$$

Diagrams $(6a)$ and $(6b)$ commute, because for each product component we have a pushout in **Sets** by assumption. So it remains to show that $(6a)$ and $(6b)$ are pushouts.

a) For diagram (6a) we have to show that the diagram (6a') is a pushout in **Sets** with

$$E_i \otimes \mathbb{N} \otimes X = \{(e, n, in) \mid (e, n) \in E_i \times \mathbb{N} \wedge |s_i(e)| = n\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_E \otimes id_{\mathbb{N}} \otimes id_X$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc} E_0 \otimes \mathbb{N} \otimes X & \xrightarrow{b_E \otimes id_{\mathbb{N}} \otimes id_X} & E_1 \otimes \mathbb{N} \otimes X \\ c_E \otimes id_{\mathbb{N}} \otimes id_X \downarrow & (6a') & \downarrow g_E \otimes id_{\mathbb{N}} \otimes id_X \\ E_2 \otimes \mathbb{N} \otimes X & \xrightarrow{h_E \otimes id_{\mathbb{N}} \otimes id_X} & E_3 \otimes \mathbb{N} \otimes X \end{array}$$

Also here we know that $f_E \otimes id_{\mathbb{N}} \otimes id_X$ morphisms for $f \in \{b, c, g, h\}$ are well-defined, because b, c, g, h are hypergraph morphisms. Furthermore, we have that the components of (6a') are pushouts and pullbacks in **Sets**, because $b = (b_V, b_E) \in \mathcal{M}_1$ by assumption. Hence, also (6a') is a pullback and we have that $(h_E \otimes id_{\mathbb{N}} \otimes id_X, g_E \otimes id_{\mathbb{N}} \otimes id_X)$ are obviously jointly surjective. Therefore, (6a') and (6a) are pushouts.

b) For diagram (6b) we have to show that the diagram (6b') is a pushout in **Sets** with

$$E_i \otimes \mathbb{N} \otimes Y = \{(e, n, out) \mid (e, n) \in E_i \times \mathbb{N} \wedge |t_i(e)| = n\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_E \otimes id_{\mathbb{N}} \otimes id_Y$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc} E_0 \otimes \mathbb{N} \otimes Y & \xrightarrow{b_E \otimes id_{\mathbb{N}} \otimes id_Y} & E_1 \otimes \mathbb{N} \otimes Y \\ c_E \otimes id_{\mathbb{N}} \otimes id_Y \downarrow & (6b') & \downarrow g_E \otimes id_{\mathbb{N}} \otimes id_Y \\ E_2 \otimes \mathbb{N} \otimes Y & \xrightarrow{h_E \otimes id_{\mathbb{N}} \otimes id_Y} & E_3 \otimes \mathbb{N} \otimes Y \end{array}$$

The proof for this case is similar to the Case 4a. Hence, (6b') and (6b) are pushouts.

5. For the E_{EA} -component we have to show that (7) is a pushout in **Sets** which holds if (7a) and similarly (7b) are pushouts.

$$\begin{array}{ccccc} E_{EA}^{G_0} & \xrightarrow{b'_{EA}} & E_{EA}^{G_1} & & \\ c'_{EA} \downarrow & (7) & \downarrow g'_{EA} & & \\ E_{EA}^{G_2} & \xrightarrow{h'_{EA}} & E_{EA}^{G_3} & & \\ \begin{array}{ccc} E_s^{G_0} & \xrightarrow{id_{\mathbb{N}} \times b_V \times b_E} & E_s^{G_1} \\ id_{\mathbb{N}} \times c_V \times c_E \downarrow & (7a) & \downarrow id_{\mathbb{N}} \times g_V \times g_E \\ E_s^{G_2} & \xrightarrow{id_{\mathbb{N}} \times h_V \times h_E} & E_s^{G_3} \end{array} & & \begin{array}{ccc} E_t^{G_0} & \xrightarrow{id_{\mathbb{N}} \times b_E \times b_V} & E_t^{G_1} \\ id_{\mathbb{N}} \times c_E \times c_V \downarrow & (7b) & \downarrow id_{\mathbb{N}} \times g_E \times g_V \\ E_t^{G_2} & \xrightarrow{id_{\mathbb{N}} \times h_E \times h_V} & E_t^{G_3} \end{array} \end{array}$$

Diagrams (7a) and (7b) commute, because also here we have a pushout in **Sets** for each product component by assumption. So it remains to show that (7a) and (7b) are pushouts. The proof for (7a) and (7b) works analogously to the Case 3 of the current proof switching the components of the products. Hence, (7a) and (7b) are pushouts.

□

Lemma 29: (Uniquely Determined \mathcal{F}_{HG} -Images [211], see page 137)

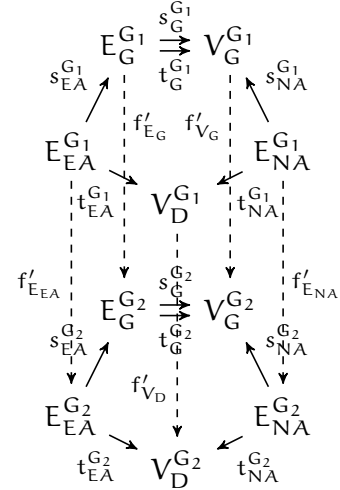
Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63, typed attributed graphs $\mathcal{F}_{\text{HG}}(G_1)$, $\mathcal{F}_{\text{HG}}(G_2)$, and a morphism $f' : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ with $f'_{V_D} = \text{id}_N$. Then we have that f' is uniquely determined by the V_G -component $f'_{V_G} : V_G^{G_1} \rightarrow V_G^{G_2}$ with $V_G^{G_i} = V_{G_i} \uplus E_{G_i}$ for $i \in \{1, 2\}$.

Proof.

Consider a typed attributed graph morphism $f' : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ with $f' = (f'_{V_G}, f'_{V_D} = \text{id}_N, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}})$ given according to the diagram below and $f_V(v) = f'_{V_G}(v)$ for $v \in V_{G_1}$, $f_E(e) = f'_{V_G}(e)$ for $e \in E_{G_1}$.

To show:

1. $f'_{E_G}(v, e, n) = (f_V(v), f_E(e), n)$ for $(v, e, n) \in E_{n2e}^{G_1}$,
2. $f'_{E_G}(e, v, n) = (f_E(e), f_V(v), n)$ for $(e, v, n) \in E_{e2n}^{G_1}$,
3. $f'_{E_{NA}}(e, n, x) = (f_E(e), n, x)$ for $(e, n, x) \in E_{in}^{G_1} \uplus E_{out}^{G_1}$,
4. $f'_{E_{EA}}(n, v, e) = (n, f_V(v), f_E(e))$ for $(n, v, e) \in E_s^{G_1}$ and
5. $f'_{E_{EA}}(n, e, v) = (n, f_E(e), f_V(v))$ for $(n, e, v) \in E_t^{G_1}$.



Part 1:

Let $f'_{E_G}(v, e, n) = (v', e', n')$ with $s_G^{G_2}(v', e', n') = v'$ and $t_G^{G_2}(v', e', n') = e'$. Then we have:

$$\begin{aligned} v' &= s_G^{G_2}(v', e', n') = s_G^{G_2}(f'_{E_G}(v, e, n)) = f'_{V_G}(s_G^{G_1}(v, e, n)) = f'_{V_G}(v) = f_V(v) \\ &\Rightarrow v' = f_V(v), \end{aligned}$$

$$\begin{aligned} e' &= t_G^{G_2}(v', e', n') = t_G^{G_2}(f'_{E_G}(v, e, n)) = f'_{V_G}(t_G^{G_1}(v, e, n)) = f'_{V_G}(e) = f_E(e) \\ &\Rightarrow e' = f_E(e), \end{aligned}$$

$$\begin{aligned} n &= \text{id}_N(t_{EA}^{G_1}(n, v, e)) = t_{EA}^{G_2}(f'_{E_{EA}}(n, v, e)) \stackrel{(*)}{=} t_{EA}^{G_2}(n', v', e') = n' \\ &\Rightarrow n' = n. \end{aligned}$$

Where the step $(*)$ holds since we have the following:

$$\begin{aligned} s_{EA}^{G_2}(f'_{E_{EA}}(n, v, e)) &= f'_{E_G}(s_{EA}^{G_1}(n, v, e)) = f'_{E_G}(v, e, n) \\ &= (v', e', n') = s_{EA}^{G_2}(n', v', e') \\ &\Rightarrow f'_{E_{EA}}(n, v, e) = (n', v', e'), \text{ because } s_{EA}^{G_2} \text{ is a bijection and therefore also an injection.} \end{aligned}$$

This implies altogether that $f'_{E_G}(v, e, n) = (v', e', n') = (f_V(v), f_E(e), n)$.

Part 2:

Similar to Part 1 replacing v by e and e by v .

Part 3:

Let $f'_{E_{NA}}(e, n, x) = (e', n', x')$ with $s_{NA}^{G_2}(e', n', x') = e'$ and $x = x'$ by compatibility with typing morphisms. Then we have:

$$e' = s_{NA}^{G_2}(e', n', x') = s_{NA}^{G_2}(f'_{E_{NA}}(e, n, x)) = f'_{V_G}(s_{NA}^{G_1}(e, n, x)) = f'_{V_G}(e) = f_E(e)$$

$$\Rightarrow e' = f_E(e),$$

$n' = n$: Similar to Part 1.

This implies altogether that $f'_{E_{NA}}(e, n, x) = (e', n', x') = (f_E(e), n, x)$.

Part 4:

Let $f'_{E_{EA}}(n, v, e) = (n', v', e')$ with $s_{EA}^{G_2}(n', v', e') = (v', e', n')$. Then we have:

$n' = n$: Similar to Part 1,

$$v' = s_G^{G_2}(s_{EA}^{G_2}(n', v', e')) = s_G^{G_2}(v', e', n')$$

$$\stackrel{\text{Part 1}}{\Rightarrow} v' = f_V(v),$$

$$e' = t_G^{G_2}(s_{EA}^{G_2}(n', v', e')) = t_G^{G_2}(v', e', n')$$

$$\stackrel{\text{Part 1}}{\Rightarrow} e' = f_E(e).$$

This implies altogether that $f'_{E_{EA}}(n, v, e) = (n', v', e') = (n, f_V(v), f_E(e))$.

Part 5:

Similar to Part 4 replacing v by e and e by v . □

Lemma 30: (\mathcal{F}_{HG} Creates Morphisms [211], see page 137)

Consider \mathcal{M} -adhesive categories (**HyperGraphs**, \mathcal{M}_1) and (**AGraphs**_{HGTG}, \mathcal{M}_2), typed attributed graphs $\mathcal{F}_{HG}(G_1)$ and $\mathcal{F}_{HG}(G_2)$ as well as a morphism $f' : \mathcal{F}_{HG}(G_1) \rightarrow \mathcal{F}_{HG}(G_2)$ that is compatible with typing morphisms. Then the \mathcal{M} -functor $\mathcal{F}_{HG} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{HGTG}$ creates a unique morphism $f : G_1 \rightarrow G_2$ such that $\mathcal{F}_{HG}(f) = f'$ or formally written:

$$\exists! f : G_1 \rightarrow G_2. \mathcal{F}_{HG}(f) = f'.$$

Proof.

- **Construction of $f : G_1 \rightarrow G_2$:**

Consider a morphism $f' : \mathcal{F}_{HG}(G_1) \rightarrow \mathcal{F}_{HG}(G_2)$ with $\mathcal{F}_{HG}(G_i) = (V_G^{G_i}, V_D^{G_i} = \mathbb{N}, E_G^{G_i}, E_{NA}^{G_i}, E_{EA}^{G_i}, (s_j^{G_i}, t_j^{G_i})_{j \in \{G, NA, EA\}})$ for $i \in \{1, 2\}$ given by $f' = (f'_{V_G}, f'_{V_D}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}})$, where

$$f'_{V_G} : V_G^{G_1} \rightarrow V_G^{G_2} \text{ with } V_G^{G_i} = V_{G_i} \uplus E_{G_i} \text{ for } i \in \{1, 2\},$$

$$f'_{V_D} : \mathbb{N} \rightarrow \mathbb{N} \text{ with } f'_{V_D} = id_{\mathbb{N}},$$

$$f'_{E_G} : E_G^{G_1} \rightarrow E_G^{G_2} \text{ with } E_G^{G_i} = E_{n2e}^{G_i} \uplus E_{e2n}^{G_i} \text{ for } i \in \{1, 2\},$$

$$f'_{E_{NA}} : E_{NA}^{G_1} \rightarrow E_{NA}^{G_2} \text{ with } E_{NA}^{G_i} = E_{in}^{G_i} \uplus E_{out}^{G_i} \text{ for } i \in \{1, 2\},$$

$$f'_{E_{EA}} : E_{EA}^{G_1} \rightarrow E_{EA}^{G_2} \text{ with } E_{EA}^{G_i} = E_s^{G_i} \uplus E_t^{G_i} \text{ for } i \in \{1, 2\}.$$

Define $f : G_1 \rightarrow G_2$ with $G_j = (V_{G_j}, E_{G_j}, s_{G_j}, t_{G_j})$ for $j \in \{1, 2\}$ by $f = (f_V : V_{G_1} \rightarrow V_{G_2}, f_E : E_{G_1} \rightarrow E_{G_2})$ with

$$f_V(v) = f'_{V_G}(v) \text{ for } v \in V_{G_1} \subseteq V_G^{G_1},$$

$$f_E(e) = f'_{V_G}(e) \text{ for } e \in E_{G_1} \subseteq V_G^{G_1}.$$

- **Well-definedness of $f : G_1 \rightarrow G_2$:**

We have to show the following three steps:

1. $f_V(v) = f'_{V_G}(v) \in V_{G_2}$ for $v \in V_{G_1}$, $f_E(e) = f'_{V_G}(e) \in E_{G_2}$ for $e \in E_{G_1}$,
2. f is a hypergraph morphism, i.e., square (1) below commutes, and
3. $\mathcal{F}_{HG}(f) = f'$.

Then the following holds:

1. To show: $f_V(v) = f'_{V_G}(v) \in V_{G_2}$ for $v \in V_{G_1}$.
 - a) $f'_{V_G}(v) \in V_{G_2} = V_{G_2} \uplus E_{G_2}$ by construction. type-compatibility of f' given by assumption implies that $(type_{V_G}^{G_2} \circ f'_{V_G})(v) = type_{V_G}^{G_1}(v) = \text{Node}$ using $v \in V_{G_1}$. This implies that $f_V(v) = f'_{V_G}(v) \in V_{G_2}$ using $type_{V_G}^{G_2}(f'_{V_G}(v)) = \text{Node}$ and $(type_{V_G}^{G_2})^{-1}(\text{Node}) = V_{G_2}$.
 - b) $f_E(e) = f'_{V_G}(e) \in E_{G_2}$ for $e \in E_{G_1}$: similarly to the proof above.
2. To show: Square (1) commutes, i.e.,
 - a) $\forall v \in V_{G_1}, \forall e \in E_{G_1}, \forall n \leq |s_{G_1}(e)|. (s_{G_1}^n(e) = v) \Rightarrow (s_{G_2}^n(f_E(e)) = f_V(v))$,
 - b) $\forall v \in V_{G_1}, \forall e \in E_{G_1}, \forall n \leq |t_{G_1}(e)|. (t_{G_1}^n(e) = v) \Rightarrow (t_{G_2}^n(f_E(e)) = f_V(v))$.

$$\begin{array}{ccc}
 E_{G_1} & \xrightarrow{s_1} & V_{G_1}^* \\
 f_E \downarrow & (1) & \downarrow f'_V \\
 E_{G_2} & \xrightarrow{s_2} & V_{G_2}^* \\
 & t_2 &
 \end{array}$$

We have the following:

(a)

$$\begin{aligned}
 s_{G_1}^n(e) = v &\Leftrightarrow (v, e, n) \in E_{n2e}^{G_1} \\
 &\Rightarrow f'_{E_G}(v, e, n) \in E_{n2e}^{G_2} \\
 &\stackrel{(*)}{\Rightarrow} (f_V(v), f_E(e), n) \in E_{n2e}^{G_2} \Leftrightarrow s_{G_2}^n(f_E(e)) = f_V(v) \\
 &\stackrel{\text{Lem. 2}}{\Rightarrow} f = (f_V, f_E) \text{ is a hypergraph morphism.}
 \end{aligned}$$

Where the step $(*)$ holds since we have the following:

$f'_{E_G}(v, e, n) = (f_V(v), f_E(e), n)$, because f'_{E_G}, f'_{V_G} are compatible with $s_G^{G_i}, t_G^{G_i}$ for $i \in \{1, 2\}$ (see the diagram below) since f' is a typed attributed graph morphism implying the following:

$$\begin{aligned}
 (s_G^{G_2} \circ f'_{E_G})(v, e, n) &= (f'_{V_G} \circ s_G^{G_1})(v, e, n) \\
 &= f'_{V_G}(s_G^{G_1}(v, e, n)) = f'_{V_G}(v) = f_V(v) \text{ and} \\
 (t_G^{G_2} \circ f'_{E_G})(v, e, n) &= (f'_{V_G} \circ t_G^{G_1})(v, e, n) \\
 &= f'_{V_G}(t_G^{G_1}(v, e, n)) = f'_{V_G}(e) = f_E(e)
 \end{aligned}$$

$$\begin{array}{ccc}
 E_G^{G_1} & \xrightarrow{s_G^{G_1}} & V_G^{G_1} \\
 f'_{E_G} \downarrow & t_G^{G_1} & \downarrow f'_{V_G} \\
 E_G^{G_2} & \xrightarrow{s_G^{G_2}} & V_G^{G_2} \\
 & t_G^{G_2} &
 \end{array}$$

(b) Similarly to the proof above.

3. To show: $\mathcal{F}_{\text{HG}}(f) = f'$.

Let $\mathcal{F}_{\text{HG}}(f) = f'' = (f''_{V_G}, f''_{V_D}, f''_{E_G}, f''_{E_{NA}}, f''_{E_{EA}})$. This implies that $f''_{V_G} \stackrel{\text{Def. 63}}{=} f_V \uplus f_E$ and $f''_{V_D} = \text{id}_{\mathbb{N}}$. But by construction above we have that:

$$\begin{aligned} f'_{V_G} &= f_V \uplus f_E \\ \Rightarrow f'_{V_G} &= f''_{V_G} \wedge f'_{V_D} = f''_{V_D} = \text{id}_{\mathbb{N}} \\ \stackrel{\text{Lem. 29}}{\Rightarrow} f' &= f'' \\ \Rightarrow \mathcal{F}_{\text{HG}}(f) &= f'' = f' \end{aligned}$$

• **Uniqueness of $f : G_1 \rightarrow G_2$:**

Let $\mathcal{F}_{\text{HG}}(f) = f'$. Assume that we have another morphism $g : G_1 \rightarrow G_2$ with $\mathcal{F}_{\text{HG}}(g) = f'$.

We have to show that $f = g$.

Since $\mathcal{F}_{\text{HG}}(f) = f' = \mathcal{F}_{\text{HG}}(g)$ and using Lemma 29 we have the following two cases:

Case 1: The argument of both morphisms is some node.

$$\begin{aligned} \mathcal{F}_{\text{HG}}(f) &= \mathcal{F}_{\text{HG}}(g) \\ \Rightarrow \forall v \in V_G^{G_1}. \mathcal{F}_{\text{HG}}(f)_{V_G}(v) &= \mathcal{F}_{\text{HG}}(g)_{V_G}(v) \\ \Rightarrow \forall v \in V_{G_1}. f_V(v) &= g_V(v) \\ \Rightarrow f_V &= g_V \end{aligned}$$

Case 2: The argument of both morphisms is some edge.

$$\begin{aligned} \mathcal{F}_{\text{HG}}(f) &= \mathcal{F}_{\text{HG}}(g) \\ \Rightarrow \forall e \in V_G^{G_1}. \mathcal{F}_{\text{HG}}(f)_{V_G}(e) &= \mathcal{F}_{\text{HG}}(g)_{V_G}(e) \\ \Rightarrow \forall e \in E_{G_1}. f_E(e) &= g_E(e) \\ \Rightarrow f_E &= g_E \end{aligned}$$

□

Lemma 31: (\mathcal{F}_{HG} Creates Injective Morphisms [213], see page 137)

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, typed attributed graphs $\mathcal{F}_{\text{HG}}(G_1), \mathcal{F}_{\text{HG}}(G_2)$, and an injective typed attributed graph morphism $f' : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ that is compatible with typing morphisms. Then the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ creates a unique injective hypergraph morphism $f : G_1 \rightarrow G_2$ such that $\mathcal{F}_{\text{HG}}(f) = f'$ or formally written:

$$\exists! f : G_1 \rightarrow G_2 \text{ in } \mathcal{M}_1. \mathcal{F}_{\text{HG}}(f) = f'.$$

Proof.

Consider an injective typed attributed graph morphism $f' : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G_2)$ with $\mathcal{F}_{\text{HG}}(G_i) = (V_G^{G_i}, V_D^{G_i} = \mathbb{N}, E_G^{G_i}, E_{NA}^{G_i}, E_{EA}^{G_i}, (s_j^{G_i}, t_j^{G_i})_{j \in \{G, NA, EA\}})$ for $i \in \{1, 2\}$ given by $f' = (f'_{V_G}, f'_{V_D}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}})$ with injective components

$$\begin{aligned} f'_{V_G} : V_G^{G_1} &\rightarrow V_G^{G_2} \text{ with } V_G^{G_i} = V_{G_i} \uplus E_{G_i} \text{ for } i \in \{1, 2\}, \\ f'_{V_D} : \mathbb{N} &\rightarrow \mathbb{N} \text{ with } f'_{V_D} = \text{id}_{\mathbb{N}}, \\ f'_{E_G} : E_G^{G_1} &\rightarrow E_G^{G_2} \text{ with } E_G^{G_i} = E_{n2e}^{G_i} \uplus E_{e2n}^{G_i} \text{ for } i \in \{1, 2\}, \end{aligned}$$

$$\begin{aligned} f'_{E_{NA}} : E_{NA}^{G_1} &\rightarrow E_{NA}^{G_2} \text{ with } E_{NA}^{G_i} = E_{in}^{G_i} \uplus E_{out}^{G_i} \text{ for } i \in \{1, 2\}, \\ f'_{E_{EA}} : E_{EA}^{G_1} &\rightarrow E_{EA}^{G_2} \text{ with } E_{EA}^{G_i} = E_s^{G_i} \uplus E_t^{G_i} \text{ for } i \in \{1, 2\}. \end{aligned}$$

Define the corresponding hypergraph morphism $f : G_1 \rightarrow G_2$ with $G_j = (V_{G_j}, E_{G_j}, s_{G_j}, t_{G_j})$ for $j \in \{1, 2\}$ according to Lemma 30 by $f = (f_V : V_{G_1} \rightarrow V_{G_2}, f_E : E_{G_1} \rightarrow E_{G_2})$ with

$$\begin{aligned} f_V(v) &= f'_{V_G}(v) \text{ for } v \in V_{G_1} \subseteq V_G^{G_1}, \\ f_E(e) &= f'_{V_G}(e) \text{ for } e \in E_{G_1} \subseteq V_G^{G_1}. \end{aligned}$$

Well-definedness of $f : G_1 \rightarrow G_2$ as well as the uniqueness of hypergraph morphism creation follow from Lemma 30.

It remains to show that $f = (f_V : V_{G_1} \rightarrow V_{G_2}, f_E : E_{G_1} \rightarrow E_{G_2})$ is an injective hypergraph morphism, i.e., f_V and f_E are injective.

- $f_V : V_{G_1} \rightarrow V_{G_2}$ **is injective:**

Fix $v_1, v_2 \in V_{G_1}$. It holds the following:

$$\begin{aligned} f_V(v_1) &= f_V(v_2) \\ \stackrel{\text{Def. } f'_{V_G}}{\Rightarrow} f'_{V_G}(v_1) &= f'_{V_G}(v_2) \\ f'_{V_G} \stackrel{\text{inj.}}{\Rightarrow} v_1 &= v_2 \end{aligned}$$

- $f_E : E_{G_1} \rightarrow E_{G_2}$ **is injective:**

The proof for this case works analogously to the proof of the case above.

□

Lemma 32: (Application of \mathcal{M} -Functor \mathcal{F}_{HG} to a Hypergraph Boundary Object B over a Morphism $f : L \rightarrow G$, see page 138)

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{HGTG}, \mathcal{M}_2)$, a boundary object $B = (V_B, E_B, s_B, t_B)$ over a given morphism $f : L \rightarrow G$ in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ constructed according to Lemma 5, and the \mathcal{M} -functor $\mathcal{F}_{HG} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{HGTG}$ from Definition 63. Then the application of \mathcal{F}_{HG} to the boundary object B results in the following typed attributed graph:

$$\begin{aligned} \mathcal{F}_{HG}(B) &= B^* = ((B_0^*, NAT), type^{B^*}) \text{ with} \\ B_0^* &= (V_G^{B_0^*}, V_D^{B_0^*} = \mathbb{N}, E_G^{B_0^*}, E_{NA}^{B_0^*}, E_{EA}^{B_0^*}, (s_j^{B_0^*}, t_j^{B_0^*})_{j \in \{G, NA, EA\}}) \text{ where} \\ V_G^{B_0^*} &= V_B \uplus E_B = IP_V \cup IP_{VE} \cup DP_V \cup IP_E, \\ E_G^{B_0^*} &= E_{n2e}^{B_0^*} \uplus E_{e2n}^{B_0^*} \text{ with} \\ E_{n2e}^{B_0^*} &= \{((v, e), n) \in (V_B \times E_B) \times \mathbb{N} \mid s_B^n(e) = v\}, \\ E_{e2n}^{B_0^*} &= \{((e, v), n) \in (E_B \times V_B) \times \mathbb{N} \mid t_B^n(e) = v\}, \\ E_{NA}^{B_0^*} &= E_{in}^{B_0^*} \uplus E_{out}^{B_0^*} \text{ with} \\ E_{in}^{B_0^*} &= \{(e, n, in) \mid (e, n) \in E_B \times \mathbb{N} \wedge |s_B(e)| = n\}, \\ E_{out}^{B_0^*} &= \{(e, n, out) \mid (e, n) \in E_B \times \mathbb{N} \wedge |t_B(e)| = n\}, \\ E_{EA}^{B_0^*} &= E_s^{B_0^*} \uplus E_t^{B_0^*} \text{ with} \\ E_s^{B_0^*} &= \{(n, (v, e)) \in \mathbb{N} \times (V_B \times E_B) \mid s_B^n(e) = v\}, \end{aligned}$$

$$\begin{aligned}
E_t^{B_0^*} &= \{(n, (e, v)) \in \mathbb{N} \times (E_B \times V_B) \mid t_B^n(e) = v\}, \\
s_G^{B_0^*} : E_G^{B_0^*} &\rightarrow V_G^{B_0^*} \text{ defined by } s_G^{B_0^*}((x, y), n) = x, \\
t_G^{B_0^*} : E_G^{B_0^*} &\rightarrow V_G^{B_0^*} \text{ defined by } t_G^{B_0^*}((x, y), n) = y, \\
s_{NA}^{B_0^*} : E_{NA}^{B_0^*} &\rightarrow V_G^{B_0^*} \text{ defined by } s_{NA}^{B_0^*}(e, n, x) = e, \\
t_{NA}^{B_0^*} : E_{NA}^{B_0^*} &\rightarrow \mathbb{N} \text{ defined by } t_{NA}^{B_0^*}(e, n, x) = n, \\
s_{EA}^{B_0^*} : E_{EA}^{B_0^*} &\rightarrow E_G^{B_0^*} \text{ defined by } s_{EA}^{B_0^*}(n, (x, y)) = ((x, y), n), \\
t_{EA}^{B_0^*} : E_{EA}^{B_0^*} &\rightarrow \mathbb{N} \text{ defined by } t_{EA}^{B_0^*}(n, (x, y)) = n,
\end{aligned}$$

and $\mathcal{F}_{HG}(b) : \mathcal{F}_{HG}(B) \rightarrow \mathcal{F}_{HG}(L)$ is an inclusion.

$$\begin{array}{ccc}
B^* = \mathcal{F}_{HG}(B) & \xrightarrow{\mathcal{F}_{HG}(b)} & \mathcal{F}_{HG}(L) \\
& & \downarrow \mathcal{F}_{HG}(f) \\
& & \mathcal{F}_{HG}(G)
\end{array}$$

Proof.

Applying Definition 63 to the boundary object $B = (V_B, E_B, s_B, t_B)$ over a morphism f constructed according to Lemma 5, we directly get $\mathcal{F}_{HG}(B)$ as given above. Moreover, by Lemma 27, we have for the inclusion $b : B \rightarrow L$ that also $\mathcal{F}_{HG}(b) : \mathcal{F}_{HG}(B) \rightarrow \mathcal{F}_{HG}(L)$ is an inclusion. \square

Lemma 33: (Boundary Object in $(\mathbf{AGraphs}_{HG\mathcal{T}G}, \mathcal{M}_2)$ over a Morphism $f' : L' \rightarrow G'$, see page 139)

Consider a typed attributed graph morphism $f' : L' \rightarrow G'$. The boundary object B' is given by $B' = ((B'_0, NAT), type^{B'})$ with the boundary points

$$B'_0 = (V_G^{B'_0}, V_D^{B'_0} = \mathbb{N}, E_G^{B'_0}, E_{NA}^{B'_0}, E_{EA}^{B'_0}, (s_j^{B'_0}, t_j^{B'_0})_{j \in \{G, NA, EA\}}).$$

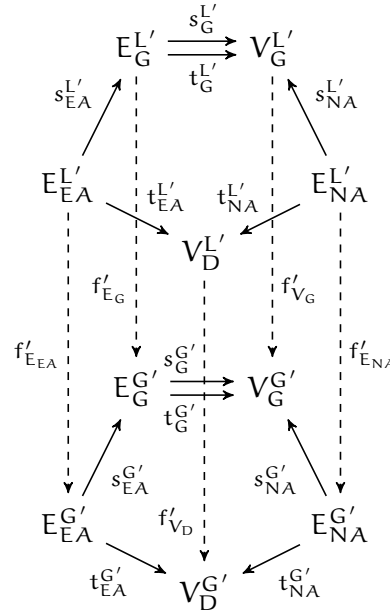
The components of B'_0 are given by the dangling and identification points as follows:

$$\begin{array}{ccc}
B' & \xrightarrow{b'} & L' \\
& & \downarrow f' \\
& & G'
\end{array}$$

$$\begin{aligned}
E_{NA}^{B'_0} &= IP_{E_{NA}} = \{a \in E_{NA}^{L'} \mid \exists a' \neq a. a' \in E_{NA}^{L'} \wedge f'_{E_{NA}}(a) = f'_{E_{NA}}(a')\}, \\
E_{EA}^{B'_0} &= IP_{E_{EA}} = \{a \in E_{EA}^{L'} \mid \exists a' \neq a. a' \in E_{EA}^{L'} \wedge f'_{E_{EA}}(a) = f'_{E_{EA}}(a')\}, \\
E_G^{B'_0} &= DP_{E_G} \cup IP_{E_G} \cup s_{EA}^{L'}(IP_{E_{EA}}) \text{ with} \\
DP_{E_G} &= \{a \in E_G^{L'} \mid \exists a' \in E_{EA}^{G'} \setminus f'_{E_{EA}}(E_{EA}^{L'}). f'_{E_G}(a) = s_{EA}^{G'}(a')\}, \\
IP_{E_G} &= \{a \in E_G^{L'} \mid \exists a' \neq a. a' \in E_G^{L'} \wedge f'_{E_G}(a) = f'_{E_G}(a')\}, \\
V_G^{B'_0} &= DP_{V_G} \cup IP_{V_G} \cup s_G^{L'}(E_G^{B'_0}) \cup t_G^{L'}(E_G^{B'_0}) \cup s_{NA}^{L'}(IP_{E_{NA}}) \text{ with}
\end{aligned}$$

$$\begin{aligned}
\text{DP}_{V_G} &= \{a \in V_G^{L'} \mid [\exists a' \in E_{NA}^{G'} \setminus f'_{E_{NA}}(E_{NA}^{L'}), f'_{V_G}(a) = s_{NA}^{G'}(a')] \\
&\quad \vee [\exists a' \in E_G^{G'} \setminus f'_{E_G}(E_G^{L'}), f'_{V_G}(a) = s_G^{G'}(a') \vee f'_{V_G}(a) = t_G^{G'}(a')]\}, \\
\text{IP}_{V_G} &= \{a \in V_G^{L'} \mid \exists a' \neq a, a' \in V_G^{L'} \wedge f'_{V_G}(a) = f'_{V_G}(a')\}, \\
s_G^{B'_0}, t_G^{B'_0} : E_G^{B'_0} &\rightarrow V_G^{B'_0} \text{ are restrictions of } s_G^{L'}, t_G^{L'} : E_G^{L'} \rightarrow V_G^{L'}, \\
s_{NA}^{B'_0} : E_{NA}^{B'_0} &\rightarrow V_G^{B'_0} \text{ is a restriction of } s_{NA}^{L'} : E_{NA}^{L'} \rightarrow V_G^{L'}, \\
t_{NA}^{B'_0} : E_{NA}^{B'_0} &\rightarrow \mathbb{N} \text{ is a restriction of } t_{NA}^{L'} : E_{NA}^{L'} \rightarrow \mathbb{N}, \\
s_{EA}^{B'_0} : E_{EA}^{B'_0} &\rightarrow E_G^{B'_0} \text{ is a restriction of } s_{EA}^{L'} : E_{EA}^{L'} \rightarrow E_G^{L'}, \\
t_{EA}^{B'_0} : E_{EA}^{B'_0} &\rightarrow \mathbb{N} \text{ is a restriction of } t_{EA}^{L'} : E_{EA}^{L'} \rightarrow \mathbb{N},
\end{aligned}$$

and $b' : B' \rightarrow L'$ is an inclusion.



Proof.

The construction above coincides with the construction in Fact 6. The only difference is in constructing the smallest well-defined graph containing certain elements. In Fact 6 this is achieved by the intersection of possible extensions (called there B') contained in L' . Here, we execute a closure operation which adds the sources and targets of all contained attributes and edges such that the resulting graph is obtained. Both approaches are obviously equivalent. \square

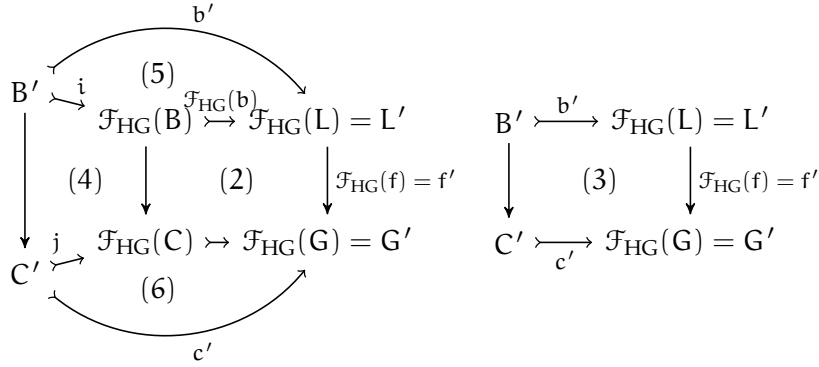
Lemma 34: (\mathcal{F}_{HG} Preserves Initial Pushouts [211], see page 140)

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{HGTG}, \mathcal{M}_2)$, \mathcal{M} -functor $\mathcal{F}_{HG} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{HGTG}$ from Definition 63, and let (1) be an initial pushout over $f : L \rightarrow G$ in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$. Then (2) is an initial pushout over $\mathcal{F}_{HG}(f) : \mathcal{F}_{HG}(L) \rightarrow \mathcal{F}_{HG}(G)$ in $(\mathbf{AGraphs}_{HGTG}, \mathcal{M}_2)$.

$$\begin{array}{ccc}
B & \xrightarrow{b} & L \\
\downarrow & (1) & \downarrow f \\
C & \longrightarrow & G
\end{array}
\quad \Rightarrow \quad
\begin{array}{ccc}
\mathcal{F}_{HG}(B) & \xrightarrow{\mathcal{F}_{HG}(b)} & \mathcal{F}_{HG}(L) \\
\downarrow & (2) & \downarrow \mathcal{F}_{HG}(f) \\
\mathcal{F}_{HG}(C) & \longrightarrow & \mathcal{F}_{HG}(G)
\end{array}$$

Proof.

We assume that (1) is an initial pushout in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ with boundary $B = (V_B, E_B, s_B = s_L, t_B = t_L) \subseteq L$ defined according to Lemma 5 and context $C = (V_C, E_C, s_C = s_G, t_C = t_G)$ defined according to Lemma 6.



Since \mathcal{F}_{HG} preserves pushouts along injective morphisms according to Lemma 28, we have that (2) is a pushout in $\mathbf{AGraphs}_{HGTG}$ with $\mathcal{F}_{HG}(b) \in \mathcal{M}_2$.

Now we construct the initial pushout (3) over the morphism $\mathcal{F}_{HG}(f) : \mathcal{F}_{HG}(L) \rightarrow \mathcal{F}_{HG}(G)$ in $(\mathbf{AGraphs}_{HGTG}, \mathcal{M}_2)$ with inclusion $b' : B' \rightarrow \mathcal{F}_{HG}(L)$ and boundary object $B' = ((B'_0, NAT), type^{B'})$ with the components of $B'_0 = (V_G^{B'_0}, V_D^{B'_0} = \mathbb{N}, E_G^{B'_0}, E_{NA}^{B'_0}, E_{EA}^{B'_0}, (s_j^{B'_0}, t_j^{B'_0})_{j \in \{G, NA, EA\}})$ defined according to Lemma 33.

Initiality of (3) implies unique morphisms $i : B' \rightarrow \mathcal{F}_{HG}(B)$ and $j : C' \rightarrow \mathcal{F}_{HG}(C)$ such that (4) is a pushout in $\mathbf{AGraphs}_{HGTG}$ and (5), (6) commute with $i, j \in \mathcal{M}_2$.

Consider now the typed attributed graph $\mathcal{F}_{HG}(B) = B^* = ((B_0^*, NAT), type^{B^*})$ with $B_0^* = (V_G^{B_0^*}, V_D^{B_0^*} = \mathbb{N}, E_G^{B_0^*}, E_{NA}^{B_0^*}, E_{EA}^{B_0^*}, (s_j^{B_0^*}, t_j^{B_0^*})_{j \in \{G, NA, EA\}})$ defined according to Lemma 32.

In the next step we have to show that $i : B' \rightarrow \mathcal{F}_{HG}(B)$ is surjective, i.e., $\mathcal{F}_{HG}(B) = B^* \subseteq B'$, i.e., $V_G^{B_0^*} \subseteq V_G^{B'_0} \wedge E_G^{B_0^*} \subseteq E_G^{B'_0} \wedge E_{NA}^{B_0^*} \subseteq E_{NA}^{B'_0} \wedge E_{EA}^{B_0^*} \subseteq E_{EA}^{B'_0}$.

1. To show: $V_G^{B_0^*} \subseteq V_G^{B'_0}$.

From constructions in Lemmas 32 and 33 we have:

$$\begin{aligned} V_G^{B_0^*} &= V_B \uplus E_B = IP_V \cup IP_{VE} \cup DP_V \cup IP_E \text{ and} \\ V_G^{B'_0} &= DP_{V_G} \cup IP_{V_G} \cup s_G^{L'}(E_G^{B'_0}) \cup t_G^{L'}(E_G^{B'_0}) \cup s_{NA}^{L'}(IP_{E_{NA}}) \end{aligned}$$

- a) Let $v \in DP_V$. Then we consider two cases:

- i. $v \in V_L$ and $\exists e \in E_G \setminus f_E(E_L)$. $f_V(v) \notin s_G(e)$, i.e., $\exists n \in \mathbb{N}$. $f_V(v) = s_G^n(e)$.

It remains to show: $v \in DP_{V_G} \subseteq V_G^{B'_0}$.

From $v \in V_L$ we get that $v \in V_G^{L'}$. Furthermore, let $a' = (f_V(v), e, n) \in E_{n2e}^{G'}$ because $f_V(v) = s_G^n(e)$. Then we have:

$$\begin{aligned} f'_{V_G}(v) &= f_V(v) = s_G^{G'}(a') \text{ and } e \in E_G \setminus f_E(E_L) \\ \Rightarrow a' &= (f_V(v), e, n) \notin f'_{E_G}(E_G^{L'}) \end{aligned}$$

$$\stackrel{\text{Lem. 33}}{\Rightarrow} v \in DP_{V_G} \subseteq V_G^{B'_0}.$$

- ii. $v \in V_L$ and $\exists e \in E_G \setminus f_E(E_L)$. $f_V(v) \notin t_G(e)$, i.e., $\exists n \in \mathbb{N}$. $f_V(v) = t_G^n(e)$:
Similar to the proof above.

b) Let $v \in \text{IP}_V$. Then we have:

$$v \in V_L \text{ and } \exists v' \neq v. v' \in V_L \wedge f_V(v) = f_V(v').$$

It remains to show: $v \in \text{IP}_{V_G} \subseteq V_G^{B'_0}$.

From $v, v' \in V_L$ we get that $v, v' \in V_G^{L'}$. Furthermore, we have:

$$f_V(v) = f_V(v') \Rightarrow f'_{V_G}(v) = f_V(v) = f_V(v') = f'_{V_G}(v')$$

$$\stackrel{\text{Lem. 33}}{\Rightarrow} v \in \text{IP}_{V_G} \subseteq V_G^{B'_0}.$$

c) Let $e \in \text{IP}_E$. Then we have:

$$e \in E_L \text{ and } \exists e' \neq e. e' \in E_L \wedge f_E(e) = f_E(e').$$

It remains to show: $e \in \text{IP}_{V_G} \subseteq V_G^{B'_0}$.

From $e, e' \in E_L$ we get that $e, e' \in V_G^{L'}$. Furthermore, we have:

$$f_E(e) = f_E(e') \Rightarrow f'_{V_G}(e) = f_E(e) = f_E(e') = f'_{V_G}(e')$$

$$\stackrel{\text{Lem. 33}}{\Rightarrow} e \in \text{IP}_{V_G} \subseteq V_G^{B'_0}.$$

d) Let $v \in \text{IP}_{V_E}$. Then we consider two cases:

i. $v \in V_L$ and $\exists e \in \text{IP}_E. v \in s_L(e)$, i.e., $\exists n \in \mathbb{N}. v = s_L^n(e)$.

It remains to show: $v \in s_G^{L'}(\text{IP}_{E_G}) \subseteq s_G^{L'}(E_G^{B'_0}) \subseteq V_G^{B'_0}$.

From $e \in \text{IP}_E$ we get that $\exists e' \neq e. e' \in E_L \wedge f_E(e) = f_E(e')$.

Let $\alpha = (v, e, n) \in E_{n2e}^{L'}$ and we construct $\alpha' \neq \alpha$ with $\alpha' = (v', e', n) \in E_{n2e}^{L'}$ s.t. $v' = s_L^n(e')$. Then we have:

$$\begin{aligned} f_V(v) &= f_V(s_L^n(e)) = [f_V^*(s_L(e))]_n = [s_G(f_E(e))]_n \\ &= [s_G(f_E(e'))]_n = [f_V^*(s_L(e'))]_n = f_V(s_L^n(e')) \\ &= f_V(v'), \text{ where } [\]_n \text{ denotes the } n\text{-th component of } [\] \end{aligned}$$

This implies:

$$\begin{aligned} f'_{E_G}(\alpha) &= f'_{E_G}(v, e, n) = (f_V(v), f_E(e), n) \\ &= (f_V(v'), f_E(e'), n) = f'_{E_G}(v', e', n) = f'_{E_G}(\alpha') \end{aligned}$$

$$\stackrel{\text{Lem. 33}}{\Rightarrow} \alpha \in \text{IP}_{E_G}$$

$$\Rightarrow v \in s_G^{L'}(\text{IP}_{E_G}) \text{ with } v = s_G^{L'}(\alpha)$$

$$\Rightarrow v \in s_G^{L'}(\text{IP}_{E_G}) \subseteq s_G^{L'}(E_G^{B'_0}) \subseteq V_G^{B'_0}.$$

$$\begin{array}{ccc} E_L & \xrightarrow{s_L} & V_L^* \\ f_E \downarrow & \begin{array}{c} t_L \\ = \\ s_G \end{array} & \downarrow f_V^* \\ E_G & \xrightarrow{s_G} & V_G^* \\ & t_G & \end{array}$$

ii. $v \in V_L$ and $\exists e \in \text{IP}_E. v \in t_L(e)$, i.e., $\exists n \in \mathbb{N}. v = t_L^n(e)$: Similar to the proof above.

2. To show: $E_G^{B'_0} \subseteq E_G^{B'_0}$.

From constructions in Lemmas 32 and 33 we have:

$$E_G^{B'_0} = E_{n2e}^{B'_0} \uplus E_{e2n}^{B'_0} \text{ and}$$

$$E_G^{B'_0} = \text{DP}_{E_G} \cup \text{IP}_{E_G} \cup s_{EA}^{L'}(\text{IP}_{E_{EA}})$$

a) Let $(v, e, n) \in E_{n2e}^{B_0^*}$. Then we have:

$$(v, e, n) \in (V_B \times E_B) \times \mathbb{N} \text{ with } s_B^n(e) = v = s_L^n(e).$$

It remains to show: $(v, e, n) \in IP_{E_G} \subseteq E_G^{B_0'}$.

From $e \in E_B = IP_E$ we get that $\exists e' \neq e$. $e' \in E_L \wedge f_E(e) = f_E(e')$.

Let $a = (v, e, n) \in E_{n2e}^{B_0^*}$ and we construct $a' \neq a$ with $a' = (v', e', n) \in E_{n2e}^{B_0'}$ s.t. $v' = s_L^n(e')$. Then we have:

$$\begin{aligned} f_V(v) &= f_V(s_L^n(e)) = [f_V^*(s_L(e))]_n = [s_G(f_E(e))]_n \\ &= [s_G(f_E(e'))]_n = [f_V^*(s_L(e'))]_n = f_V(s_L^n(e')) \\ &= f_V(v'), \text{ where } [\]_n \text{ denotes the } n\text{-th component of } [\] \end{aligned}$$

This implies:

$$\begin{aligned} f'_{E_G}(a) &= f'_{E_G}(v, e, n) = (f_V(v), f_E(e), n) \\ &= (f_V(v'), f_E(e'), n) = f'_{E_G}(v', e', n) = f'_{E_G}(a') \end{aligned}$$

$\xRightarrow{\text{Lem. 33}} a \in IP_{E_G} \subseteq E_G^{B_0'}$.

b) Let $(e, v, n) \in E_{e2n}^{B_0^*}$: Similar to the proof above.

3. To show: $E_{NA}^{B_0^*} \subseteq E_{NA}^{B_0'}$.

From constructions in Lemmas 32 and 33 we have:

$$\begin{aligned} E_{NA}^{B_0^*} &= E_{in}^{B_0^*} \uplus E_{out}^{B_0^*} \text{ and} \\ E_{NA}^{B_0'} &= IP_{E_{NA}} \end{aligned}$$

a) Let $(e, n, in) \in E_{in}^{B_0^*}$. Then we have:

$$(e, n, in) \in E_B \times \mathbb{N} \times \{in\} \text{ with } |s_B(e)| = n = |s_L(e)|.$$

It remains to show: $(e, n, in) \in IP_{E_{NA}} = E_{NA}^{B_0'}$.

From $e \in E_B = IP_E$ we get that $\exists e' \neq e$. $e' \in E_L \wedge f_E(e) = f_E(e')$.

Let $a = (e, n, in) \in E_{in}^{B_0^*}$ and we construct $a' \neq a$ with $a' = (e', n', in) \in E_{in}^{B_0'}$ s.t. $n' = |s_L(e')|$. Then we have:

$$\begin{aligned} n &= |s_L(e)| = |f_V^*(s_L(e))| = |s_G(f_E(e))| \\ &= |s_G(f_E(e'))| = |f_V^*(s_L(e'))| = |s_L(e')| \end{aligned}$$

This implies:

$$\begin{aligned} f'_{E_{NA}}(a) &= f'_{E_{NA}}(e, n, in) = (f_E(e), n, in) = (f_E(e), |s_L(e')|, in) \\ &= (f_E(e'), n', in) = f'_{E_{NA}}(e', n', in) = f'_{E_{NA}}(a') \end{aligned}$$

$\xRightarrow{\text{Lem. 33}} a \in IP_{E_{NA}} = E_{NA}^{B_0'}$.

b) Let $(e, n, out) \in E_{out}^{B_0^*}$: Similar to the proof above.

4. To show: $E_{EA}^{B_0^*} \subseteq E_{EA}^{B_0'}$.

From constructions in Lemmas 32 and 33 we have:

$$\begin{aligned} E_{EA}^{B_0^*} &= E_s^{B_0^*} \uplus E_t^{B_0^*} \text{ and} \\ E_{EA}^{B_0'} &= IP_{E_{EA}} \end{aligned}$$

a) Let $(n, v, e) \in E_s^{B^*}$. Then we have:

$$(n, v, e) \in \mathbb{N} \times (V_B \times E_B) \text{ with } s_B^n(e) = v = s_L^n(e).$$

It remains to show: $(n, v, e) \in IP_{E_{EA}} = E_{EA}^{B'_0}$.

From $e \in E_B = IP_E$ we get that $\exists e' \neq e$. $e' \in E_L \wedge f_E(e) = f_E(e')$.

Let $a = (n, v, e) \in E_s^{L'}$ and we construct $a' \neq a$ with $a' = (n, v', e') \in E_s^{L'}$ s.t. $v' = s_L^n(e')$. Then we have:

$$\begin{aligned} f_V(v) &= f_V(s_L^n(e)) = [f_V^*(s_L(e))]_n = [s_G(f_E(e))]_n \\ &= [s_G(f_E(e'))]_n = [f_V^*(s_L(e'))]_n = f_V(s_L^n(e')) \\ &= f_V(v'), \text{ where } [\]_n \text{ denotes the } n\text{-th component of } [\] \end{aligned}$$

This implies:

$$\begin{aligned} f'_{E_{EA}}(a) &= f'_{E_{EA}}(n, v, e) = (n, f_V(v), f_E(e)) \\ &= (n, f_V(v'), f_E(e')) = f'_{E_{EA}}(n, v', e') = f'_{E_{EA}}(a') \end{aligned}$$

$$\stackrel{\text{Lem. 33}}{\Rightarrow} a \in IP_{E_{EA}} = E_{EA}^{B'_0}.$$

b) Let $(n, e, v) \in E_t^{B^*}$: Similar to the proof above.

This concludes the part of surjectivity of $i : B' \rightarrow \mathcal{F}_{HG}(B)$. Now i is injective and surjective, so we get that i is an isomorphism. Since (4) is a pushout, also $j : C' \rightarrow \mathcal{F}_{HG}(C)$ is an isomorphism and hence (2) is isomorphic to (3). So we get that also (2) is an initial pushout over $\mathcal{F}_{HG}(f) : \mathcal{F}_{HG}(L) \rightarrow \mathcal{F}_{HG}(G)$. \square

Lemma 35: ($\mathcal{E} - \mathcal{M}$ -Factorization in HyperGraphs, see page 150)

The \mathcal{M} -adhesive category (**HyperGraphs**, \mathcal{M}_1) has an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization according to Definition 10 where \mathcal{M}_1 is the class of all injective hypergraph morphisms and \mathcal{E}_1 is the class of all surjective hypergraph morphisms.

Proof.

Consider hypergraphs $H_1 = (V_1, E_1, s_1 : E_1 \rightarrow V_1^*, t_1 : E_1 \rightarrow V_1^*)$, $H_2 = (V_2, E_2, s_2 : E_2 \rightarrow V_2^*, t_2 : E_2 \rightarrow V_2^*)$ and a hypergraph morphism $f : H_1 \rightarrow H_2$. We construct an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization $(f_1 : H_1 \rightarrow H_3, f_2 : H_3 \rightarrow H_2)$ of f given to the right with $H_3 = (V_3, E_3, s_3 : E_3 \rightarrow V_3^*, t_3 : E_3 \rightarrow V_3^*)$ componentwise in **Sets** for nodes and hyperedges as follows

$$\begin{array}{ccc} H_1 & \xrightarrow{f_1 = (f_{V_1}, f_{E_1}) \text{ surj.}} & H_3 \\ f \downarrow & \searrow & \uparrow \\ H_2 & \xleftarrow{f_2 = (f_{V_2}, f_{E_2}) \text{ inj.}} & H_3 \end{array}$$

$$\begin{array}{ccccc} E_1 & \xrightarrow{s_1} & V_1^* & & \\ \downarrow f_E & \searrow f_{E_1} & \downarrow f_{V_1}^* & & \\ & E_3 & \xrightarrow{s_3} & V_3^* & \\ & \downarrow f_{E_2} & \downarrow f_V^* & \swarrow f_{V_2}^* & \\ E_2 & \xrightarrow{s_2} & V_2^* & & \end{array} \quad \begin{array}{ccc} V_1 & & \\ \downarrow f_V & \searrow f_{V_1} \text{ surj.} & \\ & V_3 & \\ & \downarrow f_{V_2} \text{ inj.} & \\ & V_2 & \end{array}$$

where $f_{E_1} : E_1 \rightarrow E_3$ is surjective, $f_{E_2} : E_3 \rightarrow E_2$ is injective and the corresponding factorization diagram for the node component is given on the right side of the diagram above. For the construction of the source function s_3 consider the following diagram.

$$\begin{array}{ccc}
E_1 & \xrightarrow{s_1} & V_1^* \\
f_{E_1} \text{ surj.} \downarrow & & \downarrow f_{V_1}^* \\
E_3 & \xrightarrow{s_3} & V_3^* \\
f_{E_2} \text{ inj.} \downarrow & & \downarrow f_{V_2}^* \\
E_2 & \xrightarrow{s_2} & V_2^*
\end{array}$$

Since f_{E_1} is surjective and hence an epimorphism in **Sets**, we also know that f_{E_1} is a coequalizer in **Sets**. Moreover, by Definition 2.6 from [251, p. 171] we have that f_{E_1} is a regular epimorphism implying by Lemma 2.10 from [251, p. 171] that f_{E_1} is a strong epimorphism. Finally, by definition of a strong epimorphism (see Definition 2.1 from [251, p. 170]) we get that there is a unique source function $s_3 : E_3 \rightarrow V_3^*$ making the diagram below commute if $f_{V_2}^*$ is injective, which can be easily shown as follows:

$f_{V_2}^*$ is injective, i.e., $\forall w = x_1 \dots x_n, w' = x'_1 \dots x'_n \in V_3^*. (f_{V_2}^*(x_1 \dots x_n) = f_{V_2}^*(x'_1 \dots x'_n)) \Rightarrow (x_1 \dots x_n = x'_1 \dots x'_n)$.

Fix $w, w' \in V_3^*$ and assume that $f_{V_2}^*(x_1 \dots x_n) = f_{V_2}^*(x'_1 \dots x'_n)$. Then we have:

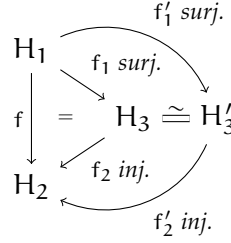
$$\begin{aligned}
f_{V_2}^*(x_1 \dots x_n) &= f_{V_2}^*(x'_1 \dots x'_n) \\
\stackrel{\text{Def. } f_{V_2}^*}{\Rightarrow} f_{V_2}(x_1) \dots f_{V_2}(x_n) &= f_{V_2}(x'_1) \dots f_{V_2}(x'_n) \\
\stackrel{f_{V_2} \text{ inj.}}{\Rightarrow} x_1 \dots x_n &= x'_1 \dots x'_n
\end{aligned}$$

$$\begin{array}{ccc}
E_1 & \xrightarrow{f_{E_1} \text{ surj.}} & E_3 \\
f_{V_1}^* \circ s_1 \downarrow & \begin{array}{c} = \\ \swarrow s_3 \\ = \end{array} & \downarrow s_2 \circ f_{E_2} \\
V_3^* & \xrightarrow{f_{V_2}^*} & V_2^*
\end{array}$$

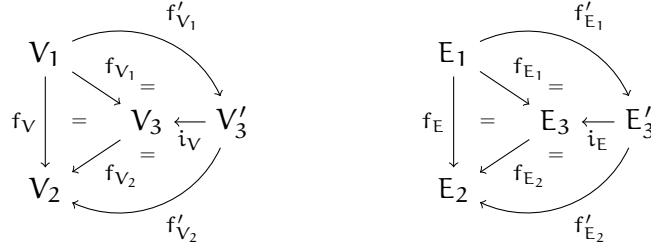
The unique target function $t_3 : E_3 \rightarrow V_3^*$ can be constructed similarly to the case of the source function considering the following diagrams:

$$\begin{array}{ccc}
E_1 & \xrightarrow{t_1} & V_1^* \\
f_{E_1} \text{ surj.} \downarrow & & \downarrow f_{V_1}^* \\
E_3 & \xrightarrow{t_3} & V_3^* \\
f_{E_2} \text{ inj.} \downarrow & & \downarrow f_{V_2}^* \\
E_2 & \xrightarrow{t_2} & V_2^*
\end{array}
\qquad
\begin{array}{ccc}
E_1 & \xrightarrow{f_{E_1} \text{ surj.}} & E_3 \\
f_{V_1}^* \circ t_1 \downarrow & \begin{array}{c} = \\ \swarrow t_3 \\ = \end{array} & \downarrow t_2 \circ f_{E_2} \\
V_3^* & \xrightarrow{f_{V_2}^*} & V_2^*
\end{array}$$

It remains to show that the hypergraph H_3 constructed in this way is unique up to isomorphism. For this reason assume that there are another hypergraph $H'_3 = (V'_3, E'_3, s'_3 : E'_3 \rightarrow V'_3, t'_3 : E'_3 \rightarrow V'_3)$ and morphisms $f'_1 : H_1 \rightarrow H'_3$ with $f'_1 = (f'_{V_1} : V_1 \rightarrow V'_3, f'_{E_1} : E_1 \rightarrow E'_3)$ in \mathcal{E}_1 , $f'_2 : H'_3 \rightarrow H_2$ with $f'_2 = (f'_{V_2} : V'_3 \rightarrow V_2, f'_{E_2} : E'_3 \rightarrow E_2)$ in \mathcal{M}_1 such that it holds $f = f'_2 \circ f'_1$. We have to show that $H'_3 \cong H_3$.



By the componentwise construction of H_3 , we know that there are isomorphisms $i_V : V'_3 \rightarrow V_3$, $i_E : E'_3 \rightarrow E_3$ in **Sets** such that $f_{V_1} = i_V \circ f'_{V_1}$, $f_{E_1} = i_E \circ f'_{E_1}$, $f'_{V_2} = f_{V_2} \circ i_V$ and $f'_{E_2} = f_{E_2} \circ i_E$ as given in the diagrams below.



Thus $i : H'_3 \rightarrow H_3$ with $i = (i_V, i_E)$ is a hypergraph isomorphism if it is compatible with the corresponding source and target functions, i.e.,

- $s_3 \circ i_E = i_V^* \circ s'_3$:

It holds the following:

$$\begin{aligned} s_3 \circ i_E \circ f'_{E_1} &\stackrel{f_{E_1} = i_E \circ f'_{E_1}}{=} s_3 \circ f_{E_1} \stackrel{(1) \text{ comm.}}{=} f_{V_1}^* \circ s_1 \\ &\stackrel{f_{V_1} = i_V \circ f'_{V_1}}{=} (i_V \circ f'_{V_1})^* \circ s_1 = i_V^* \circ f_{V_1}'^* \circ s_1 \stackrel{(2) \text{ comm.}}{=} i_V^* \circ s'_3 \circ f'_{E_1} \end{aligned}$$

Thus, we have that $s_3 \circ i_E \circ f'_{E_1} = i_V^* \circ s'_3 \circ f'_{E_1}$ and since f'_{E_1} is surjective by assumption, we get that $s_3 \circ i_E = i_V^* \circ s'_3$.

- $t_3 \circ i_E = i_V^* \circ t'_3$:

The proof for this case works analogously to the proof of the case above.

$$\begin{array}{ccc} \begin{array}{ccc} E'_3 & \xrightarrow{s'_3} & V_3'^* \\ i_E \downarrow & \xrightarrow[t'_3]{=} & \downarrow i_V^* \\ E_3 & \xrightarrow[t_3]{s_3} & V_3^* \end{array} & \begin{array}{ccc} E_1 & \xrightarrow{s_1} & V_1^* \\ f_{E_1} \downarrow & \xrightarrow[t_1]{(1)} & \downarrow f_{V_1}'^* \\ E_3 & \xrightarrow[t_3]{s_3} & V_3^* \end{array} & \begin{array}{ccc} E_1 & \xrightarrow{s_1} & V_1^* \\ f_{E_1}' \downarrow & \xrightarrow[t_1]{(2)} & \downarrow f_{V_1}'^* \\ E'_3 & \xrightarrow[t'_3]{s'_3} & V_3'^* \end{array} \end{array}$$

This implies that $i : H'_3 \rightarrow H_3$ is a hypergraph isomorphism and hence $H'_3 \cong H_3$. □

Lemma 36: (\mathcal{F}_{HG} Preserves Coproducts, see page 150)

Consider a hypergraph A , a family of hypergraphs $(A_j)_{j \in I}$, a family of hypergraph morphisms $(i_j : A_j \rightarrow A)_{j \in I}$, a coproduct $(A, (i_j)_{j \in I})$ of $(A_j)_{j \in I}$ in **HyperGraphs**, and the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Then $(\mathcal{F}_{\text{HG}}(A), (\mathcal{F}_{\text{HG}}(i_j))_{j \in I})$ is a coproduct of $(\mathcal{F}_{\text{HG}}(A_j))_{j \in I}$ in **AGraphs**_{HGTG}.

$$\begin{array}{ccc}
A_j & \xrightarrow{i_j} & A \\
& \searrow f_j & \downarrow f \\
& & B
\end{array}
\quad \xrightarrow{\mathcal{F}_{\text{HG}}} \quad
\begin{array}{ccc}
\mathcal{F}_{\text{HG}}(A_j) & \xrightarrow{\mathcal{F}_{\text{HG}}(i_j)} & \mathcal{F}_{\text{HG}}(A) \\
& \searrow \mathcal{F}_{\text{HG}}(f_j) & \downarrow \mathcal{F}_{\text{HG}}(f) \\
& & \mathcal{F}_{\text{HG}}(B)
\end{array}$$

Proof.

To prove the preservation of coproducts by \mathcal{F}_{HG} it is sufficient to show, that \mathcal{F}_{HG} preserves pushouts and initial objects, because pushouts over initial objects are coproducts. In Lemma 28 we have already shown, that \mathcal{F}_{HG} preserves pushouts. It remains to show that \mathcal{F}_{HG} preserves initial objects, i.e., for the initial hypergraph \emptyset , $\mathcal{F}_{\text{HG}}(\emptyset) = \emptyset^{\mathbb{N}}$ is initial in $\mathbf{AGraphs}_{\text{HGTG}}^{\mathbb{N}}$. This holds obviously according to the explanation in Section 5.1. Thus, \mathcal{F}_{HG} preserves coproducts. \square

Lemma 37: (\mathcal{F}_{HG} Preserves Surjective Morphisms [211], see page 150)

Consider two hypergraphs $H_1 = (V_1, E_1, s_1, t_1)$, $H_2 = (V_2, E_2, s_2, t_2)$, a surjective hypergraph morphism $f : H_1 \rightarrow H_2$ with $f = (f_V : V_1 \rightarrow V_2, f_E : E_1 \rightarrow E_2)$, and the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63. Then the corresponding typed attributed graph morphism $\mathcal{F}_{\text{HG}}(f) : \mathcal{F}_{\text{HG}}(H_1) \rightarrow \mathcal{F}_{\text{HG}}(H_2)$ with $\mathcal{F}_{\text{HG}}(f) = f' = (f'_{V_G} : V_G^{G_1} \rightarrow V_G^{G_2}, f'_{V_D} : \mathbb{N} \rightarrow \mathbb{N}, f'_{E_G} : E_G^{G_1} \rightarrow E_G^{G_2}, f'_{E_{NA}} : E_{NA}^{G_1} \rightarrow E_{NA}^{G_2}, f'_{E_{EA}} : E_{EA}^{G_1} \rightarrow E_{EA}^{G_2})$ is also surjective.

Proof.

Consider surjective morphisms f_V and f_E , i.e., $\forall v' \in V_{G_2}. \exists v \in V_{G_1}. v' = f_V(v)$ and $\forall e' \in E_{G_2}. \exists e \in E_{G_1}. e' = f_E(e)$.

1. To show: f'_{V_G} is surjective, i.e., $\forall x' \in V_G^{G_2}. \exists x \in V_G^{G_1}. (x' = f_V(x)) \vee (x' = f_E(x))$.
 $f'_{V_G} : V_G^{G_1} \rightarrow V_G^{G_2}$ with $f'_{V_G} = f_V \uplus f_E$ for $V_G^{G_i} = V_{G_i} \uplus E_{G_i}$ where $i \in \{1, 2\}$ is surjective, because the components f_V and f_E are surjective.
2. To show: f'_{V_D} is surjective, i.e., $\forall n' \in \mathbb{N}. \exists n \in \mathbb{N}. n' = \text{id}_{\mathbb{N}}(n)$.
 f'_{V_D} is obviously surjective.
3. To show: f'_{E_G} is surjective.
 $f'_{E_G} : E_G^{G_1} \rightarrow E_G^{G_2}$ for $E_G^{G_i} = E_{n2e}^{G_i} \uplus E_{e2n}^{G_i}$ with $i \in \{1, 2\}$,
where $E_{n2e}^{G_i} = \{(v, e, n) \in (V_{G_i} \times E_{G_i}) \times \mathbb{N} \mid s_{G_i}^n(e) = v\}$
and $E_{e2n}^{G_i} = \{(e, v, n) \in (E_{G_i} \times V_{G_i}) \times \mathbb{N} \mid t_{G_i}^n(e) = v\}$.

a) Let $(v', e', n') \in E_{n2e}^{G_2}$ with $s_{G_2}^{n'}(e') = v'$.

It remains to show: $\exists (v, e, n) \in E_{n2e}^{G_1}. f'_{E_G}(v, e, n) = (v', e', n')$.

Because of surjectivity of f_E holds: $\exists e \in E_{G_1}. e' = f_E(e)$ and let $v = s_{G_1}^{n'}(e)$. Then we get that $(v, e, n') \in E_{n2e}^{G_1}$, because $s_{G_1}^{n'}(e) = v$. Furthermore, we have:

$$\begin{aligned}
v' &= s_{G_2}^{n'}(e') \stackrel{f_E \text{ surj.}}{=} s_{G_2}^{n'}(f_E(e)) = [s_{G_2}(f_E(e))]_{n'} = [f_V^*(s_{G_1}(e))]_{n'} \\
&= f_V(s_{G_1}^{n'}(e)) = f_V(v), \text{ where } [\]_{n'} \text{ denotes the } n\text{-th component of } [\]
\end{aligned}$$

This implies that $f'_{E_G}(v, e, n') = (f_V(v), f_E(e), n') = (v', e', n')$.

$$\begin{array}{ccc}
E_{G_1} & \xrightarrow{s_{G_1}} & V_{G_1}^* \\
f_E \downarrow & \begin{array}{c} t_{G_1} \\ = \\ t_{G_2} \end{array} & \downarrow f_V^* \\
E_{G_2} & \xrightarrow{s_{G_2}} & V_{G_2}^*
\end{array}$$

b) Let $(e', v', n') \in E_{e2n}^{G_2}$ with $t_{G_2}^{n'}(e') = v'$: The proof for this case is similar to the proof above.

4. To show: $f'_{E_{NA}}$ is surjective.

$f'_{E_{NA}} : E_{NA}^{G_1} \rightarrow E_{NA}^{G_2}$ for $E_{NA}^{G_i} = E_{in}^{G_i} \uplus E_{out}^{G_i}$ with $i \in \{1, 2\}$,

where $E_{in}^{G_i} = \{(e, n, in) \mid (e, n) \in E_{G_i} \times \mathbb{N} \wedge |s_{G_i}(e)| = n\}$

and $E_{out}^{G_i} = \{(e, n, out) \mid (e, n) \in E_{G_i} \times \mathbb{N} \wedge |t_{G_i}(e)| = n\}$.

a) Let $(e', n', in) \in E_{in}^{G_2}$ with $|s_{G_2}(e')| = n'$.

It remains to show: $\exists (e, n, in) \in E_{in}^{G_1}. f'_{E_{NA}}(e, n, in) = (e', n', in)$.

Because of surjectivity of f_E holds: $\exists e \in E_{G_1}. e' = f_E(e)$ and let $n = |s_{G_1}(e)|$. Then we get that $(e, n, in) \in E_{in}^{G_1}$, because $|s_{G_1}(e)| = n$. Furthermore, we have:

$$\begin{aligned} n' &= |s_{G_2}(e')| \stackrel{f_E \text{ surj.}}{=} |s_{G_2}(f_E(e))| \\ &= |f_V^*(s_{G_1}(e))| = |s_{G_1}(e)| = n \end{aligned}$$

This implies that $f'_{E_{NA}}(e, n, in) = (f_E(e), n, in) = (e', n', in)$.

b) Let $(e', n', out) \in E_{out}^{G_2}$ with $|t_{G_2}(e')| = n'$: The proof for this case is similar to the proof above.

5. To show: $f'_{E_{EA}}$ is surjective.

$f'_{E_{EA}} : E_{EA}^{G_1} \rightarrow E_{EA}^{G_2}$ for $E_{EA}^{G_i} = E_s^{G_i} \uplus E_t^{G_i}$ with $i \in \{1, 2\}$,

where $E_s^{G_i} = \{(n, v, e) \in \mathbb{N} \times (V_{G_i} \times E_{G_i}) \mid s_{G_i}^n(e) = v\}$

and $E_t^{G_i} = \{(n, e, v) \in \mathbb{N} \times (E_{G_i} \times V_{G_i}) \mid t_{G_i}^n(e) = v\}$.

a) Let $(n', v', e') \in E_s^{G_2}$ with $s_{G_2}^{n'}(e') = v'$.

It remains to show: $\exists (n, v, e) \in E_s^{G_1}. f'_{E_{EA}}(n, v, e) = (n', v', e')$.

Because of surjectivity of f_E holds: $\exists e \in E_{G_1}. e' = f_E(e)$ and let $v = s_{G_1}^{n'}(e)$. Then we get that $(n', v, e) \in E_s^{G_1}$, because $s_{G_1}^{n'}(e) = v$. Furthermore, we have:

$$\begin{aligned} v' &= s_{G_2}^{n'}(e') \stackrel{f_E \text{ surj.}}{=} s_{G_2}^{n'}(f_E(e)) = [s_{G_2}(f_E(e))]_{n'} = [f_V^*(s_{G_1}(e))]_{n'} \\ &= f_V(s_{G_1}^{n'}(e)) = f_V(v), \text{ where } [\]_n \text{ denotes the } n\text{-th component of } [\] \end{aligned}$$

This implies that $f'_{E_{EA}}(n', v, e) = (n', f_V(v), f_E(e)) = (n', v', e')$.

b) Let $(n', e', v') \in E_t^{G_2}$ with $t_{G_2}^{n'}(e') = v'$: The proof for this case is similar to the proof above.

□

Lemma 38: (\mathcal{F}_{HG} is Compatible with Pair Factorization [211], see page 151)

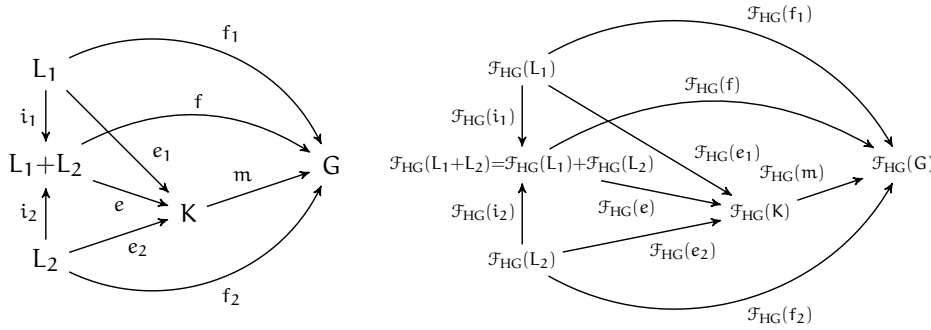
Consider \mathcal{M} -adhesive transformation systems $AS_1 = (\mathbf{HyperGraphs}, \mathcal{M}_1, P)$, $AS_2 = (\mathbf{AGraphs}_{HGTG}, \mathcal{M}_2, \mathcal{F}_{HG}(P))$, and the \mathcal{M} -functor $\mathcal{F}_{HG} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{HGTG}$ from Definition 63. Then \mathcal{F}_{HG} is compatible with pair factorization.

Proof.

According to Definition 51 and as already discussed before, we have to show that the categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ and $(\mathbf{AGraphs}_{HGTG}, \mathcal{M}_2)$ have $\mathcal{E}' - \mathcal{M}$ pair factorizations and for each $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization $(f_1 = m \circ e_1, f_2 = m \circ e_2)$ in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ also $(\mathcal{F}_{HG}(f_1) = \mathcal{F}_{HG}(m) \circ \mathcal{F}_{HG}(e_1), \mathcal{F}_{HG}(f_2) = \mathcal{F}_{HG}(m) \circ \mathcal{F}_{HG}(e_2))$ is an $\mathcal{E}'_2 - \mathcal{M}_2$ pair factorization in $(\mathbf{AGraphs}_{HGTG}, \mathcal{M}_2)$ (see the diagram below). According to Lemma 1, an $\mathcal{E}'_1 - \mathcal{M}_1$

pair factorization of $(f_1 : L_1 \rightarrow G, f_2 : L_2 \rightarrow G)$ based on $\mathcal{E}_1 - \mathcal{M}_1$ -factorization and coproducts is given by $(f_1 = m \circ e_1, f_2 = m \circ e_2)$, where $f : L_1 + L_2 \rightarrow G$ is the induced morphism of $f_i : L_i \rightarrow G$ for $i \in \{1, 2\}$, $f = m \circ e$ is an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization according to Definition 10 and $e_1 = e \circ i_1$, $e_2 = e \circ i_2$ are defined via the coproduct morphisms $i_1 : L_1 \rightarrow L_1 + L_2$ and $i_2 : L_2 \rightarrow L_1 + L_2$ in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$. Similar we obtain a pair factorization in $(\mathbf{AGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2)$. Lemma 1 is applicable to the categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ and $(\mathbf{AGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2)$, because these categories have $\mathcal{E}_i - \mathcal{M}_i$ -factorizations for $i \in \{1, 2\}$ according to Lemma 35 and [88] as well as coproducts constructed componentwise as disjoint union, because the empty hypergraph \emptyset resp. almost empty typed attributed graph $\mathcal{F}_{\mathbf{HG}}(\emptyset)$ are initial in the categories $\mathbf{HyperGraphs}$ resp. $\mathbf{AGraphs}_{\mathbf{HGTG}}$ and we have pushouts in both categories.

In order to show that $\mathcal{F}_{\mathbf{HG}}$ is compatible with pair factorization it remains to show that $\mathcal{F}_{\mathbf{HG}}$ preserves pair factorization, i.e., for each pair factorization $(f_1 = m \circ e_1, f_2 = m \circ e_2)$ in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ also $(\mathcal{F}_{\mathbf{HG}}(f_1) = \mathcal{F}_{\mathbf{HG}}(m) \circ \mathcal{F}_{\mathbf{HG}}(e_1), \mathcal{F}_{\mathbf{HG}}(f_2) = \mathcal{F}_{\mathbf{HG}}(m) \circ \mathcal{F}_{\mathbf{HG}}(e_2))$ is a pair factorization in $(\mathbf{AGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2)$. This can be concluded, if $\mathcal{F}_{\mathbf{HG}}$ preserves coproducts and surjective morphisms according to Lemma 16. Both properties are shown to hold in Lemmas 36 and 37 above. Thus, we get that $\mathcal{F}_{\mathbf{HG}}$ is compatible with pair factorization.



□

Lemma 39: ($\mathcal{F}_{\mathbf{HG}}$ Translates Jointly Surjective Morphisms, see page 152)

Consider $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ and $\mathcal{E}'_2 - \mathcal{M}_2$ pair factorization in $(\mathbf{AGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2)$. Then the \mathcal{M} -functor $\mathcal{F}_{\mathbf{HG}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2)$ from Definition 63 translates jointly surjective hypergraph morphisms (a', b') into the corresponding jointly surjective typed attributed graph morphisms (a'', b'') with $a'' = \mathcal{F}_{\mathbf{HG}}(a')$ and $b'' = \mathcal{F}_{\mathbf{HG}}(b')$.

$$\begin{array}{ccc}
 P' & & \mathcal{F}_{\mathbf{HG}}(P') \\
 \downarrow a' & & \downarrow a'' = \mathcal{F}_{\mathbf{HG}}(a') \\
 C \xrightarrow{b'} C' & & \mathcal{F}_{\mathbf{HG}}(C) \xrightarrow{b'' = \mathcal{F}_{\mathbf{HG}}(b')} C'' = \mathcal{F}_{\mathbf{HG}}(C')
 \end{array}$$

Proof.

We have to show the following according to Definition 61:

$$\forall (a', b') \in \mathcal{E}'_1. \exists (a'', b'') \in \mathcal{E}'_2. a'' = \mathcal{F}_{\mathbf{HG}}(a') \wedge b'' = \mathcal{F}_{\mathbf{HG}}(b')$$

Let (a', b') be a pair of jointly surjective morphisms in \mathcal{E}'_1 . According to Lemma 66 from Appendix A, we know that $((a', b'), id)$ is an $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization of (a', b') in $(\mathbf{HyperGraphs},$

\mathcal{M}_1). Furthermore, we get by application of Lemma 38 that $((\mathcal{F}_{\text{HG}}(a'), \mathcal{F}_{\text{HG}}(b')), \mathcal{F}_{\text{HG}}(id))$ is an \mathcal{E}'_2 - \mathcal{M}_2 pair factorization of $(\mathcal{F}_{\text{HG}}(a'), \mathcal{F}_{\text{HG}}(b'))$ in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, which implies by the definition of \mathcal{E}' - \mathcal{M} pair factorization that $(\mathcal{F}_{\text{HG}}(a'), \mathcal{F}_{\text{HG}}(b')) \in \mathcal{E}'_2$. \square

Lemma 40: (\mathcal{F}_{HG} Creates Jointly Surjective Morphisms, see page 152)

Consider \mathcal{E}'_1 - \mathcal{M}_1 pair factorization in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, \mathcal{E}'_2 - \mathcal{M}_2 pair factorization in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, and hypergraph morphisms $a : P \rightarrow C$, $b : P \rightarrow P'$. Then the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ from Definition 63 creates jointly surjective hypergraph morphisms (a', b') from the corresponding typed attributed graph morphisms (a'', b'') in $\bar{\mathcal{E}}'_2 = \mathcal{F}_{\text{HG}}(\mathcal{E}'_1)$ if for all spans $(C \xleftarrow{a} P \xrightarrow{b} P')$ holds that the diagram (1) below commutes, $\mathcal{F}_{\text{HG}}(a') = a''$, $\mathcal{F}_{\text{HG}}(b') = b''$, and the injectivity of b'' implies the injectivity of b' .

$$\begin{array}{ccc} P & \xrightarrow{b} & P' \\ a \downarrow & (1) & \downarrow a' \\ C & \xrightarrow{b'} & C' \end{array} \quad \begin{array}{ccc} \mathcal{F}_{\text{HG}}(P) & \xrightarrow{\mathcal{F}_{\text{HG}}(b)} & \mathcal{F}_{\text{HG}}(P') \\ \mathcal{F}_{\text{HG}}(a) \downarrow & (2) & \downarrow a'' = \mathcal{F}_{\text{HG}}(a') \\ \mathcal{F}_{\text{HG}}(C) & \xrightarrow{b'' = \mathcal{F}_{\text{HG}}(b')} & C'' = \mathcal{F}_{\text{HG}}(C') \end{array}$$

Proof.

We have to show the following according to Definition 61:

$$\begin{aligned} & \forall (a'', b'') \in \bar{\mathcal{E}}'_2. (2) \text{ commutes} \wedge b'' \in \mathcal{M}_2 \Rightarrow \\ & \exists (a', b') \in \mathcal{E}'_1. a'' = \mathcal{F}_{\text{HG}}(a') \wedge b'' = \mathcal{F}_{\text{HG}}(b') \wedge (1) \text{ commutes} \wedge b' \in \mathcal{M}_1. \end{aligned}$$

Since $\bar{\mathcal{E}}'_2 = \mathcal{F}_{\text{HG}}(\mathcal{E}'_1)$, we have that there is $(a', b') \in \mathcal{E}'_1$ with $a'' = \mathcal{F}_{\text{HG}}(a')$ and $b'' = \mathcal{F}_{\text{HG}}(b')$. Furthermore, by commutativity of (2) holds:

$$\begin{aligned} & \mathcal{F}_{\text{HG}}(a') \circ \mathcal{F}_{\text{HG}}(b) = \mathcal{F}_{\text{HG}}(b') \circ \mathcal{F}_{\text{HG}}(a) \\ & \xRightarrow{\text{funct. prop.}} \mathcal{F}_{\text{HG}}(a' \circ b) = \mathcal{F}_{\text{HG}}(b' \circ a) \\ & \xRightarrow{\text{Lem. 10}} a' \circ b = b' \circ a \end{aligned}$$

Finally, we have that $b' \in \mathcal{M}_1$, because $b'' = \mathcal{F}_{\text{HG}}(b')$ in \mathcal{M}_2 and \mathcal{F}_{HG} creates injective morphisms by Lemma 31. \square

Lemma 41: (\mathcal{F}_{HG} Preserves Pullbacks of Injective Morphisms [213], see page 153)

Consider \mathcal{M} -adhesive categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$, the \mathcal{M} -functor $\mathcal{F}_{\text{HG}} : \mathbf{HyperGraphs} \rightarrow \mathbf{AGraphs}_{\text{HGTG}}$ from Definition 63, hypergraphs G_i for $i \in \{0, 1, 2, 3\}$ with hypergraph morphisms $b = (b_V, b_E)$, $c = (c_V, c_E)$, $g = (g_V, g_E)$, $h = (h_V, h_E)$, and typed attributed graphs $\mathcal{F}_{\text{HG}}(G_i)$ for $i \in \{0, 1, 2, 3\}$ with typed attributed graph morphisms $\mathcal{F}_{\text{HG}}(b) = b' = (b'_{V_G}, b'_{V_D}, b'_{E_G}, b'_{E_{NA}}, b'_{E_{EA}})$, $\mathcal{F}_{\text{HG}}(c) = c' = (c'_{V_G}, c'_{V_D}, c'_{E_G}, c'_{E_{NA}}, c'_{E_{EA}})$, $\mathcal{F}_{\text{HG}}(g) = g' = (g'_{V_G}, g'_{V_D}, g'_{E_G}, g'_{E_{NA}}, g'_{E_{EA}})$, $\mathcal{F}_{\text{HG}}(h) = h' = (h'_{V_G}, h'_{V_D}, h'_{E_G}, h'_{E_{NA}}, h'_{E_{EA}})$. If (1) is a pullback in $\mathbf{HyperGraphs}$ with $g, h \in \mathcal{M}_1$ then we have that (2) is a pullback in $\mathbf{AGraphs}_{\text{HGTG}}$ with $\mathcal{F}_{\text{HG}}(g), \mathcal{F}_{\text{HG}}(h) \in \mathcal{M}_2$.

$$\begin{array}{ccc} G_0 & \xrightarrow{b} & G_1 \\ c \downarrow & (1) & \downarrow g \\ G_2 & \xrightarrow{h} & G_3 \end{array} \quad \begin{array}{ccc} \mathcal{F}_{\text{HG}}(G_0) & \xrightarrow{\mathcal{F}_{\text{HG}}(b)=b'} & \mathcal{F}_{\text{HG}}(G_1) \\ \mathcal{F}_{\text{HG}}(c)=c' \downarrow & (2) & \downarrow \mathcal{F}_{\text{HG}}(g)=g' \\ \mathcal{F}_{\text{HG}}(G_2) & \xrightarrow{\mathcal{F}_{\text{HG}}(h)=h'} & \mathcal{F}_{\text{HG}}(G_3) \end{array}$$

Proof.

For the given morphisms $g, h \in \mathcal{M}_1$ we have that $\mathcal{F}_{\text{HG}}(g) = g'$, $\mathcal{F}_{\text{HG}}(h) = h'$ are in \mathcal{M}_2 according to Lemma 27.

Let (1) be a pullback in **HyperGraphs** with $g, h \in \mathcal{M}_1$, i.e., V - and E -components of (1) are pullbacks in **Sets**, because pullbacks in **HyperGraphs** are constructed componentwise as we have already shown in Lemma 4.

$$\begin{array}{ccc} V_0 & \xrightarrow{b_V} & V_1 \\ c_V \downarrow & \text{(PB)} & \downarrow g_V \text{ inj.} \\ V_2 & \xrightarrow{h_V \text{ inj.}} & V_3 \end{array} \quad \begin{array}{ccc} E_0 & \xrightarrow{b_E} & E_1 \\ c_E \downarrow & \text{(PB)} & \downarrow g_E \text{ inj.} \\ E_2 & \xrightarrow{h_E \text{ inj.}} & E_3 \end{array}$$

To show: V_G -, V_D -, E_G -, E_{NA} - and E_{EA} -components of (2) are pullbacks of injective morphisms in **Sets**, because according to [88] pullbacks in **AGraphs**_{ATG} and hence also in **AGraphs**_{HGTG} are constructed componentwise as well.

1. V_G -component of (2) is a pullback of injective morphisms in **Sets** (see diagram (3)) with $f'_{V_G} = f_V \uplus f_E$ for $f \in \{b, c, g, h\}$, because the components of (3) are pullbacks and pushouts of injective morphisms in **Sets** since $g = (g_V, g_E)$, $h = (h_V, h_E)$ are in \mathcal{M}_1 by assumption and pushouts are compatible with coproducts (and coproduct in **Sets** is disjoint union \uplus).

$$\begin{array}{ccc} V_G^{G_0} & \xrightarrow{b_V \uplus b_E} & V_G^{G_1} \\ c_V \uplus c_E \downarrow & \text{(3)} & \downarrow g_V \uplus g_E \\ V_G^{G_2} & \xrightarrow{h_V \uplus h_E} & V_G^{G_3} \end{array}$$

2. V_D -component of (2) is obviously a pullback of injective morphisms in **Sets** (see diagram (4)) with $f'_{V_D} = \text{id}_{\mathbb{N}}$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc} V_D^{G_0} = \mathbb{N} & \xrightarrow{\text{id}_{\mathbb{N}}} & V_D^{G_1} = \mathbb{N} \\ \text{id}_{\mathbb{N}} \downarrow & \text{(4)} & \downarrow \text{id}_{\mathbb{N}} \\ V_D^{G_2} = \mathbb{N} & \xrightarrow{\text{id}_{\mathbb{N}}} & V_D^{G_3} = \mathbb{N} \end{array}$$

3. For the E_G -component we have to show that (5) is a pullback of injective morphisms in **Sets**, which follows if (5a) and similarly (5b) are pullbacks of injective morphisms.

$$\begin{array}{ccc} E_G^{G_0} & \xrightarrow{b'_{E_G}} & E_G^{G_1} \\ c'_{E_G} \downarrow & \text{(5)} & \downarrow g'_{E_G} \\ E_G^{G_2} & \xrightarrow{h'_{E_G}} & E_G^{G_3} \end{array}$$

$$\begin{array}{ccc} E_{n2e}^{G_0} & \xrightarrow{b_V \times b_E \times \text{id}_{\mathbb{N}}} & E_{n2e}^{G_1} \\ c_V \times c_E \times \text{id}_{\mathbb{N}} \downarrow & \text{(5a)} & \downarrow g_V \times g_E \times \text{id}_{\mathbb{N}} \\ E_{n2e}^{G_2} & \xrightarrow{h_V \times h_E \times \text{id}_{\mathbb{N}}} & E_{n2e}^{G_3} \end{array} \quad \begin{array}{ccc} E_{e2n}^{G_0} & \xrightarrow{b_E \times b_V \times \text{id}_{\mathbb{N}}} & E_{e2n}^{G_1} \\ c_E \times c_V \times \text{id}_{\mathbb{N}} \downarrow & \text{(5b)} & \downarrow g_E \times g_V \times \text{id}_{\mathbb{N}} \\ E_{e2n}^{G_2} & \xrightarrow{h_E \times h_V \times \text{id}_{\mathbb{N}}} & E_{e2n}^{G_3} \end{array}$$

Diagrams (5a) and (5b) commute, because for each product component we have a pullback in **Sets** by assumption. So it remains to show that (5a) and (5b) are pullbacks.

a) For diagram (5a) we have to show that the diagram (5a') is a pullback in **Sets** with

$$V_i \otimes E_i \otimes \mathbb{N} = \{((v, e), n) \in (V_i \times E_i) \times \mathbb{N} \mid s_i^n(e) = v\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_V \otimes f_E \otimes \text{id}_{\mathbb{N}}$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc} V_0 \otimes E_0 \otimes \mathbb{N} & \xrightarrow{b_V \otimes b_E \otimes \text{id}_{\mathbb{N}}} & V_1 \otimes E_1 \otimes \mathbb{N} \\ c_V \otimes c_E \otimes \text{id}_{\mathbb{N}} \downarrow & (5a') & \downarrow g_V \otimes g_E \otimes \text{id}_{\mathbb{N}} \\ V_2 \otimes E_2 \otimes \mathbb{N} & \xrightarrow{h_V \otimes h_E \otimes \text{id}_{\mathbb{N}}} & V_3 \otimes E_3 \otimes \mathbb{N} \end{array}$$

Since b, c, g, h are hypergraph morphisms, we have that all $f_V \otimes f_E \otimes \text{id}_{\mathbb{N}}$ morphisms for $f \in \{b, c, g, h\}$ are well-defined. Furthermore, we have that the components of (5a') are pullbacks in **Sets** by assumption. Hence, also (5a') (as well as (5a)) is a pullback, because pullbacks are compatible with products.

b) For diagram (5b) we have to show that the diagram (5b') is a pullback in **Sets** with

$$E_i \otimes V_i \otimes \mathbb{N} = \{((e, v), n) \in (E_i \times V_i) \times \mathbb{N} \mid t_i^n(e) = v\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_E \otimes f_V \otimes \text{id}_{\mathbb{N}}$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc} E_0 \otimes V_0 \otimes \mathbb{N} & \xrightarrow{b_E \otimes b_V \otimes \text{id}_{\mathbb{N}}} & E_1 \otimes V_1 \otimes \mathbb{N} \\ c_E \otimes c_V \otimes \text{id}_{\mathbb{N}} \downarrow & (5b') & \downarrow g_E \otimes g_V \otimes \text{id}_{\mathbb{N}} \\ E_2 \otimes V_2 \otimes \mathbb{N} & \xrightarrow{h_E \otimes h_V \otimes \text{id}_{\mathbb{N}}} & E_3 \otimes V_3 \otimes \mathbb{N} \end{array}$$

The proof for this case is similar to the Case 3a. Hence, (5b') and (5b) are pullbacks.

4. For the E_{NA} -component we have to show that (6) is a pullback of injective morphisms in **Sets**, which follows if (6a) and similarly (6b) are pullbacks of injective morphisms for $X = \{in\}$ and $Y = \{out\}$.

$$\begin{array}{ccccc} E_{NA}^{G_0} & \xrightarrow{b'_{E_{NA}}} & E_{NA}^{G_1} & & \\ c'_{E_{NA}} \downarrow & (6) & \downarrow g'_{E_{NA}} & & \\ E_{NA}^{G_2} & \xrightarrow{h'_{E_{NA}}} & E_{NA}^{G_3} & & \\ \begin{array}{ccc} E_{in}^{G_0} & \xrightarrow{b_E \times \text{id}_{\mathbb{N}} \times \text{id}_X} & E_{in}^{G_1} \\ c_E \times \text{id}_{\mathbb{N}} \times \text{id}_X \downarrow & (6a) & \downarrow g_E \times \text{id}_{\mathbb{N}} \times \text{id}_X \\ E_{in}^{G_2} & \xrightarrow{h_E \times \text{id}_{\mathbb{N}} \times \text{id}_X} & E_{in}^{G_3} \end{array} & & \begin{array}{ccc} E_{out}^{G_0} & \xrightarrow{b_E \times \text{id}_{\mathbb{N}} \times \text{id}_Y} & E_{out}^{G_1} \\ c_E \times \text{id}_{\mathbb{N}} \times \text{id}_Y \downarrow & (6b) & \downarrow g_E \times \text{id}_{\mathbb{N}} \times \text{id}_Y \\ E_{out}^{G_2} & \xrightarrow{h_E \times \text{id}_{\mathbb{N}} \times \text{id}_Y} & E_{out}^{G_3} \end{array} \end{array}$$

Diagrams (6a) and (6b) commute, because for each product component we have a pullback in **Sets** by assumption. So it remains to show that (6a) and (6b) are pullbacks.

a) For diagram (6a) we have to show that the diagram (6a') is a pullback in **Sets** with

$$E_i \otimes \mathbb{N} \otimes X = \{(e, n, in) \mid (e, n) \in E_i \times \mathbb{N} \wedge |s_i(e) = n|\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_X$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
E_0 \otimes \mathbb{N} \otimes X & \xrightarrow{b_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_X} & E_1 \otimes \mathbb{N} \otimes X \\
c_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_X \downarrow & (6a') & \downarrow g_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_X \\
E_2 \otimes \mathbb{N} \otimes X & \xrightarrow{h_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_X} & E_3 \otimes \mathbb{N} \otimes X
\end{array}$$

Also here we know that $f_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_X$ morphisms for $f \in \{b, c, g, h\}$ are well-defined since b, c, g, h are hypergraph morphisms. Furthermore, we have that the components of (6a') are pullbacks in **Sets** by assumption. Hence, also (6a') (as well as (6a)) is a pullback, because pullbacks are compatible with products.

b) For diagram (6b) we have to show that the diagram (6b') is a pullback in **Sets** with

$$E_i \otimes \mathbb{N} \otimes Y = \{(e, n, \text{out}) \mid (e, n) \in E_i \times \mathbb{N} \wedge |t_i(e) = n|\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_Y$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
E_0 \otimes \mathbb{N} \otimes Y & \xrightarrow{b_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_Y} & E_1 \otimes \mathbb{N} \otimes Y \\
c_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_Y \downarrow & (6b') & \downarrow g_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_Y \\
E_2 \otimes \mathbb{N} \otimes Y & \xrightarrow{h_E \otimes \text{id}_{\mathbb{N}} \otimes \text{id}_Y} & E_3 \otimes \mathbb{N} \otimes Y
\end{array}$$

The proof for this case is similar to the Case 4a. Hence, (6b') and (6b) are pullbacks.

5. For the E_{EA} -component we have to show that (7) is a pullback of injective morphisms in **Sets**, which follows if (7a) and similarly (7b) are pullbacks of injective morphisms.

$$\begin{array}{ccccc}
E_{EA}^{G_0} & \xrightarrow{b'_{EA}} & E_{EA}^{G_1} & & \\
c'_{EA} \downarrow & (7) & \downarrow g'_{EA} & & \\
E_{EA}^{G_2} & \xrightarrow{h'_{EA}} & E_{EA}^{G_3} & & \\
\downarrow & & \downarrow & & \\
E_s^{G_0} & \xrightarrow{\text{id}_{\mathbb{N}} \times b_V \times b_E} & E_s^{G_1} & & E_t^{G_0} \xrightarrow{\text{id}_{\mathbb{N}} \times b_E \times b_V} E_t^{G_1} \\
\downarrow \text{id}_{\mathbb{N}} \times c_V \times c_E & (7a) & \downarrow \text{id}_{\mathbb{N}} \times g_V \times g_E & & \downarrow \text{id}_{\mathbb{N}} \times g_E \times g_V \\
E_s^{G_2} & \xrightarrow{\text{id}_{\mathbb{N}} \times h_V \times h_E} & E_s^{G_3} & & E_t^{G_2} \xrightarrow{\text{id}_{\mathbb{N}} \times h_E \times h_V} E_t^{G_3}
\end{array}$$

Diagrams (7a) and (7b) commute, because also here we have a pullback in **Sets** for each product component by assumption. So it remains to show that (7a) and (7b) are pullbacks. The proof for (7a) and (7b) is very similar to the Case 3 of the current proof switching the components of the products. Hence, (7a) and (7b) are pullbacks.

□

Lemma 62: (\mathcal{F}_{HGC} is a Category Equivalence, see [page 221](#))

The categories (**HyperGraphs**, \mathcal{M}_1) and (**SubAGraphs**_{HGTG}, \mathcal{M}_2^*) are equivalent, i.e., there exists a category equivalence $\mathcal{F}_{\text{HGC}} : (\text{HyperGraphs}, \mathcal{M}_1) \xrightarrow{\sim} (\text{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$.

Proof.

In the first step we construct the functor $\mathcal{F}_{\text{HGC}} : (\text{HyperGraphs}, \mathcal{M}_1) \rightarrow (\text{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$ according to Definition 63. \mathcal{F}_{HGC} defined in this way is a well-defined functor as it is shown in the proof of Lemma 27. Now we define the inverse functor $\mathcal{F}_{\text{HGC}}^{-1} : (\text{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*) \rightarrow (\text{HyperGraphs}, \mathcal{M}_1)$.

$\mathcal{M}_2^*) \rightarrow (\mathbf{HyperGraphs}, \mathcal{M}_1)$ on objects and morphisms as is given in the following. Consider a typed attributed graph $((G', \text{NAT}^1), \text{type})$ in $\mathbf{SubAGraphs}_{\text{HGTG}}$ with E-graph $G' = (V_G^{G'}, V_D^{G'} = \mathbb{N}, E_G^{G'}, E_{\text{NA}}^{G'}, E_{\text{EA}}^{G'}, (s_j^{G'}, t_j^{G'})_{j \in \{G, \text{NA}, \text{EA}\}})$ and morphism $\text{type} : (G', \text{NAT}) \rightarrow (\text{HGTG}, D_{\text{fin}})$ given by final morphism of data types from NAT to the final algebra D_{fin} and $\text{type}^{G'} : G' \rightarrow \text{HGTG}$ is given by E-graph morphism $\text{type}^{G'} = (\text{type}_{V_G}, \text{type}_{V_D}, \text{type}_{E_G}, \text{type}_{E_{\text{NA}}}, \text{type}_{E_{\text{EA}}})$. Since the category $\mathbf{SubAGraphs}_{\text{HGTG}}$ contains only \mathcal{F}_{HGC} -images, we have that $((G', \text{NAT}), \text{type}) = \mathcal{F}_{\text{HGC}}(G)$ for some hypergraph $G = (V_G, E_G, s_G, t_G)$. We define the object $\mathcal{F}_{\text{HGC}}^{-1}(G') = (V_{\mathcal{F}_{\text{HGC}}^{-1}(G')}, E_{\mathcal{F}_{\text{HGC}}^{-1}(G')}, s_{\mathcal{F}_{\text{HGC}}^{-1}(G')}, t_{\mathcal{F}_{\text{HGC}}^{-1}(G')})$ as follows:

$$\begin{aligned} V_{\mathcal{F}_{\text{HGC}}^{-1}(G')} &= \{v \in V_G^{G'} \mid \text{type}_{V_G}(v) = \text{Node}\} \\ E_{\mathcal{F}_{\text{HGC}}^{-1}(G')} &= \{e \in V_G^{G'} \mid \text{type}_{V_G}(e) = \text{Edge}\} \\ s_{\mathcal{F}_{\text{HGC}}^{-1}(G')} &= \{(e, v_1 \dots v_n) \mid (e, n, \text{in}) \in E_{\text{NA}}^{G'} \wedge \forall i \leq n. ((v_i, e), i) \in E_G^{G'}\} \\ t_{\mathcal{F}_{\text{HGC}}^{-1}(G')} &= \{(e, v_1 \dots v_n) \mid (e, n, \text{out}) \in E_{\text{NA}}^{G'} \wedge \forall i \leq n. ((e, v_i), i) \in E_G^{G'}\} \end{aligned}$$

Consider additionally another typed attributed graph $((H', \text{NAT}), \text{type}')$ in $\mathbf{SubAGraphs}_{\text{HGTG}}$ with E-graph $H' = (V_G^{H'}, V_D^{H'} = \mathbb{N}, E_G^{H'}, E_{\text{NA}}^{H'}, E_{\text{EA}}^{H'}, (s_j^{H'}, t_j^{H'})_{j \in \{G, \text{NA}, \text{EA}\}})$ and morphism $\text{type}' : (H', \text{NAT}) \rightarrow (\text{HGTG}, D_{\text{fin}})$. For each typed attributed graph morphism $f' : ((G', \text{NAT}), \text{type}) \rightarrow ((H', \text{NAT}), \text{type}')$ in $\mathbf{SubAGraphs}_{\text{HGTG}}$ with $f' = (f'_{V_G} : V_G^{G'} \rightarrow V_G^{H'}, f'_{V_D} : \mathbb{N} \rightarrow \mathbb{N}, f'_{E_G} : E_G^{G'} \rightarrow E_G^{H'}, f'_{E_{\text{NA}}} : E_{\text{NA}}^{G'} \rightarrow E_{\text{NA}}^{H'}, f'_{E_{\text{EA}}} : E_{\text{EA}}^{G'} \rightarrow E_{\text{EA}}^{H'})$, we define $\mathcal{F}_{\text{HGC}}^{-1}(f') : \mathcal{F}_{\text{HGC}}^{-1}(G') \rightarrow \mathcal{F}_{\text{HGC}}^{-1}(H')$ where $\mathcal{F}_{\text{HGC}}^{-1}(G') = (V_{\mathcal{F}_{\text{HGC}}^{-1}(G')}, E_{\mathcal{F}_{\text{HGC}}^{-1}(G')}, s_{\mathcal{F}_{\text{HGC}}^{-1}(G')}, t_{\mathcal{F}_{\text{HGC}}^{-1}(G')})$ and $\mathcal{F}_{\text{HGC}}^{-1}(H') = (V_{\mathcal{F}_{\text{HGC}}^{-1}(H')}, E_{\mathcal{F}_{\text{HGC}}^{-1}(H')}, s_{\mathcal{F}_{\text{HGC}}^{-1}(H')}, t_{\mathcal{F}_{\text{HGC}}^{-1}(H')})$ by $\mathcal{F}_{\text{HGC}}^{-1}(f') = f = (f_V, f_E)$ where:

$$\begin{aligned} f_V : V_{\mathcal{F}_{\text{HGC}}^{-1}(G')} &\rightarrow V_{\mathcal{F}_{\text{HGC}}^{-1}(H')} \text{ with } f_V(v) = f'_{V_G}(v) \\ f_E : E_{\mathcal{F}_{\text{HGC}}^{-1}(G')} &\rightarrow E_{\mathcal{F}_{\text{HGC}}^{-1}(H')} \text{ with } f_E(e) = f'_{V_G}(e) \end{aligned}$$

As the next step we have to show, that the functor $\mathcal{F}_{\text{HGC}}^{-1}$ introduced above is a well-defined functor. For this reason we have to show the following for an arbitrary object $((G', \text{NAT}), \text{type})$ as well as for an arbitrary morphism $f' : ((G', \text{NAT}), \text{type}) \rightarrow ((H', \text{NAT}), \text{type}')$ in $\mathbf{SubAGraphs}_{\text{HGTG}}$:

1. **The components of $\mathcal{F}_{\text{HGC}}^{-1}(G')$ are well-defined w.r.t. codomain.**

Consider a typed attributed graph $((G', \text{NAT}), \text{type})$ in $\mathbf{SubAGraphs}_{\text{HGTG}}$ with E-graph $G' = (V_G^{G'}, V_D^{G'} = \mathbb{N}, E_G^{G'}, E_{\text{NA}}^{G'}, E_{\text{EA}}^{G'}, (s_j^{G'}, t_j^{G'})_{j \in \{G, \text{NA}, \text{EA}\}})$ and morphism $\text{type} : (G', \text{NAT}) \rightarrow (\text{HGTG}, D_{\text{fin}})$ with $\text{type}^{G'} = (\text{type}_{V_G}, \text{type}_{V_D}, \text{type}_{E_G}, \text{type}_{E_{\text{NA}}}, \text{type}_{E_{\text{EA}}})$. We have to show that $\mathcal{F}_{\text{HGC}}^{-1}(G') = (V_{\mathcal{F}_{\text{HGC}}^{-1}(G')}, E_{\mathcal{F}_{\text{HGC}}^{-1}(G')}, s_{\mathcal{F}_{\text{HGC}}^{-1}(G')}, t_{\mathcal{F}_{\text{HGC}}^{-1}(G')})$ is a well-defined hypergraph. For this reason it suffices to show that the corresponding components of $\mathcal{F}_{\text{HGC}}^{-1}(G')$ and G are equal.

- $V_{\mathcal{F}_{\text{HGC}}^{-1}(G')} = V_G$:

$$\begin{aligned} &V_{\mathcal{F}_{\text{HGC}}^{-1}(G')} \\ &\stackrel{\text{Def. } \mathcal{F}_{\text{HGC}}^{-1}}{=} \{v \in V_G^{G'} \mid \text{type}_{V_G}(v) = \text{Node}\} \\ &\stackrel{V_G^{G'} \text{ from Def. 63}}{=} \{v \in V_G \uplus E_G \mid \text{type}_{V_G}(v) = \text{Node}\} \\ &\stackrel{\text{type}_{V_G} \text{ from Def. 63}}{=} V_G \end{aligned}$$

1 Similarly to Section 5.2, we consider the category $\mathbf{SubAGraphs}_{\text{HGTG}}$ with the fixed data type NAT and an identical algebra homomorphism, which implies that the V_D -component of morphisms is an identity.

- $E_{\mathcal{F}_{\text{HGC}}^{-1}(G')} = E_G :$

$$\begin{aligned}
& E_{\mathcal{F}_{\text{HGC}}^{-1}(G')} \\
& \stackrel{\text{Def. } \mathcal{F}_{\text{HGC}}^{-1}}{=} \{e \in V_G^{G'} \mid \text{type}_{V_G}(e) = \text{Edge}\} \\
& V_G^{G'} \stackrel{\text{from Def. 63}}{=} \{e \in V_G \uplus E_G \mid \text{type}_{V_G}(e) = \text{Edge}\} \\
& \text{type}_{V_G} \stackrel{\text{from Def. 63}}{=} E_G
\end{aligned}$$

- $s_{\mathcal{F}_{\text{HGC}}^{-1}(G')} = s_G :$

$$\begin{aligned}
& s_{\mathcal{F}_{\text{HGC}}^{-1}(G')} \\
& \stackrel{\text{Def. } \mathcal{F}_{\text{HGC}}^{-1}}{=} \{(e, v_1 \dots v_n) \mid (e, n, \text{in}) \in E_{\text{NA}}^{G'} \wedge \forall i \leq n. ((v_i, e), i) \in E_G^{G'}\} \\
& E_G^{G'} \stackrel{\text{from Def. 63}}{=} \{(e, v_1 \dots v_n) \mid (e, n, \text{in}) \in E_{\text{NA}}^{G'} \\
& \wedge \forall i \leq n. ((v_i, e), i) \in E_{n2e}^{G'} \uplus E_{e2n}^{G'}\} \\
& E_{\text{NA}}^{G'} \stackrel{\text{from Def. 63}}{=} \{(e, v_1 \dots v_n) \mid (e, n) \in E_G \times \mathbb{N} \wedge |s_G(e)| = n \\
& \wedge \forall i \leq n. ((v_i, e), i) \in E_{n2e}^{G'}\} \\
& E_{n2e}^{G'} \stackrel{\text{from Def. 63}}{=} \{(e, v_1 \dots v_n) \mid (e, n) \in E_G \times \mathbb{N} \wedge |s_G(e)| = n \\
& \wedge \forall i \leq n. s_G^i(e) = v_i\} \\
& = s_G
\end{aligned}$$

- $t_{\mathcal{F}_{\text{HGC}}^{-1}(G')} = t_G :$

$$\begin{aligned}
& t_{\mathcal{F}_{\text{HGC}}^{-1}(G')} \\
& \stackrel{\text{Def. } \mathcal{F}_{\text{HGC}}^{-1}}{=} \{(e, v_1 \dots v_n) \mid (e, n, \text{out}) \in E_{\text{NA}}^{G'} \wedge \forall i \leq n. ((e, v_i), i) \in E_G^{G'}\} \\
& E_G^{G'} \stackrel{\text{from Def. 63}}{=} \{(e, v_1 \dots v_n) \mid (e, n, \text{out}) \in E_{\text{NA}}^{G'} \\
& \wedge \forall i \leq n. ((e, v_i), i) \in E_{n2e}^{G'} \uplus E_{e2n}^{G'}\} \\
& E_{\text{NA}}^{G'} \stackrel{\text{from Def. 63}}{=} \{(e, v_1 \dots v_n) \mid (e, n) \in E_G \times \mathbb{N} \wedge |t_G(e)| = n \\
& \wedge \forall i \leq n. ((e, v_i), i) \in E_{e2n}^{G'}\} \\
& E_{e2n}^{G'} \stackrel{\text{from Def. 63}}{=} \{(e, v_1 \dots v_n) \mid (e, n) \in E_G \times \mathbb{N} \wedge |t_G(e)| = n \\
& \wedge \forall i \leq n. t_G^i(e) = v_i\} \\
& = t_G
\end{aligned}$$

2. **The components of $\mathcal{F}_{\text{HGC}}^{-1}(f')$ are well-defined w.r.t. codomain.**

Consider a typed attributed graph morphism $f' : ((G', \text{NAT}), \text{type}) \rightarrow ((H', \text{NAT}), \text{type}')$ in $\text{SubAGraphs}_{\text{HGTG}}$ with $f' = (f'_{V_G} : V_G^{G'} \rightarrow V_G^{H'}, f'_{V_D} : \mathbb{N} \rightarrow \mathbb{N}, f'_{E_G} : E_G^{G'} \rightarrow E_G^{H'}, f'_{E_{\text{NA}}} : E_{\text{NA}}^{G'} \rightarrow E_{\text{NA}}^{H'}, f'_{E_{\text{EA}}} : E_{\text{EA}}^{G'} \rightarrow E_{\text{EA}}^{H'})$. Since the category $\text{SubAGraphs}_{\text{HGTG}}$ contains only \mathcal{F}_{HGC} -images, we have that $f' = \mathcal{F}_{\text{HGC}}(f)$ for some hypergraph morphism $f = (f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H)$. We have to show that $\mathcal{F}_{\text{HGC}}^{-1}(f') = ((\mathcal{F}_{\text{HGC}}^{-1}(f'))_V :$

$V_{\mathcal{F}_{\text{HGC}}^{-1}(G')} \rightarrow V_{\mathcal{F}_{\text{HGC}}^{-1}(H')}, (\mathcal{F}_{\text{HGC}}^{-1}(f'))_E : E_{\mathcal{F}_{\text{HGC}}^{-1}(G')} \rightarrow E_{\mathcal{F}_{\text{HGC}}^{-1}(H')}$ is a well-defined hypergraph morphism. It suffices to show that the corresponding components of $\mathcal{F}_{\text{HGC}}^{-1}(f')$ and f are equal.

- $(\mathcal{F}_{\text{HGC}}^{-1}(f'))_V = f_V :$
Fix $v \in V_{\mathcal{F}_{\text{HGC}}^{-1}(G')}.$

$$(\mathcal{F}_{\text{HGC}}^{-1}(f'))_V(v) \stackrel{\text{Def. } \mathcal{F}_{\text{HGC}}^{-1}}{=} f'_{V_G}(v) \stackrel{f'_{V_G} \text{ from Def. 63}}{=} (f_V \uplus f_E)(v) = f_V(v)$$

- $(\mathcal{F}_{\text{HGC}}^{-1}(f'))_E = f_E :$
Fix $e \in E_{\mathcal{F}_{\text{HGC}}^{-1}(G')}.$

$$(\mathcal{F}_{\text{HGC}}^{-1}(f'))_E(e) \stackrel{\text{Def. } \mathcal{F}_{\text{HGC}}^{-1}}{=} f'_{V_G}(e) \stackrel{f'_{V_G} \text{ from Def. 63}}{=} (f_V \uplus f_E)(e) = f_E(e)$$

3. Compositionality axiom holds for $\mathcal{F}_{\text{HGC}}^{-1}$.

Consider morphisms $g' : ((G', \text{NAT}), \text{type}) \rightarrow ((H', \text{NAT}), \text{type}')$ and $f' : ((H', \text{NAT}), \text{type}') \rightarrow ((D', \text{NAT}), \text{type}'')$ in **SubAGraphs_{HGTG}**. We have to show that $\mathcal{F}_{\text{HGC}}^{-1}(f' \circ g') = \mathcal{F}_{\text{HGC}}^{-1}(f') \circ \mathcal{F}_{\text{HGC}}^{-1}(g')$. Since the category **SubAGraphs_{HGTG}** contains only \mathcal{F}_{HGC} -images, we have that $g' = \mathcal{F}_{\text{HGC}}(g)$ and $f' = \mathcal{F}_{\text{HGC}}(f)$ for hypergraph morphisms $g : G \rightarrow H$, $f : H \rightarrow D$. Thus, it suffices to show that: $\mathcal{F}_{\text{HGC}}^{-1}(\mathcal{F}_{\text{HGC}}(f) \circ \mathcal{F}_{\text{HGC}}(g)) = \mathcal{F}_{\text{HGC}}^{-1}(\mathcal{F}_{\text{HGC}}(f)) \circ \mathcal{F}_{\text{HGC}}^{-1}(\mathcal{F}_{\text{HGC}}(g))$. It holds the following:

$$\begin{aligned} & \mathcal{F}_{\text{HGC}}^{-1}(\mathcal{F}_{\text{HGC}}(f) \circ \mathcal{F}_{\text{HGC}}(g)) \\ & \stackrel{\text{Funct. prop.}}{=} \mathcal{F}_{\text{HGC}}^{-1}(\mathcal{F}_{\text{HGC}}(f \circ g)) \\ & \stackrel{\mathcal{F}_{\text{HGC}}^{-1} \text{ well-def. on morph.}}{=} f \circ g \\ & \stackrel{\mathcal{F}_{\text{HGC}}^{-1} \text{ well-def. on morph.}}{=} \mathcal{F}_{\text{HGC}}^{-1}(\mathcal{F}_{\text{HGC}}(f)) \circ \mathcal{F}_{\text{HGC}}^{-1}(\mathcal{F}_{\text{HGC}}(g)) \end{aligned}$$

4. Identity axiom holds for $\mathcal{F}_{\text{HGC}}^{-1}$.

Consider an identity morphism $\text{id}_{X'} : ((X', \text{NAT}), \text{type}) \rightarrow ((X', \text{NAT}), \text{type})$ in **SubAGraphs_{HGTG}**. We have to show that $\mathcal{F}_{\text{HGC}}^{-1}(\text{id}_{X'}) = \text{id}_{\mathcal{F}_{\text{HGC}}^{-1}(X')}$. Since the category **SubAGraphs_{HGTG}** contains only \mathcal{F}_{HGC} -images, we have that $X' = \mathcal{F}_{\text{HGC}}(X)$ for some $X \in \text{Ob}_{\text{HyperGraphs}}$. Thus, it suffices to show that: $\mathcal{F}_{\text{HGC}}^{-1}(\text{id}_{\mathcal{F}_{\text{HGC}}(X)}) = \text{id}_{\mathcal{F}_{\text{HGC}}^{-1}(\mathcal{F}_{\text{HGC}}(X))}$. It holds the following:

$$\begin{aligned} & \mathcal{F}_{\text{HGC}}^{-1}(\text{id}_{\mathcal{F}_{\text{HGC}}(X)}) \stackrel{\mathcal{F}_{\text{HGC}} \text{ is funct.}}{=} \mathcal{F}_{\text{HGC}}^{-1}(\mathcal{F}_{\text{HGC}}(\text{id}_X)) \\ & \stackrel{\mathcal{F}_{\text{HGC}}^{-1} \text{ well-def. on morph.}}{=} \text{id}_X \stackrel{\mathcal{F}_{\text{HGC}}^{-1} \text{ well-def. on morph.}}{=} \text{id}_{\mathcal{F}_{\text{HGC}}^{-1}(\mathcal{F}_{\text{HGC}}(X))} \end{aligned}$$

It remains now to show that $\mathcal{F}_{\text{HGC}} : (\text{HyperGraphs}, \mathcal{M}_1) \xrightarrow{\sim} (\text{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$ is a category equivalence. For this reason, according to Definitions 66 and 67, we have to show that:

$$\begin{aligned} & \forall A \in \text{Ob}_{\text{HyperGraphs}}. \exists t_1(A) : (\mathcal{F}_{\text{HGC}}^{-1} \circ \mathcal{F}_{\text{HGC}})(A) \rightarrow \text{Id}_{\text{HyperGraphs}}(A). \\ & t_1(A) \text{ is an isomorphism in } \text{HyperGraphs} \wedge \\ & \forall A' \in \text{Ob}_{\text{SubAGraphs}_{\text{HGTG}}}. \exists t_2(A') : (\mathcal{F}_{\text{HGC}} \circ \mathcal{F}_{\text{HGC}}^{-1})(A') \rightarrow \text{Id}_{\text{SubAGraphs}_{\text{HGTG}}}(A'). \\ & t_2(A') \text{ is an isomorphism in } \text{SubAGraphs}_{\text{HGTG}} \end{aligned}$$

Fix objects $A \in \mathbf{Ob}_{\mathbf{HyperGraphs}}$, $A' \in \mathbf{Ob}_{\mathbf{SubAGraphs}_{\mathbf{HGTG}}}$ and let $t_1(A) = \text{id}_A$ and $t_2(A') = \text{id}_{A'}$. Obviously, id_A and $\text{id}_{A'}$ are isomorphisms in **HyperGraphs**, **SubAGraphs_{HGTG}**, respectively. Furthermore, it holds the following:

$$\begin{aligned} t_1(A) &= \text{id}_A : A \rightarrow A \\ \mathcal{F}_{\mathbf{HGC}}^{-1} \text{ well-def. on obj.} &= \text{id}_A : (\mathcal{F}_{\mathbf{HGC}}^{-1} \circ \mathcal{F}_{\mathbf{HGC}})(A) \rightarrow A \\ \text{Def. Id}_{\mathbf{HyperGraphs}} &= \text{id}_A : (\mathcal{F}_{\mathbf{HGC}}^{-1} \circ \mathcal{F}_{\mathbf{HGC}})(A) \rightarrow \text{Id}_{\mathbf{HyperGraphs}}(A) \end{aligned}$$

and

$$\begin{aligned} t_2(A') &= \text{id}_{A'} : A' \rightarrow A' \\ \text{Def. Id}_{\mathbf{SubAGraphs}_{\mathbf{HGTG}}} &= \text{id}_{A'} : A' \rightarrow \text{Id}_{\mathbf{SubAGraphs}_{\mathbf{HGTG}}}(A') \\ (*) &= \text{id}_{A'} : (\mathcal{F}_{\mathbf{HGC}} \circ \mathcal{F}_{\mathbf{HGC}}^{-1})(A') \rightarrow \text{Id}_{\mathbf{SubAGraphs}_{\mathbf{HGTG}}}(A') \end{aligned}$$

(*): This step is possible since we know that the functor $\mathcal{F}_{\mathbf{HGC}}^{-1}$ is well-defined on objects and for an arbitrary object $B \in \mathbf{Ob}_{\mathbf{HyperGraphs}}$ it holds the following:

$$\begin{aligned} (\mathcal{F}_{\mathbf{HGC}}^{-1} \circ \mathcal{F}_{\mathbf{HGC}})(B) &= B \\ \text{Funct. prop.} &\Leftrightarrow \mathcal{F}_{\mathbf{HGC}}^{-1}(\mathcal{F}_{\mathbf{HGC}}(B)) = B \\ &\Leftrightarrow \mathcal{F}_{\mathbf{HGC}}(\mathcal{F}_{\mathbf{HGC}}^{-1}(\mathcal{F}_{\mathbf{HGC}}(B))) = \mathcal{F}_{\mathbf{HGC}}(B) \\ A' = \mathcal{F}_{\mathbf{HGC}}(B) &\Leftrightarrow \mathcal{F}_{\mathbf{HGC}}(\mathcal{F}_{\mathbf{HGC}}^{-1}(A')) = A' \\ \text{Funct. prop.} &\Leftrightarrow (\mathcal{F}_{\mathbf{HGC}} \circ \mathcal{F}_{\mathbf{HGC}}^{-1})(A') = A' \end{aligned}$$

Thus, we obtain that $\mathcal{F}_{\mathbf{HGC}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{SubAGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2^*)$ is a category equivalence² implying that the categories $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ and $(\mathbf{SubAGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2^*)$ are equivalent, which was to be shown. \square

Lemma 63: ($\mathbf{I}_{\mathbf{HG}}$ Satisfies Required Properties, see [page 222](#))

Consider a transformation system $(\mathbf{SubAGraphs}_{\mathbf{HGTG}}, \mathcal{F}_{\mathbf{HG}}(P))$ with a distinguished class of monomorphisms \mathcal{M}_2^* , an \mathcal{M} -adhesive transformation system $(\mathbf{AGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2, \mathcal{F}_{\mathbf{HG}}(P))$, and a functor $\mathcal{F}_{\mathbf{HG}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2)$. Then the inclusion functor $\mathbf{I}_{\mathbf{HG}} : (\mathbf{SubAGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2^*) \rightarrow (\mathbf{AGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2)$ satisfies the properties listed in Definition 69.

Proof.

Consider \mathcal{M} -adhesive transformation system $(\mathbf{HyperGraphs}, \mathcal{M}_1, P)$, the functor $\mathcal{F}_{\mathbf{HG}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{AGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2)$ from Definition 63, the subcategory of typed attributed graphs $\mathbf{SubAGraphs}_{\mathbf{HGTG}}$ with $\mathbf{Ob}_{\mathbf{SubAGraphs}_{\mathbf{HGTG}}} = \mathcal{F}_{\mathbf{HG}}(\mathbf{Ob}_{\mathbf{HyperGraphs}})$ and $\mathbf{Mor}_{\mathbf{SubAGraphs}_{\mathbf{HGTG}}}(\mathcal{F}_{\mathbf{HG}}(A), \mathcal{F}_{\mathbf{HG}}(B)) = \mathcal{F}_{\mathbf{HG}}(\mathbf{Mor}_{\mathbf{HyperGraphs}}(A, B))$ for arbitrary $A, B \in \mathbf{Ob}_{\mathbf{HyperGraphs}}$, and functors $\mathcal{F}_{\mathbf{HGC}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \rightarrow (\mathbf{SubAGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2^*)$, $\mathcal{F}_{\mathbf{HGC}}^{-1} : (\mathbf{SubAGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2^*) \rightarrow (\mathbf{HyperGraphs}, \mathcal{M}_1)$ building a category equivalence $\mathcal{F}_{\mathbf{HGC}} : (\mathbf{HyperGraphs}, \mathcal{M}_1) \xrightarrow{\sim} (\mathbf{SubAGraphs}_{\mathbf{HGTG}}, \mathcal{M}_2^*)$. Let moreover, $((G, \text{NAT}), \text{type})$ be a typed attributed graph in $\mathbf{SubAGraphs}_{\mathbf{HGTG}}$ with E -graph $G = (V_G^G, V_D^G = \mathbb{N}, E_G^G, E_{\text{NA}}^G, E_{\text{EA}}^G, (s_j^G, t_j^G)_{j \in \{G, \text{NA}, \text{EA}\}})$ and morphism $\text{type} : (G, \text{NAT}) \rightarrow (\mathbf{HGTG}, D_{\text{fin}})$ given by final

² In this case, we even have a category isomorphism according to Definition 68 since it obviously holds that $\mathcal{F}_{\mathbf{HGC}} \circ \mathcal{F}_{\mathbf{HGC}}^{-1} = \text{Id}_{\mathbf{SubAGraphs}_{\mathbf{HGTG}}}$ and $\mathcal{F}_{\mathbf{HGC}}^{-1} \circ \mathcal{F}_{\mathbf{HGC}} = \text{Id}_{\mathbf{HyperGraphs}}$.

morphism of data types from \mathbf{NAT} to the final algebra D_{fin} and $\text{type}^G : G \rightarrow \mathbf{HGTG}$ with $\text{type}^G = (\text{type}_{V_G}, \text{type}_{V_D}, \text{type}_{E_G}, \text{type}_{E_{NA}}, \text{type}_{E_{EA}})$.

We have to show that the inclusion functor $I_{\text{HG}} : (\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*) \rightarrow (\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ satisfies the following properties:

1. I_{HG} **preserves monomorphisms, i.e.**, $I_{\text{HG}}(\mathcal{M}_2^*) \subseteq \mathcal{M}_2$:

By definition of I_{HG} , we know that $I_{\text{HG}}(\mathcal{M}_2^*) = \mathcal{M}_2^*$. Furthermore, we have by construction of \mathcal{M}_2^* that $\mathcal{M}_2^* \subseteq \mathcal{M}_2$.

2. I_{HG} **preserves pushouts along \mathcal{M} -morphisms**:

Let $(D, g_1 : B \rightarrow D, g_2 : C \rightarrow D)$ be a pushout in $\mathbf{SubAGraphs}_{\text{HGTG}}$ with $f_1 : A \rightarrow B$ in \mathcal{M}_2^* . Since pushouts in $\mathbf{HyperGraphs}$ are constructed componentwise and the functor \mathcal{F}_{HGC} preserves the componentwise construction, pushouts in $\mathbf{SubAGraphs}_{\text{HGTG}}$ are constructed componentwise as well and we have that the V_G -, V_D -, E_G -, E_{NA} - and E_{EA} -components of (1) are pushouts in **Sets**. Thus, also $(I_{\text{HG}}(D), I_{\text{HG}}(g_1) : I_{\text{HG}}(B) \rightarrow I_{\text{HG}}(D), I_{\text{HG}}(g_2) : I_{\text{HG}}(C) \rightarrow I_{\text{HG}}(D))$ with $I_{\text{HG}}(D) = D$, $I_{\text{HG}}(g_1) = g_1$ and $I_{\text{HG}}(g_2) = g_2$ is constructed componentwise in $\mathbf{AGraphs}_{\text{HGTG}}$ such that the V_G -, V_D -, E_G -, E_{NA} - and E_{EA} -components of (2) are pushouts in **Sets**. This implies that (2) is a pushout in $\mathbf{AGraphs}_{\text{HGTG}}$. Furthermore, since I_{HG} preserves monomorphisms according to the property shown before, we have for $f_1 \in \mathcal{M}_2^*$ that $I_{\text{HG}}(f_1) \in \mathcal{M}_2$.

$$\begin{array}{ccc}
 A \xrightarrow{f_1 \in \mathcal{M}_2^*} B & & I_{\text{HG}}(A)=A \xrightarrow{(I_{\text{HG}}(f_1)=f_1) \in \mathcal{M}_2} I_{\text{HG}}(B)=B \\
 f_2 \downarrow \quad (1) \quad \downarrow g_1 & & I_{\text{HG}}(f_2)=f_2 \downarrow \quad (2) \quad \downarrow I_{\text{HG}}(g_1)=g_1 \\
 C \xrightarrow{g_2} D & & I_{\text{HG}}(C)=C \xrightarrow{I_{\text{HG}}(g_2)=g_2} I_{\text{HG}}(D)=D
 \end{array}$$

3. I_{HG} **creates morphisms**:

Consider a morphism $f' : I_{\text{HG}}(A) \rightarrow I_{\text{HG}}(B)$ in $\mathbf{Mor}_{\mathbf{AGraphs}_{\text{HGTG}}}$. We have to show that there is exactly one morphism $f : A \rightarrow B$ in $\mathbf{Mor}_{\mathbf{SubAGraphs}_{\text{HGTG}}}$ such that $I_{\text{HG}}(f) = f'$. f' satisfies already the existence property with $f' : A \rightarrow B$ and $I_{\text{HG}}(f') = f'$, because $I_{\text{HG}}(A) = A$ and $I_{\text{HG}}(B) = B$. Let $f'' : A \rightarrow B$ be another $\mathbf{SubAGraphs}_{\text{HGTG}}$ -morphism for which it holds that $I_{\text{HG}}(f'') = f'$. This implies that $I_{\text{HG}}(f'') = I_{\text{HG}}(f')$ and by definition of I_{HG} we get that $f'' = f'$.

4. I_{HG} **creates \mathcal{M} -morphisms**:

Consider an \mathcal{M}_2 -morphism $f' : I_{\text{HG}}(A) \rightarrow I_{\text{HG}}(B)$ in $\mathbf{Mor}_{\mathbf{AGraphs}_{\text{HGTG}}}$. We have to show that there is exactly one \mathcal{M}_2^* -morphism $f : A \rightarrow B$ in $\mathbf{Mor}_{\mathbf{SubAGraphs}_{\text{HGTG}}}$ such that $I_{\text{HG}}(f) = f'$. f' satisfies already the existence property with $f' : A \rightarrow B$ and $I_{\text{HG}}(f') = f'$, because $I_{\text{HG}}(A) = A$ and $I_{\text{HG}}(B) = B$. Since the category $\mathbf{SubAGraphs}_{\text{HGTG}}$ contains only \mathcal{F}_{HGC} -images and \mathcal{F}_{HGC} creates \mathcal{M} -morphisms according to Lemma 58, there is exactly one \mathcal{M}_1 -morphism $\bar{f} : \bar{A} \rightarrow \bar{B}$ such that $\mathcal{F}_{\text{HGC}}(\bar{f}) \stackrel{\text{Def. } \mathcal{F}_{\text{HGC}}}{=} \mathcal{F}_{\text{HGC}}(\bar{f}) = f'$. This implies that $f' \in \mathcal{F}_{\text{HGC}}(\mathcal{M}_1) \cap \mathcal{M}_2 \stackrel{\mathcal{F}_{\text{HGC}}(\mathcal{M}_1) \subseteq \mathcal{M}_2^* \subseteq \mathcal{M}_2}{=} \mathcal{F}_{\text{HGC}}(\mathcal{M}_1) \stackrel{\text{Def. } \mathcal{F}_{\text{HGC}}}{=} \mathcal{F}_{\text{HGC}}(\mathcal{M}_1) \stackrel{\text{Def. } \mathcal{M}_2^*}{=} \mathcal{M}_2^*$. Let $f'' : A \rightarrow B$ be another $\mathbf{SubAGraphs}_{\text{HGTG}}$ -monomorphism for which it holds that $I_{\text{HG}}(f'') = f'$. This implies that $I_{\text{HG}}(f'') = I_{\text{HG}}(f')$ and by definition of I_{HG} we get that $f'' = f'$.

5. I_{HG} **preserves initial pushouts**:

Consider an initial pushout (2) over the morphism $\mathcal{F}_{\text{HGC}}(f) : \mathcal{F}_{\text{HGC}}(L) \rightarrow \mathcal{F}_{\text{HGC}}(G)$ in $(\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*)$ consisting of \mathcal{F}_{HGC} -images only with boundary object $\mathcal{F}_{\text{HGC}}(B)$, context object $\mathcal{F}_{\text{HGC}}(C)$ and morphisms $\mathcal{F}_{\text{HGC}}(b), \mathcal{F}_{\text{HGC}}(c) \in \mathcal{M}_2^*$. This initial pushout is the \mathcal{F}_{HGC} -translation of the diagram (1) with boundary object B , context object

C constructed as given in Lemma 5 resp. Lemma 6 and morphisms $b, c \in \mathcal{M}_1$, which is an initial pushout in $(\mathbf{HyperGraphs}, \mathcal{M}_1)$ since $\mathcal{F}_{\text{HGC}}^{-1} : (\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*) \xrightarrow{\sim} (\mathbf{HyperGraphs}, \mathcal{M}_1)$ is a category equivalence as well and hence the functor $\mathcal{F}_{\text{HGC}}^{-1} : (\mathbf{SubAGraphs}_{\text{HGTG}}, \mathcal{M}_2^*) \rightarrow (\mathbf{HyperGraphs}, \mathcal{M}_1)$ from this category equivalence preserves initial pushouts by the corresponding instantiation of Lemma 58.

We have to show that the diagram (3) is an initial pushout in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$.

Let $\mathcal{F}_{\text{HGC}}(B) = B^* = ((B_0^*, \text{NAT}), \text{type})$ be given according to Lemma 32. Since I_{HG} preserves monomorphisms and pushouts along \mathcal{M} -morphisms as already shown before, we have that (3) is a pushout in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ with \mathcal{M}_2 -morphisms $I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(b)) = \mathcal{F}_{\text{HGC}}(b)$ and $I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(c)) = \mathcal{F}_{\text{HGC}}(c)$. Furthermore, by definition of I_{HG} we obviously have that the diagrams (2) and (3) are isomorphic. Now we construct the initial pushout (4) over the morphism $\mathcal{F}_{\text{HGC}}(f) : \mathcal{F}_{\text{HGC}}(L) \rightarrow \mathcal{F}_{\text{HGC}}(G)$ in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ with boundary $B' = ((B'_0, \text{NAT}), \text{type})$ defined according to Lemma 33 and morphisms $b' : B' \rightarrow \mathcal{F}_{\text{HGC}}(L)$, $c' : C' \rightarrow \mathcal{F}_{\text{HGC}}(G)$ in \mathcal{M}_2 . The initiality of (4) implies the existence of unique morphisms $i : B' \rightarrow \mathcal{F}_{\text{HGC}}(B)$ and $j : C' \rightarrow \mathcal{F}_{\text{HGC}}(C)$ such that (5) is a pushout in $\mathbf{AGraphs}_{\text{HGTG}}$ and (6), (7) commute with $i, j \in \mathcal{M}_2$. The surjectivity of $i : B' \rightarrow \mathcal{F}_{\text{HGC}}(B)$, i.e., $\mathcal{F}_{\text{HGC}}(B) = B^* \subseteq B'$, follows from the surjectivity proof given for Lemma 34 replacing \mathcal{F}_{HG} by $I_{\text{HG}} \circ \mathcal{F}_{\text{HGC}}$. Thus, i is injective and surjective, so we get that i is an isomorphism. Since (5) is a pushout, also $j : C' \rightarrow \mathcal{F}_{\text{HGC}}(C)$ is an isomorphism and hence (3) is isomorphic to (4). So we get that also (3) is an initial pushout over $\mathcal{F}_{\text{HGC}}(f) : \mathcal{F}_{\text{HGC}}(L) \rightarrow \mathcal{F}_{\text{HGC}}(G)$.

$$\begin{array}{ccc}
 B & \xrightarrow{b} & L \\
 g \downarrow & (1) & \downarrow f \\
 C & \xrightarrow{c} & G
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathcal{F}_{\text{HGC}}(B) & \xrightarrow{\mathcal{F}_{\text{HGC}}(b)} & \mathcal{F}_{\text{HGC}}(L) \\
 \mathcal{F}_{\text{HGC}}(g) \downarrow & (2) & \downarrow \mathcal{F}_{\text{HGC}}(f) \\
 \mathcal{F}_{\text{HGC}}(C) & \xrightarrow{\mathcal{F}_{\text{HGC}}(c)} & \mathcal{F}_{\text{HGC}}(G)
 \end{array}$$

$$\begin{array}{ccccc}
 & & b' & & \\
 & \swarrow & & \searrow & \\
 B' & \xrightarrow{i} & I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(B)) & \xrightarrow{I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(b))} & I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(L)) \\
 g' \downarrow & (5) & \downarrow I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(g)) & (3) & \downarrow I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(f)) \\
 C' & \xrightarrow{j} & I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(C)) & \xrightarrow{I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(c))} & I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(G)) \\
 & & c' & & \\
 & \swarrow & & \searrow & \\
 B' & \xrightarrow{b'} & I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(L)) & & \\
 g' \downarrow & (4) & \downarrow I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(f)) & & \\
 C' & \xrightarrow{c'} & I_{\text{HG}}(\mathcal{F}_{\text{HGC}}(G)) & &
 \end{array}$$

6. I_{HG} preserves pullbacks of \mathcal{M} -morphisms:

Let $(A, f_1 : A \rightarrow B, f_2 : A \rightarrow C)$ be a pullback in $\mathbf{SubAGraphs}_{\text{HGTG}}$ with $g_1 : B \rightarrow D$, $g_2 : C \rightarrow D$ in \mathcal{M}_2^* . Since pullbacks in $\mathbf{HyperGraphs}$ are constructed componentwise and the functor \mathcal{F}_{HGC} preserves the componentwise construction, pullbacks in $\mathbf{SubAGraphs}_{\text{HGTG}}$ are constructed componentwise as well and we have that

the V_G^- , V_D^- , E_G^- , E_{NA^-} and E_{EA^-} -components of (1) are pullbacks in **Sets**. Thus, also $(I_{HG}(A), I_{HG}(f_1) : I_{HG}(A) \rightarrow I_{HG}(B), I_{HG}(f_2) : I_{HG}(A) \rightarrow I_{HG}(C))$ with $I_{HG}(A) = A$, $I_{HG}(f_1) = f_1$ and $I_{HG}(f_2) = f_2$ is constructed componentwise in **AGraphs_{HGTG}** such that the V_G^- , V_D^- , E_G^- , E_{NA^-} and E_{EA^-} -components of (2) are pullbacks in **Sets**. This implies that (2) is a pullback in **AGraphs_{HGTG}**. Furthermore, since I_{HG} preserves monomorphisms according to the first shown property of this proof, we have for $g_1, g_2 \in \mathcal{M}_2^*$ that $I_{HG}(g_1), I_{HG}(g_2) \in \mathcal{M}_2$.

$$\begin{array}{ccc}
 A & \xrightarrow{f_1} & B \\
 f_2 \downarrow & (1) & \downarrow g_1 \in \mathcal{M}_2^* \\
 C & \xrightarrow{g_2 \in \mathcal{M}_2^*} & D
 \end{array}
 \qquad
 \begin{array}{ccc}
 I_{HG}(A)=A & \xrightarrow{I_{HG}(f_1)=f_1} & I_{HG}(B)=B \\
 I_{HG}(f_2)=f_2 \downarrow & (2) & \downarrow (I_{HG}(g_1)=g_1) \in \mathcal{M}_2 \\
 I_{HG}(C)=C & \xrightarrow{(I_{HG}(g_2)=g_2) \in \mathcal{M}_2} & I_{HG}(D)=D
 \end{array}$$

7. I_{HG} **preserves epimorphisms, i.e.,** $I_{HG}(\mathcal{E}_2^*) \subseteq \mathcal{E}_2$:

By definition of I_{HG} , we know that $I_{HG}(\mathcal{E}_2^*) = \mathcal{E}_2^*$. Furthermore, we have by construction of \mathcal{E}_2^* that $\mathcal{E}_2^* \subseteq \mathcal{E}_2$.

8. I_{HG} **preserves coproducts:**

Let $(A, (u_i)_{i \in I})$ be a coproduct in **SubAGraphs_{HGTG}** with $A \in \text{Ob}_{\text{SubAGraphs}_{\text{HGTG}}}$, a family of **SubAGraphs_{HGTG}**-morphisms $u_i : A_i \rightarrow A$ and an index set I . Since coproducts in **HyperGraphs** are constructed componentwise and the functor \mathcal{F}_{HGC} preserves the componentwise construction, coproducts in **SubAGraphs_{HGTG}** are constructed componentwise as well and we have that the corresponding V_G^- , V_D^- , E_G^- , E_{NA^-} and E_{EA^-} -components of (1) are coproducts in **Sets**. Thus, also $(I_{HG}(A), (I_{HG}(u_i))_{i \in I})$ with $I_{HG}(A) = A$ and $I_{HG}(u_i) = u_i$ is constructed componentwise in **AGraphs_{HGTG}** such that the corresponding V_G^- , V_D^- , E_G^- , E_{NA^-} and E_{EA^-} -components of (2) are coproducts in **Sets**. This implies that $(I_{HG}(A), (I_{HG}(u_i))_{i \in I})$ is a coproduct in **AGraphs_{HGTG}**.

$$\begin{array}{ccc}
 A_i & \xrightarrow{u_i} & A \\
 & \searrow f_i & \downarrow f \\
 & & B
 \end{array}
 \qquad
 \begin{array}{ccc}
 I_{HG}(A_i) & \xrightarrow{I_{HG}(u_i)} & I_{HG}(A) \\
 & \searrow I_{HG}(f_i) & \downarrow I_{HG}(f) \\
 & & I_{HG}(B)
 \end{array}$$

9. I_{HG} **preserves \mathcal{E}' -instances:**

According to Definition 61, we have to show the following:

$$\forall (a', b') \in \mathcal{E}_2''. \exists (a'', b'') \in \mathcal{E}_2'. a'' = I_{HG}(a') \wedge b'' = I_{HG}(b')$$

Let (a', b') be a pair of jointly surjective morphisms in \mathcal{E}_2'' and define $a'' = I_{HG}(a')$, $b'' = I_{HG}(b')$. Then it remains to show that $(I_{HG}(a'), I_{HG}(b')) \in \mathcal{E}_2'$. For this reason it is obviously sufficient to show that I_{HG} is compatible with pair factorization, which means by Definition 51 that categories **SubAGraphs_{HGTG}** and **AGraphs_{HGTG}** have pair factorizations and I_{HG} translates pair factorization in **SubAGraphs_{HGTG}** into the corresponding pair factorization in **AGraphs_{HGTG}**. According to [88], we know that the category **AGraphs_{ATG}** and hence also **AGraphs_{HGTG}** has $\mathcal{E}_2' - \mathcal{M}_2$ pair factorizations. In the next step, we want to show that also the subcategory **SubAGraphs_{HGTG}** has pair factorizations. As already mentioned in Section 7.1, we can construct an $\mathcal{E}_2'' - \mathcal{M}_2^*$ pair factorization using binary coproducts and $\mathcal{E}_2^* - \mathcal{M}_2^*$ -factorizations in **SubAGraphs_{HGTG}**. We have binary coproducts in **SubAGraphs_{HGTG}** because \emptyset is an initial object in **SubAGraphs_{HGTG}** and **SubAGraphs_{HGTG}** has pushouts since the functor \mathcal{F}_{HGC} preserves pushouts along \mathcal{M} -morphisms by application of Lemma 58. Moreover, we have $\mathcal{E}_2^* - \mathcal{M}_2^*$ -factorizations in **SubAGraphs_{HGTG}** since the category **HyperGraphs** has $\mathcal{E}_1 - \mathcal{M}_1$ -factorizations according to Lemma 35 and the functor \mathcal{F}_{HGC} preserves $\mathcal{E} - \mathcal{M}$ -factorizations, which can be shown

as follows: Let $A \xrightarrow{e} X \xrightarrow{m} B$ be an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization of the morphism $f : A \rightarrow B$ in **HyperGraphs** with $e \in \mathcal{E}_1$ and $m \in \mathcal{M}_1$ according to Definition 10. Applying the functor \mathcal{F}_{HGC} to the commuting triangle (1), we get a commuting triangle (2), because functors preserve commuting diagrams by general functor property. Furthermore, according to Lemma 58 the functor \mathcal{F}_{HGC} preserves monomorphisms and epimorphisms, which implies that $\mathcal{F}_{\text{HGC}}(e) \in \mathcal{E}_2^*$ and $\mathcal{F}_{\text{HGC}}(m) \in \mathcal{M}_2^*$. It remains to show that the object $\mathcal{F}_{\text{HGC}}(X)$ is unique up to isomorphism. For this reason consider another object $X' \in \text{Ob}_{\text{SubAGraphs}_{\text{HGTG}}}$ and morphisms $e' : \mathcal{F}_{\text{HGC}}(A) \rightarrow X'$, $m' : X' \rightarrow \mathcal{F}_{\text{HGC}}(B)$ in $\text{Mor}_{\text{SubAGraphs}_{\text{HGTG}}}$ such that it holds that $e' \in \mathcal{E}_2^*$, $m' \in \mathcal{M}_2^*$ and $\mathcal{F}_{\text{HGC}}(f) = m' \circ e'$. We have to show that $\mathcal{F}_{\text{HGC}}(X) \cong X'$. According to the definition of the category **SubAGraphs**_{HGTG}, we know that X' , e' and m' are \mathcal{F}_{HGC} -images. This means that for some object $X'' \in \text{Ob}_{\text{HyperGraphs}}$ and morphisms $e'' : A \rightarrow X''$, $m'' : X'' \rightarrow B$ in $\text{Mor}_{\text{HyperGraphs}}$ it holds that $X' = \mathcal{F}_{\text{HGC}}(X'')$, $e' = \mathcal{F}_{\text{HGC}}(e'')$ and $m' = \mathcal{F}_{\text{HGC}}(m'')$. Furthermore, by Lemma 58 we know that the functor \mathcal{F}_{HGC} creates \mathcal{M} -morphisms implying that $m'' \in \mathcal{M}_1$. To show that $e'' \in \mathcal{E}_1$, let us first assume that $e'' \notin \mathcal{E}_1$. This implies directly a contradiction, namely that also $(\mathcal{F}_{\text{HGC}}(e'')) = e' \notin \mathcal{E}_2^*$ since we have the following: Let $e'' = (e''_V, e''_E)$. Then $e'' \notin \mathcal{E}_1$ means that either e''_V or e''_E is not surjective. Let us first assume that e''_V is not surjective, which means that there is $y \in V_{X''}$ such that $\forall x \in V_A. e''_V(x) \neq y$. It remains to show that also $\mathcal{F}_{\text{HGC}}(e'')$ is not surjective. For this reason it suffices to show that $\mathcal{F}_{\text{HGC}}(e'')_{V_G}$ is not surjective, which follows from the property that $\forall x \in V_G^A. \mathcal{F}_{\text{HGC}}(e'')_{V_G}(x) \neq y$ since y is also in $V_G^{\overline{X''}} = V_{X''} \uplus E_{X''}$. Fix $x \in (V_G^A = V_A \uplus E_A)$. Then we have the following:

$$\mathcal{F}_{\text{HGC}}(e'')_{V_G}(x) = (e''_V \uplus e''_E)(x)$$

Let $x \in V_A$ then it holds:

$$(e''_V \uplus e''_E)(x) = e''_V(x) \neq y \text{ by assumption for } y \in V_{X''}$$

Let $x \in E_A$ then it holds:

$$(e''_V \uplus e''_E)(x) = e''_E(x) \neq y \text{ for } y \in V_{X''}$$

Hence, we get that $e'' \in \mathcal{E}_1$. Finally, we can show that $f = m'' \circ e''$ as follows:

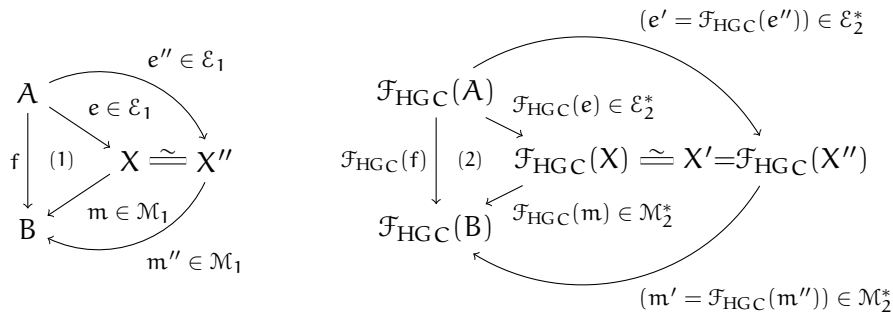
$$\mathcal{F}_{\text{HGC}}(f) = e' \circ m'$$

$$\Rightarrow \mathcal{F}_{\text{HGC}}(f) = \mathcal{F}_{\text{HGC}}(e'') \circ \mathcal{F}_{\text{HGC}}(m'')$$

$$\Rightarrow \mathcal{F}_{\text{HGC}}(f) = \mathcal{F}_{\text{HGC}}(e'' \circ m'')$$

$$\xRightarrow{\text{Def. } \mathcal{F}_{\text{HGC}}} \mathcal{F}_{\text{HG}}(f) = \mathcal{F}_{\text{HG}}(e'' \circ m'')$$

$$\xRightarrow{\text{Lem. } 10} f = e'' \circ m''$$



Thus, we get that $A \xrightarrow{e''} X'' \xrightarrow{m''} B$ is an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization of the morphism $f : A \rightarrow B$ in **HyperGraphs** with $e'' \in \mathcal{E}_1$, $m'' \in \mathcal{M}_1$ and $f = m'' \circ e''$ according to Definition 10. But since $A \xrightarrow{e} X \xrightarrow{m} B$ is an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization of the morphism $f : A \rightarrow B$ in **HyperGraphs** by assumption, we have that $X'' \cong X$, which implies by application of the functor \mathcal{F}_{HGC} that also $\mathcal{F}_{\text{HGC}}(X) \cong X'$. Thus, we get that $\mathcal{F}_{\text{HGC}}(A) \xrightarrow{\mathcal{F}_{\text{HGC}}(e)} \mathcal{F}_{\text{HGC}}(X) \xrightarrow{\mathcal{F}_{\text{HGC}}(m)} \mathcal{F}_{\text{HGC}}(B)$ is an $\mathcal{E}_2^* - \mathcal{M}_2^*$ -factorization of the morphism $\mathcal{F}_{\text{HGC}}(f)$ in **SubAGraphs_{HGTG}** and hence the functor \mathcal{F}_{HGC} preserves $\mathcal{E} - \mathcal{M}$ -factorizations.

Moreover, since I_{HG} preserves coproducts, monomorphisms and epimorphisms according to the proofs of the previous properties, we get using Lemma 16 that I_{HG} translates $\mathcal{E}_2'' - \mathcal{M}_2^*$ pair factorization in **SubAGraphs_{HGTG}** into the corresponding $\mathcal{E}_2' - \mathcal{M}_2$ pair factorization in **AGraphs_{HGTG}**. Thus, by Definition 51, we have that I_{HG} is compatible with pair factorization implying that I_{HG} preserves \mathcal{E}' -instances.

10. I_{HG} creates \mathcal{E}' -instances:

According to Definition 61, we have to show the following:

$$\begin{aligned} & \forall (\alpha'', \beta'') \in I_{\text{HG}}(\mathcal{E}_2''). \text{ (2) commutes} \wedge \beta'' \in \mathcal{M}_2 \Rightarrow \\ & \exists (\alpha', \beta') \in \mathcal{E}_2''. \alpha'' = I_{\text{HG}}(\alpha') \wedge \beta'' = I_{\text{HG}}(\beta') \wedge \text{(1) commutes} \wedge \beta' \in \mathcal{M}_2^* \end{aligned}$$

Let (α'', β'') be a pair of jointly surjective morphisms in $I_{\text{HG}}(\mathcal{E}_2'')$ such that (2) commutes and $\beta'' \in \mathcal{M}_2$. Since $I_{\text{HG}}(\mathcal{E}_2'') = \mathcal{E}_2''$, we have that there is $(\alpha', \beta') \in \mathcal{E}_2''$, namely the pair (α'', β'') , with $I_{\text{HG}}(\alpha'') = \alpha''$, $I_{\text{HG}}(\beta'') = \beta''$ and the diagram (1) obviously commutes. It remains to show that $(\beta' = \beta'') \in \mathcal{M}_2^*$. Since I_{HG} creates \mathcal{M} -morphisms as already shown above, we have that for $\beta'' \in \mathcal{M}_2$ there is exactly one morphism $\beta^* \in \mathcal{M}_2^*$ such that $I_{\text{HG}}(\beta^*) = \beta''$, which implies directly that $\beta'' \in \mathcal{M}_2^*$.

$$\begin{array}{ccc} P & \xrightarrow{b} & P' \\ \alpha \downarrow & (1) & \downarrow \alpha' \\ C & \xrightarrow{\beta' \in \mathcal{M}_2^*} & C' \end{array} \qquad \begin{array}{ccc} I_{\text{HG}}(P) & \xrightarrow{I_{\text{HG}}(b)} & I_{\text{HG}}(P') \\ I_{\text{HG}}(\alpha) \downarrow & (2) & \downarrow \alpha'' = I_{\text{HG}}(\alpha') \\ I_{\text{HG}}(C) & \xrightarrow{(b'' = I_{\text{HG}}(\beta')) \in \mathcal{M}_2} & C'' = I_{\text{HG}}(C') \end{array}$$

□

In this appendix, we give the detailed proofs for indicated lemmas from the main part of this work concerning the PTI net application.

Lemma 42: (Well-Definedness of PTI Net Morphism Translation [207], see page 177)

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2)$, and the restricted functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\mathbf{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then for each PTI net morphism $f : \text{NI}_1 \rightarrow \text{NI}_2$ in \mathcal{M}_1 the corresponding morphism $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(\text{NI}_1) \rightarrow \mathcal{F}_{\text{PTI}}(\text{NI}_2)$ is well-defined in $\mathbf{AGraphs}_{\mathbf{PNTG}}$ with $\mathcal{F}_{\text{PTI}}(f) \in \mathcal{M}_2$. Moreover, \mathcal{F}_{PTI} preserves compositionality, injectivity of morphisms, inclusions, and identities.

Proof.

We have to show the following five steps:

1. The components of $\mathcal{F}_{\text{PTI}}(f)$ are well-defined w.r.t. the codomain.
2. The components of $\mathcal{F}_{\text{PTI}}(f)$ are compatible with the source and target functions.
3. The components of $\mathcal{F}_{\text{PTI}}(f)$ are compatible with the typing morphisms.
4. Compositionality axiom holds for \mathcal{F}_{PTI} .
5. $f \in \mathcal{M}_1$ (inclusion, identity) implies that $\mathcal{F}_{\text{PTI}}(f) \in \mathcal{M}_2$ (inclusion, identity).

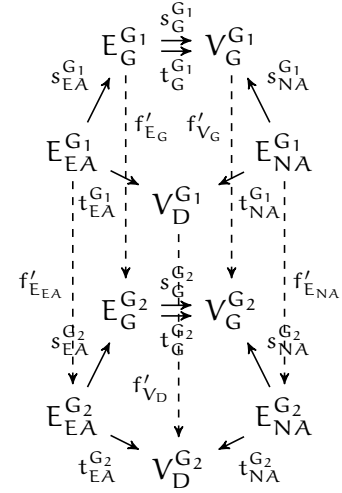
Let $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(\text{NI}_1) \rightarrow \mathcal{F}_{\text{PTI}}(\text{NI}_2)$ be defined by $\mathcal{F}_{\text{PTI}}(f) = f' = (f'_{V_G}, f'_{V_D} = \text{id}_{\mathbb{N}}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}})$, where in the short notation $\mathcal{F}_{\text{PTI}}(\text{NI}_i) = (V_G^{G_i}, \mathbb{N}, E_G^{G_i}, E_{NA}^{G_i}, E_{EA}^{G_i}, (s_j^{G_i}, t_j^{G_i})_{j \in \{G, NA, EA\}})_{i \in \{1, 2\}}$.

1. $f'_{V_G}, f'_{V_D}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}}$ are well-defined w.r.t. the codomain.
 - a) f'_{V_G} is well-defined, i.e., $f'_{V_G}(x) \in V_G^{G_2}$ for $x \in V_G^{G_1}$:
 - Case 1: Let $x \in V_G^{G_1}$ with $x \in I_1$.
 $f'_{V_G}(x) = f_I(x) \in I_2$, because $f = (f_P, f_T, f_I) \in \mathcal{M}_1$ is a **PTINet**-morphism
 $\Rightarrow f_I(x) \in V_G^{G_2}$.
 - Case 2: Let $x \in V_G^{G_1}$ with $x \in P_1$: similarly to Case 1 replacing I by P .
 - Case 3: Let $x \in V_G^{G_1}$ with $x \in T_1$: similarly to Case 1 replacing I by T .
 - b) f'_{V_D} is well-defined, i.e., $f'_{V_D}(i) \in V_D^{G_2}$ for $i \in V_D^{G_1}$:
 - Let $i \in V_D^{G_1}$ with $V_D^{G_1} = \mathbb{N}$.
 $f'_{V_D}(i) = i \in \mathbb{N}$
 $\Rightarrow i \in V_D^{G_2}$.
 - c) f'_{E_G} is well-defined, i.e., $f'_{E_G}(x, y) \in E_G^{G_2}$ for $(x, y) \in E_G^{G_1} = E_{to2p}^{G_1} \uplus E_{p2t}^{G_1} \uplus E_{t2p}^{G_1}$:
 - Case 1: Let $(x, p) \in E_{to2p}^{G_1}$ with $x \in I_1, p \in P_1$.
 $f'_{E_G}(x, p) = (f_I(x), f_P(p))$ with $f_I(x) \in I_2$ and $f_P(p) \in P_2$
 $\Rightarrow m_2(f_I(x)) = f_P(p)$, because $f = (f_P, f_T, f_I) \in \mathcal{M}_1$ is a **PTINet**-morphism
 $\Rightarrow (f_I(x), f_P(p)) \in E_{to2p}^{G_2} \subseteq E_G^{G_2}$.
 - Case 2: Let $(p, t) \in E_{p2t}^{G_1}$ with $p \in P_1, t \in T_1$.
 $f'_{E_G}(p, t) = (f_P(p), f_T(t))$ with $f_P(p) \in P_2$ and $f_T(t) \in T_2$
 $\Rightarrow \text{pre}(f_T(t))(f_P(p)) > 0$, i.e., there exists an edge between $f_P(p)$ and $f_T(t)$ in G_2 , because $f = (f_P, f_T, f_I) \in \mathcal{M}_1$ is a **PTINet**-morphism
 $\Rightarrow (f_P(p), f_T(t)) \in E_{p2t}^{G_2} \subseteq E_G^{G_2}$.

- Case 3: Let $(t, p) \in E_{t_2 p}^{G_1}$ with $t \in T_1, p \in P_1$: similarly to Case 2 replacing pre by post.
 - d) $f'_{E_{NA}}$ is well-defined, i.e., $f'_{E_{NA}}(x, y, z) \in E_{NA}^{G_2}$ for $(x, y, z) \in E_{NA}^{G_1} = E_{in}^{G_1} \uplus E_{out}^{G_1}$:
 - Case 1: Let $(t, n, in) \in E_{in}^{G_1}$ with $t \in T_1, n \in \mathbb{N}$.
 $f'_{E_{NA}}(t, n, in) = (f_T(t), n, in)$ with $f_T(t) \in T_2$
 $\Rightarrow |\bullet f_T(t)| = n$, because $f = (f_P, f_T, f_I) \in \mathcal{M}_1$ is a **PTINet**-morphism
 $\Rightarrow (f_T(t), n, in) \in E_{in}^{G_2} \subseteq E_{NA}^{G_2}$.
 - Case 2: Let $(t, n, out) \in E_{out}^{G_1}$ with $t \in T_1, n \in \mathbb{N}$: similarly to Case 1 replacing in by out and $\bullet f_T(t)$ by $f_T(t)\bullet$.
 - e) $f'_{E_{EA}}$ is well-defined, i.e., $f'_{E_{EA}}(x, y, z) \in E_{EA}^{G_2}$ for $(x, y, z) \in E_{EA}^{G_1} = E_{w_{pre}}^{G_1} \uplus E_{w_{post}}^{G_1}$:
 - Case 1: Let $(p, t, n) \in E_{w_{pre}}^{G_1}$ with $p \in P_1, t \in T_1, n \in \mathbb{N}$.
 $f'_{E_{EA}}(p, t, n) = (f_P(p), f_T(t), n)$ with $f_P(p) \in P_2$ and $f_T(t) \in T_2$
 $\Rightarrow \text{pre}(f_T(t))(f_P(p)) = n$, because $f = (f_P, f_T, f_I) \in \mathcal{M}_1$ is a **PTINet**-morphism
 $\Rightarrow (f_P(p), f_T(t), n) \in E_{w_{pre}}^{G_2} \subseteq E_{EA}^{G_2}$.
 - Case 2: Let $(t, p, n) \in E_{w_{post}}^{G_1}$ with $t \in T_1, p \in P_1, n \in \mathbb{N}$: similarly to Case 1 replacing pre by post.
2. $f'_{V_G}, f'_{V_D}, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}}$ are compatible with the source and target functions.

To show:

- a) $f'_{V_D} \circ t_{EA}^{G_1} = t_{EA}^{G_2} \circ f'_{E_{EA}}$,
- b) $t_{NA}^{G_2} \circ f'_{E_{NA}} = f'_{V_D} \circ t_{NA}^{G_1}$,
- c) $s_{NA}^{G_2} \circ f'_{E_{NA}} = f'_{V_G} \circ s_{NA}^{G_1}$,
- d) $s_{EA}^{G_2} \circ f'_{E_{EA}} = f'_{E_G} \circ s_{EA}^{G_1}$,
- e) $s_G^{G_2} \circ f'_{E_G} = f'_{V_G} \circ s_G^{G_1}$,
- f) $t_G^{G_2} \circ f'_{E_G} = f'_{V_G} \circ t_G^{G_1}$.



Part 2a:

Case 1: Let $(p, t, n) \in E_{EA}^{G_1}$ with $p \in P_1$ and $t \in T_1$.

$$\begin{aligned} (f'_{V_D} \circ t_{EA}^{G_1})(p, t, n) &= f'_{V_D}(t_{EA}^{G_1}(p, t, n)) = f'_{V_D}(n) = n = t_{EA}^{G_2}(f_P(p), f_T(t), n) \\ &= t_{EA}^{G_2}(f'_{E_{EA}}(p, t, n)) = (t_{EA}^{G_2} \circ f'_{E_{EA}})(p, t, n) \end{aligned}$$

Case 2: Let $(t, p, n) \in E_{EA}^{G_1}$ with $t \in T_1$ and $p \in P_1$: similarly to Case 1 replacing p by t and t by p.

Part 2b:

Let $(t, n, i) \in E_{NA}^{G_1}$ with $i \in \{in, out\}$.

$$\begin{aligned} (t_{NA}^{G_2} \circ f'_{E_{NA}})(t, n, i) &= t_{NA}^{G_2}(f'_{E_{NA}}(t, n, i)) = t_{NA}^{G_2}(f_T(t), n, i) = n = f'_{V_D}(n) \\ &= f'_{V_D}(t_{NA}^{G_1}(t, n, i)) = (f'_{V_D} \circ t_{NA}^{G_1})(t, n, i) \end{aligned}$$

Part 2c:

Let $(t, n, i) \in E_{NA}^{G_1}$ with $i \in \{in, out\}$.

$$\begin{aligned} (s_{NA}^{G_2} \circ f'_{E_{NA}})(t, n, i) &= s_{NA}^{G_2}(f'_{E_{NA}}(t, n, i)) = s_{NA}^{G_2}(f_T(t), n, i) = f_T(t) = f'_{V_G}(t) \\ &= f'_{V_G}(s_{NA}^{G_1}(t, n, i)) = (f'_{V_G} \circ s_{NA}^{G_1})(t, n, i) \end{aligned}$$

Part 2d:

Case 1: Let $(p, t, n) \in E_{EA}^{G_1}$ with $p \in P_1$ and $t \in T_1$.

$$\begin{aligned} (s_{EA}^{G_2} \circ f'_{EA})(p, t, n) &= s_{EA}^{G_2}(f'_{EA}(p, t, n)) = s_{EA}^{G_2}(f_P(p), f_T(t), n) = (f_P(p), f_T(t)) \\ &= f'_{EG}(p, t) = f'_{EG}(s_{EA}^{G_1}(p, t, n)) = (f'_{EG} \circ s_{EA}^{G_1})(p, t, n) \end{aligned}$$

Case 2: Let $(t, p, n) \in E_{EA}^{G_1}$ with $t \in T_1$ and $p \in P_1$: similarly to Case 1 replacing p by t and t by p .

Part 2e:

Case 1: Let $(x, p) \in E_G^{G_1}$ with $x \in I_1$ and $p \in P_1$.

$$\begin{aligned} (s_G^{G_2} \circ f'_{EG})(x, p) &= s_G^{G_2}(f'_{EG}(x, p)) = s_G^{G_2}(f_I(x), f_P(p)) = f_I(x) = f'_{VG}(x) \\ &= f'_{VG}(s_G^{G_1}(x, p)) = (f'_{VG} \circ s_G^{G_1})(x, p) \end{aligned}$$

Case 2: Let $(p, t) \in E_G^{G_1}$ with $p \in P_1$ and $t \in T_1$: similarly to Case 1 replacing x by p and p by t .

Case 3: Let $(t, p) \in E_G^{G_1}$ with $t \in T_1$ and $p \in P_1$: similarly to Case 1 replacing x by t .

Part 2f:

Case 1: Let $(x, p) \in E_G^{G_1}$ with $x \in I_1$ and $p \in P_1$.

$$\begin{aligned} (t_G^{G_2} \circ f'_{EG})(x, p) &= t_G^{G_2}(f'_{EG}(x, p)) = t_G^{G_2}(f_I(x), f_P(p)) = f_P(p) = f'_{VG}(p) \\ &= f'_{VG}(t_G^{G_1}(x, p)) = (f'_{VG} \circ t_G^{G_1})(x, p) \end{aligned}$$

Case 2: Let $(p, t) \in E_G^{G_1}$ with $p \in P_1$ and $t \in T_1$: similarly to Case 1 replacing x by p and p by t .

Case 3: Let $(t, p) \in E_G^{G_1}$ with $t \in T_1$ and $p \in P_1$: similarly to Case 1 replacing x by t .

3. $f'_{VG}, f'_{VD}, f'_{EG}, f'_{ENA}, f'_{EEA}$ are compatible with the typing morphisms.

Consider $\mathcal{F}_{PTI}(NI_1) = ((G_1, NAT), type^{G_1})$ and $\mathcal{F}_{PTI}(NI_2) = ((G_2, NAT), type^{G_2})$.

To show: $type^{G_2} \circ \mathcal{F}_{PTI}(f) = type^{G_1}$ with $type^{G_i} = (type_{VG}^{G_i}, type_{VD}^{G_i}, type_{EG}^{G_i}, type_{ENA}^{G_i}, type_{EEA}^{G_i})$ where $i \in \{1, 2\}$ and $\mathcal{F}_{PTI}(f) = f' = (f'_{VG}, f'_{VD}, f'_{EG}, f'_{ENA}, f'_{EEA})$.

Or in particular:

- a) $type_{VG}^{G_2} \circ f'_{VG} = type_{VG}^{G_1}$,
- b) $type_{VD}^{G_2} \circ f'_{VD} = type_{VD}^{G_1}$,
- c) $type_{EG}^{G_2} \circ f'_{EG} = type_{EG}^{G_1}$,
- d) $type_{ENA}^{G_2} \circ f'_{ENA} = type_{ENA}^{G_1}$,
- e) $type_{EEA}^{G_2} \circ f'_{EEA} = type_{EEA}^{G_1}$.

$$\begin{array}{ccc} \mathcal{F}_{PTI}(NI_1) & \xrightarrow{\mathcal{F}_{PTI}(f)} & \mathcal{F}_{PTI}(NI_2) \\ & \searrow \quad \swarrow & \\ & type^{G_1} \quad type^{G_2} & \\ & \text{PNTG} & \end{array}$$

Part 3a:

Case 1: Let $p \in V_G^{G_1}$ with $p \in P_1$.

$$(type_{VG}^{G_2} \circ f'_{VG})(p) = type_{VG}^{G_2}(f'_{VG}(p)) = type_{VG}^{G_2}(f_P(p)) = Place = type_{VG}^{G_1}(p)$$

Case 2: Let $t \in V_G^{G_1}$ with $t \in T_1$: similarly to Case 1 replacing p by t .

Case 3: Let $x \in V_G^{G_1}$ with $x \in I_1$: similarly to Case 1 replacing p by x .

Part 3b:

Let $i \in \mathbb{N}$.

$$(type_{VD}^{G_2} \circ f'_{VD})(i) = type_{VD}^{G_2}(f'_{VD}(i)) = type_{VD}^{G_2}(i) = nat = type_{VD}^{G_1}(i)$$

Part 3c:

Case 1: Let $(x, p) \in E_G^{G_1}$ with $x \in I_1$ and $p \in P_1$.

$$\begin{aligned} (\text{type}_{E_G}^{G_2} \circ f'_{E_G})(x, p) &= \text{type}_{E_G}^{G_2}(f'_{E_G}(x, p)) = \text{type}_{E_G}^{G_2}(f_I(x), f_P(p)) \\ &= \text{token2place} = \text{type}_{E_G}^{G_1}(x, p) \end{aligned}$$

Case 2: Let $(p, t) \in E_G^{G_1}$ with $p \in P_1$ and $t \in T_1$: similarly to Case 1 replacing x by p and p by t .

Case 3: Let $(t, p) \in E_G^{G_1}$ with $t \in T_1$ and $p \in P_1$: similarly to Case 1 replacing x by t .

Part 3d:

Case 1: Let $(t, n, in) \in E_{NA}^{G_1}$.

$$\begin{aligned} (\text{type}_{E_{NA}}^{G_2} \circ f'_{E_{NA}})(t, n, in) &= \text{type}_{E_{NA}}^{G_2}(f'_{E_{NA}}(t, n, in)) = \text{type}_{E_{NA}}^{G_2}(f_T(t), n, in) \\ &= in = \text{type}_{E_{NA}}^{G_1}(t, n, in) \end{aligned}$$

Case 2: Let $(t, n, out) \in E_{NA}^{G_1}$: similarly to Case 1 replacing in by out .

Part 3e:

Case 1: Let $(p, t, n) \in E_{EA}^{G_1}$ with $p \in P_1$ and $t \in T_1$.

$$\begin{aligned} (\text{type}_{E_{EA}}^{G_2} \circ f'_{E_{EA}})(p, t, n) &= \text{type}_{E_{EA}}^{G_2}(f'_{E_{EA}}(p, t, n)) = \text{type}_{E_{EA}}^{G_2}(f_P(p), f_T(t), n) \\ &= \text{weight}_{pre} = \text{type}_{E_{EA}}^{G_1}(p, t, n) \end{aligned}$$

Case 2: Let $(t, p, n) \in E_{EA}^{G_1}$ with $t \in T_1$ and $p \in P_1$: similarly to Case 1 replacing p by t and t by p .

4. Compositionality axiom holds for \mathcal{F}_{PTI} .

Consider PTI net morphisms $f : NI_2 \rightarrow NI_3$ and $g : NI_1 \rightarrow NI_2$.

To show: $\mathcal{F}_{PTI}(f \circ g) = \mathcal{F}_{PTI}(f) \circ \mathcal{F}_{PTI}(g)$.

- We consider the V_G -component of some typed attributed graph morphism.

Case 1: Let $p \in P_1$.

$$\begin{aligned} &\mathcal{F}_{PTI}(f \circ g)_{V_G}(p) \\ &= ((f \circ g)_P \uplus (f \circ g)_T \uplus (f \circ g)_I)(p) \\ &= (f \circ g)_P(p) \\ &\stackrel{\text{Def.}}{=} (f_P \circ g_P)(p) \\ &= ((f_P \uplus f_T \uplus f_I) \circ (g_P \uplus g_T \uplus g_I))(p) \\ &= (\mathcal{F}_{PTI}(f)_{V_G} \circ \mathcal{F}_{PTI}(g)_{V_G})(p) \\ &\stackrel{\text{Def.}}{=} (\mathcal{F}_{PTI}(f) \circ \mathcal{F}_{PTI}(g))_{V_G}(p) \end{aligned}$$

Case 2: Let $t \in T_1$. The proof works similar to the Case 1.

Case 2: Let $x \in I_1$. The proof works similar to the Case 1.

- We consider the V_D -component of some typed attributed graph morphism.

Let $x \in I_1$.

$$\begin{aligned} &\mathcal{F}_{PTI}(f \circ g)_{V_D}(x) \\ &= id_N(x) \\ &= (id_N \circ id_N)(x) \\ &= (\mathcal{F}_{PTI}(f)_{V_D} \circ \mathcal{F}_{PTI}(g)_{V_D})(x) \\ &= (\mathcal{F}_{PTI}(f) \circ \mathcal{F}_{PTI}(g))_{V_D}(x) \end{aligned}$$

- We consider the E_G -component of some typed attributed graph morphism.

Case 1: Let $(p, t) \in E_{p2t}^{G_1}$.

$$\begin{aligned}
 & \mathcal{F}_{PTI}(f \circ g)_{E_G}(p, t) \\
 &= ((f \circ g)_P(p), (f \circ g)_T(t)) \\
 &\stackrel{Def.}{=} ((f_P \circ g_P)(p), (f_T \circ g_T)(t)) \\
 &= (f_P(g_P(p)), f_T(g_T(t))) \\
 &= \mathcal{F}_{PTI}(f)_{E_G}(g_P(p), g_T(t)) \\
 &= \mathcal{F}_{PTI}(f)_{E_G}(\mathcal{F}_{PTI}(g)_{E_G}(p, t)) \\
 &= (\mathcal{F}_{PTI}(f)_{E_G} \circ \mathcal{F}_{PTI}(g)_{E_G})(p, t) \\
 &\stackrel{Def.}{=} (\mathcal{F}_{PTI}(f) \circ \mathcal{F}_{PTI}(g))_{E_G}(p, t)
 \end{aligned}$$

Case 2: Let $(t, p) \in E_{t2p}^{G_1}$. The proof is similar to the Case 1 replacing p by t and t by p .

Case 3: Let $(x, p) \in E_{to2p}^{G_1}$. The proof is similar to the Case 1 replacing p by x and t by p .

- We consider the E_{NA} -component of some typed attributed graph morphism.

Let $(t, n, x) \in E_{in}^{G_1} \uplus E_{out}^{G_1}$.

$$\begin{aligned}
 & \mathcal{F}_{PTI}(f \circ g)_{E_{NA}}(t, n, x) \\
 &= ((f \circ g)_T(t), n, x) \\
 &\stackrel{Def.}{=} ((f_T \circ g_T)(t), n, x) \\
 &= (f_T(g_T(t)), n, x) \\
 &= \mathcal{F}_{PTI}(f)_{E_{NA}}(g_T(t), n, x) \\
 &= \mathcal{F}_{PTI}(f)_{E_{NA}}(\mathcal{F}_{PTI}(g)_{E_{NA}}(t, n, x)) \\
 &= (\mathcal{F}_{PTI}(f)_{E_{NA}} \circ \mathcal{F}_{PTI}(g)_{E_{NA}})(t, n, x) \\
 &\stackrel{Def.}{=} (\mathcal{F}_{PTI}(f) \circ \mathcal{F}_{PTI}(g))_{E_{NA}}(t, n, x)
 \end{aligned}$$

- We consider the E_{EA} -component of some typed attributed graph morphism.

Case 1: Let $(p, t, n) \in E_{wpre}^{G_1}$.

$$\begin{aligned}
 & \mathcal{F}_{PTI}(f \circ g)_{E_{EA}}(p, t, n) \\
 &= ((f \circ g)_P(p), (f \circ g)_T(t), n) \\
 &\stackrel{Def.}{=} ((f_P \circ g_P)(p), (f_T \circ g_T)(t), n) \\
 &= (f_P(g_P(p)), f_T(g_T(t)), n) \\
 &= \mathcal{F}_{PTI}(f)_{E_{EA}}(g_P(p), g_T(t), n) \\
 &= \mathcal{F}_{PTI}(f)_{E_{EA}}(\mathcal{F}_{PTI}(g)_{E_{EA}}(p, t, n)) \\
 &= (\mathcal{F}_{PTI}(f)_{E_{EA}} \circ \mathcal{F}_{PTI}(g)_{E_{EA}})(p, t, n) \\
 &\stackrel{Def.}{=} (\mathcal{F}_{PTI}(f) \circ \mathcal{F}_{PTI}(g))_{E_{EA}}(p, t, n)
 \end{aligned}$$

Case 2: Let $(t, p, n) \in E_{wpost}^{G_1}$. The proof is similar to the Case 1 replacing p by t and t by p .

5. $f \in \mathcal{M}_1$ (**inclusion, identity**) implies that $\mathcal{F}_{\text{PTI}}(f) \in \mathcal{M}_2$ (**inclusion, identity**).

a) Consider injective PTI net morphisms $f_A : A \rightarrow A'$ and $f_B : B \rightarrow B'$. For the restricted \mathcal{M} -functor \mathcal{F}_{PTI} it follows directly that $f_A \uplus f_B : A \uplus B \rightarrow A' \uplus B'$ is also injective.

b) Consider inclusions (identities) $f_A : A \rightarrow A'$ with $f_A(a) = a$ and $f_B : B \rightarrow B'$ with $f_B(b) = b$.

To show: $f_A \uplus f_B : A \uplus B \rightarrow A' \uplus B'$ with $A \uplus B = (A \times \{1\} \cup B \times \{2\})$ and $A' \uplus B' = (A' \times \{1\} \cup B' \times \{2\})$ is an inclusion (identity).

Case 1: Let $i = 1$ and $x \in A \uplus B$:

$$(f_A \uplus f_B)(x, i) = (f_A \uplus f_B)(x, 1) = (f_A(x), 1) = (x, 1)$$

Case 2: Let $i = 2$ and $x \in A \uplus B$:

$$(f_A \uplus f_B)(x, i) = (f_A \uplus f_B)(x, 2) = (f_B(x), 2) = (x, 2)$$

□

Lemma 43: (\mathcal{F}_{PTI} Preserves Pushouts of Injective Morphisms [207], see page 179)

Consider \mathcal{M} -adhesive categories ($\mathbf{PTINet}, \mathcal{M}_1$) and ($\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2$), restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ introduced in Definition 64, PTI nets NI_i with PTI net morphisms $f_i = (f_{iP}, f_{iT}, f_{iI})$, and typed attributed graphs $\mathcal{F}_{\text{PTI}}(\text{NI}_i)$ with typed attributed graph morphisms $\mathcal{F}_{\text{PTI}}(f_i) = f'_i = (f'_{iV_G}, f'_{iV_D}, f'_{iE_G}, f'_{iE_{NA}}, f'_{iE_{EA}})$ for $i \in \{0, 1, 2, 3\}$. If (1) is a pushout in \mathbf{PTINet} with $f_i \in \mathcal{M}_1$ then we have that (2) is a pushout in $\mathbf{AGraphs}_{\text{PNTG}}$ with $\mathcal{F}_{\text{PTI}}(f_i) \in \mathcal{M}_2$ for $i \in \{0, 1, 2, 3\}$.

$$\begin{array}{ccc} \text{NI}_0 & \xrightarrow{f_1} & \text{NI}_1 \\ f_2 \downarrow & (1) & \downarrow f_4 \\ \text{NI}_2 & \xrightarrow{f_3} & \text{NI}_3 \end{array} \quad \begin{array}{ccc} \mathcal{F}_{\text{PTI}}(\text{NI}_0) & \xrightarrow{\mathcal{F}_{\text{PTI}}(f_1) = f'_1} & \mathcal{F}_{\text{PTI}}(\text{NI}_1) \\ \mathcal{F}_{\text{PTI}}(f_2) = f'_2 \downarrow & (2) & \downarrow \mathcal{F}_{\text{PTI}}(f_4) = f'_4 \\ \mathcal{F}_{\text{PTI}}(\text{NI}_2) & \xrightarrow{\mathcal{F}_{\text{PTI}}(f_3) = f'_3} & \mathcal{F}_{\text{PTI}}(\text{NI}_3) \end{array}$$

Proof.

Let (1) be a pushout in \mathbf{PTINet} with $f_i \in \mathcal{M}_1$ for $i \in \{0, 1, 2, 3\}$, i.e., P-, T- and I-components of (1) are pushouts in **Sets**, because pushouts in \mathbf{PTINet} are constructed componentwise according to [224].

$$\begin{array}{ccccc} P_0 & \xrightarrow{f_{0P} \text{ inj.}} & P_1 & & T_0 & \xrightarrow{f_{0T} \text{ inj.}} & T_1 & & I_0 & \xrightarrow{f_{0I} \text{ inj.}} & I_1 \\ f_{1P} \text{ inj.} \downarrow & (PO) & \downarrow f_{2P} \text{ inj.} & & f_{1T} \text{ inj.} \downarrow & (PO) & \downarrow f_{2T} \text{ inj.} & & f_{1I} \text{ inj.} \downarrow & (PO) & \downarrow f_{2I} \text{ inj.} \\ P_2 & \xrightarrow{f_{3P} \text{ inj.}} & P_3 & & T_2 & \xrightarrow{f_{3T} \text{ inj.}} & T_3 & & I_2 & \xrightarrow{f_{3I} \text{ inj.}} & I_3 \end{array}$$

To show: V_G^- , V_D^- , E_G^- , E_{NA}^- and E_{EA}^- -components of (2) are pushouts in **Sets** with $\mathcal{F}_{\text{PTI}}(f_i) = f'_i \in \mathcal{M}_2$ for $i \in \{0, 1, 2, 3\}$, because pushouts in $\mathbf{AGraphs}_{\text{PNTG}}$ are constructed componentwise as well.

For given morphisms $f_i \in \mathcal{M}_1$ we have that $\mathcal{F}_{\text{PTI}}(f_i) = f'_i \in \mathcal{M}_2$ for $i \in \{0, 1, 2, 3\}$ according to Lemma 42.

1. V_G -component of (2) is a pushout in **Sets** (see diagram (3)) with $f'_{iV_G} = f_{iP} \uplus f_{iT} \uplus f_{iI}$ for $i \in \{0, 1, 2, 3\}$, because pushouts are compatible with coproducts (and coproduct in **Sets** is the disjoint union \uplus).

$$\begin{array}{ccc} V_G^{G_0} & \xrightarrow{f_{0P} \uplus f_{0T} \uplus f_{0I}} & V_G^{G_1} \\ f_{1P} \uplus f_{1T} \uplus f_{1I} \downarrow & (3) & \downarrow f_{2P} \uplus f_{2T} \uplus f_{2I} \\ V_G^{G_2} & \xrightarrow{f_{3P} \uplus f_{3T} \uplus f_{3I}} & V_G^{G_3} \end{array}$$

2. V_D -component of (2) is obviously a pushout in **Sets** (see diagram (4)) with $f'_{iV_D} = \text{id}_N$ for $i \in \{0, 1, 2, 3\}$.

$$\begin{array}{ccc} V_D^{G_0} = N & \xrightarrow{\text{id}_N} & V_D^{G_1} = N \\ \text{id}_N \downarrow & (4) & \downarrow \text{id}_N \\ V_D^{G_2} = N & \xrightarrow{\text{id}_N} & V_D^{G_3} = N \end{array}$$

3. For the E_G -component we have to show that (5) is a pushout in **Sets** which holds if (5a), (5b) and (5c) are pushouts.

$$\begin{array}{ccc} E_G^{G_0} & \xrightarrow{f'_{0E_G}} & E_G^{G_1} \\ f'_{1E_G} \downarrow & (5) & \downarrow f'_{2E_G} \\ E_G^{G_2} & \xrightarrow{f'_{3E_G}} & E_G^{G_3} \end{array} \quad \begin{array}{ccc} E_{p2t}^{G_0} & \xrightarrow{f_{0P} \times f_{0T}} & E_{p2t}^{G_1} \\ f_{1P} \times f_{1T} \downarrow & (5a) & \downarrow f_{2P} \times f_{2T} \\ E_{p2t}^{G_2} & \xrightarrow{f_{3P} \times f_{3T}} & E_{p2t}^{G_3} \end{array}$$

$$\begin{array}{ccc} E_{t2p}^{G_0} & \xrightarrow{f_{0T} \times f_{0P}} & E_{t2p}^{G_1} \\ f_{1T} \times f_{1P} \downarrow & (5b) & \downarrow f_{2T} \times f_{2P} \\ E_{t2p}^{G_2} & \xrightarrow{f_{3T} \times f_{3P}} & E_{t2p}^{G_3} \end{array} \quad \begin{array}{ccc} E_{to2p}^{G_0} & \xrightarrow{f_{0I} \times f_{0P}} & E_{to2p}^{G_1} \\ f_{1I} \times f_{1P} \downarrow & (5c) & \downarrow f_{2I} \times f_{2P} \\ E_{to2p}^{G_2} & \xrightarrow{f_{3I} \times f_{3P}} & E_{to2p}^{G_3} \end{array}$$

Diagrams (5a), (5b) and (5c) commute, because for each product component we have a pushout in **Sets** by assumption. So it remains to show that (5a), (5b) and (5c) are pushouts, because products of pushouts are not necessarily pushouts (the merging morphisms are in general not jointly surjective).

- a) For diagram (5a) we have to show that the diagram (5a') is a pushout in **Sets** with

$$P_i \otimes T_i = \{(p, t) \in P_i \times T_i \mid \text{pre}_i(t)(p) > 0\} \text{ and } f_{iP} \otimes f_{iT} \text{ for } i \in \{0, 1, 2, 3\}.$$

$$\begin{array}{ccc} P_0 \otimes T_0 & \xrightarrow{f_{0P} \otimes f_{0T}} & P_1 \otimes T_1 \\ f_{1P} \otimes f_{1T} \downarrow & (5a') & \downarrow f_{2P} \otimes f_{2T} \\ P_2 \otimes T_2 & \xrightarrow{f_{3P} \otimes f_{3T}} & P_3 \otimes T_3 \end{array}$$

Since f_i are PTI net morphisms, we have that all $f_{iP} \otimes f_{iT}$ morphisms for $i \in \{0, 1, 2, 3\}$ are well-defined. Furthermore, we have that the components of (5a') are pushouts and pullbacks in **Sets**, because $f_i = (f_{iP}, f_{iT}, f_{iI}) \in \mathcal{M}_1$ for $i \in \{0, 1, 2, 3\}$ by assumption. Hence, also (5a') is a pullback, because pullbacks are compatible with products and it remains to show that $(f_{2P} \otimes f_{2T}, f_{3P} \otimes f_{3T})$ are jointly surjective.

Consider $(p_3, t_3) \in P_3 \otimes T_3$. The T-component of $(5a')$ is a pushouts in **Sets**, thus it holds that $t_1 \in T_1$ with $f_{2T}(t_1) = t_3$ (or $t_2 \in T_2$ with $f_{3T}(t_2) = t_3$). Without loss of generality we consider the first case. Since $(p_3, t_3) \in P_3 \otimes T_3$, we know that $\text{pre}_3(t_3)(p_3) > 0$. This implies by compatibility of f_2 with the corresponding pre-functions (see diagram below) that $f_{2P}^\oplus(\text{pre}_1(t_1)) = \text{pre}_3(f_{2T}(t_1)) = \text{pre}_3(t_3)$. Thus it also holds that $\text{pre}_3(t_3)(p_3) = (f_{2P}^\oplus(\text{pre}_1(t_1)))(p_3) > 0$ implying that there is $p_1 \in P_1$ s.t. $f_{2P}(p_1) = p_3$ and $\text{pre}_1(t_1)(p_1) > 0$, which by definition means that $(p_1, t_1) \in P_1 \otimes T_1$. Hence, we get that $(5a')$ and $(5a)$ are pushouts.

$$\begin{array}{ccc} T_1 & \xrightleftharpoons[\text{post}_1]{\text{pre}_1} & P_1^\oplus \\ f_{2T} \downarrow & \text{=} & \downarrow f_{2P}^\oplus \\ T_3 & \xrightleftharpoons[\text{post}_3]{\text{pre}_3} & P_3^\oplus \end{array}$$

b) For diagram $(5b)$ we have to show that the diagram $(5b')$ is a pushout in **Sets** with

$$T_i \otimes P_i = \{(t, p) \in T_i \times P_i \mid \text{post}_i(t)(p) > 0\} \text{ and } f_{iT} \otimes f_{iP} \text{ for } i \in \{0, 1, 2, 3\}.$$

$$\begin{array}{ccc} T_0 \otimes P_0 & \xrightarrow{f_{0T} \otimes f_{0P}} & T_1 \otimes P_1 \\ f_{1T} \otimes f_{1P} \downarrow & (5b') & \downarrow f_{2T} \otimes f_{2P} \\ T_2 \otimes P_2 & \xrightarrow{f_{3T} \otimes f_{3P}} & T_3 \otimes P_3 \end{array}$$

The proof for this case is similar to the Case 3a. Hence, $(5b')$ and $(5b)$ are pushouts.

c) For diagram $(5c)$ we have to show that the diagram $(5c')$ is a pushout in **Sets** with

$$I_i \otimes P_i = \{(x, p) \in I_i \times P_i \mid m_i(x) = p\} \text{ and } f_{iI} \otimes f_{iP} \text{ for } i \in \{0, 1, 2, 3\}.$$

$$\begin{array}{ccc} I_0 \otimes P_0 & \xrightarrow{f_{0I} \otimes f_{0P}} & I_1 \otimes P_1 \\ f_{1I} \otimes f_{1P} \downarrow & (5c') & \downarrow f_{2I} \otimes f_{2P} \\ I_2 \otimes P_2 & \xrightarrow{f_{3I} \otimes f_{3P}} & I_3 \otimes P_3 \end{array}$$

Since f_i are PTI net morphisms, we have that all $f_{iI} \otimes f_{iP}$ morphisms for $i \in \{0, 1, 2, 3\}$ are well-defined. Furthermore, we have that the components of $(5c')$ are pushouts and pullbacks in **Sets**, because $f_i = (f_{iP}, f_{iT}, f_{iI}) \in \mathcal{M}_1$ for $i \in \{0, 1, 2, 3\}$ by assumption. Hence, also $(5c')$ is a pullback, because pullbacks are compatible with products and it remains to show that $(f_{2I} \otimes f_{2P}, f_{3I} \otimes f_{3P})$ are jointly surjective.

Consider $(x_3, p_3) \in I_3 \otimes P_3$. The I-component of $(5c')$ is a pushouts in **Sets** s.t. we have $x_1 \in I_1$ with $f_{2I}(x_1) = x_3$ (or we have $x_2 \in I_2$ with $f_{3I}(x_2) = x_3$). Without loss of generality we consider the first case. Let $p_1 = m_1(x_1)$, then $(f_{2I} \otimes f_{2P})(x_1, p_1) = (x_3, p_3)$, because $f_2 = (f_{2P}, f_{2T}, f_{2I})$ is a PTI net morphism, i.e., the compatibility with the corresponding marking functions holds s.t. we have $m_3(x_3) = p_3$ (see the diagram below). Hence, $(5c')$ and $(5c)$ are pushouts.

$$\begin{array}{ccc} I_1 & \xrightarrow{m_1} & P_1 \\ f_{2I} \downarrow & \text{=} & \downarrow f_{2P} \\ I_3 & \xrightarrow{m_3} & P_3 \end{array}$$

4. For the E_{NA} -component we have to show that (6) is a pushout in **Sets** which holds if (6a) and similarly (6b) are pushouts for $X = \{in\}$ and $Y = \{out\}$.

$$\begin{array}{ccccc}
 E_{NA}^{G_0} & \xrightarrow{f'_{0E_{NA}}} & E_{NA}^{G_1} \\
 f'_{1E_{NA}} \downarrow & (6) & \downarrow f'_{2E_{NA}} \\
 E_{NA}^{G_2} & \xrightarrow{f'_{3E_{NA}}} & E_{NA}^{G_3} \\
 \begin{array}{ccc}
 E_{in}^{G_0} & \xrightarrow{f_{0T} \times id_N \times id_X} & E_{in}^{G_1} \\
 f_{1T} \times id_N \times id_X \downarrow & (6a) & \downarrow f_{2T} \times id_N \times id_X \\
 E_{in}^{G_2} & \xrightarrow{f_{3T} \times id_N \times id_X} & E_{in}^{G_3}
 \end{array} & & \begin{array}{ccc}
 E_{out}^{G_0} & \xrightarrow{f_{0T} \times id_N \times id_Y} & E_{out}^{G_1} \\
 f_{1T} \times id_N \times id_Y \downarrow & (6b) & \downarrow f_{2T} \times id_N \times id_Y \\
 E_{out}^{G_2} & \xrightarrow{f_{3T} \times id_N \times id_Y} & E_{out}^{G_3}
 \end{array}
 \end{array}$$

Diagrams (6a) and (6b) commute, because for each product component we have a pushout in **Sets** by assumption. So it remains to show that (6a) and (6b) are pushouts.

- a) For diagram (6a) we have to show that the diagram (6a') is a pushout in **Sets** with

$$T_i \otimes \mathbb{N} \otimes X = \{(t, n, in) \mid (t, n) \in T_i \times \mathbb{N} \wedge |\bullet t| = n\}$$

and $f_{iT} \otimes id_N \otimes id_X$ for $i \in \{0, 1, 2, 3\}$.

$$\begin{array}{ccc}
 T_0 \otimes \mathbb{N} \otimes X & \xrightarrow{f_{0T} \otimes id_N \otimes id_X} & T_1 \otimes \mathbb{N} \otimes X \\
 f_{1T} \otimes id_N \otimes id_X \downarrow & (6a') & \downarrow f_{2T} \otimes id_N \otimes id_X \\
 T_2 \otimes \mathbb{N} \otimes X & \xrightarrow{f_{3T} \otimes id_N \otimes id_X} & T_3 \otimes \mathbb{N} \otimes X
 \end{array}$$

Also here we know that $f_{iT} \otimes id_N \otimes id_X$ morphisms for $i \in \{0, 1, 2, 3\}$ are well-defined, because f_i are PTI net morphisms. Furthermore, we have that the components of (6a') are pushouts and pullbacks in **Sets**, because $f_i = (f_{iP}, f_{iT}, f_{iI}) \in \mathcal{M}_1$ for $i \in \{0, 1, 2, 3\}$ by assumption. Hence, also (6a') is a pullback and we have that $(f_{2T} \otimes id_N \otimes id_X, f_{3T} \otimes id_N \otimes id_X)$ are obviously jointly surjective. Therefore, (6a') and (6a) are pushouts.

- b) For diagram (6b) we have to show that the diagram (6b') is a pushout in **Sets** with

$$T_i \otimes \mathbb{N} \otimes Y = \{(t, n, out) \mid (t, n) \in T_i \times \mathbb{N} \wedge |t \bullet| = n\}$$

and $f_{iT} \otimes id_N \otimes id_Y$ for $i \in \{0, 1, 2, 3\}$.

$$\begin{array}{ccc}
 T_0 \otimes \mathbb{N} \otimes Y & \xrightarrow{f_{0T} \otimes id_N \otimes id_Y} & T_1 \otimes \mathbb{N} \otimes Y \\
 f_{1T} \otimes id_N \otimes id_Y \downarrow & (6b') & \downarrow f_{2T} \otimes id_N \otimes id_Y \\
 T_2 \otimes \mathbb{N} \otimes Y & \xrightarrow{f_{3T} \otimes id_N \otimes id_Y} & T_3 \otimes \mathbb{N} \otimes Y
 \end{array}$$

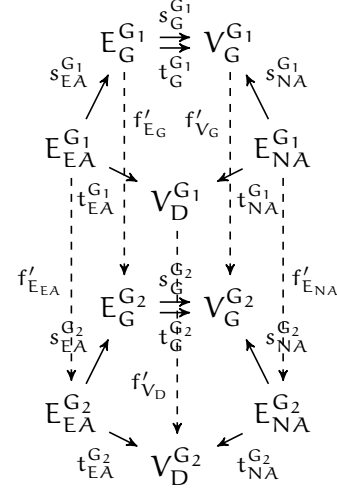
The proof for this case is similar to the Case 4a. Hence, (6b') and (6b) are pushouts.

5. For the E_{EA} -component we have to show that (7) is a pushout in **Sets** which holds if (7a) and similarly (7b) are pushouts.

$\text{id}_N, f'_{E_G}, f'_{E_{NA}}, f'_{E_{EA}}$ given according to the diagram below and $f_P(p) = f'_{V_G}(p)$ for $p \in P_1$, $f_T(t) = f'_{V_G}(t)$ for $t \in T_1$, $f_I(x) = f'_{V_G}(x)$ for $x \in I_1$.

To show:

1. $f'_{E_G}(x, p) = (f_I(x), f_P(p))$ for $(x, p) \in E_{1 \text{ to } 2P}$,
2. $f'_{E_G}(p, t) = (f_P(p), f_T(t))$ for $(p, t) \in E_{1P \text{ to } 2T}$,
3. $f'_{E_G}(t, p) = (f_T(t), f_P(p))$ for $(t, p) \in E_{1T \text{ to } 2P}$,
4. $f'_{E_{NA}}(t, n, x) = (f_T(t), n, x)$ for $(t, n, x) \in E_{1 \text{ in}} \uplus E_{1 \text{ out}}$,
5. $f'_{E_{EA}}(p, t, n) = (f_P(p), f_T(t), n)$ for $(p, t, n) \in E_{1 \text{ wpre}}$,
6. $f'_{E_{EA}}(t, p, n) = (f_T(t), f_P(p), n)$ for $(t, p, n) \in E_{1 \text{ wpost}}$.



Part 1:

Let $f'_{E_G}(x, p) = (x', p')$ with $s_{2G}(x', p') = x'$ and $t_{2G}(x', p') = p'$. Then we have:

$$\begin{aligned} x' &= s_{2G}(x', p') = s_{2G}(f'_{E_G}(x, p)) = f'_{V_G}(s_{1G}(x, p)) = f'_{V_G}(x) = f_I(x) \\ &\Rightarrow x' = f_I(x), \\ p' &= t_{2G}(x', p') = t_{2G}(f'_{E_G}(x, p)) = f'_{V_G}(t_{1G}(x, p)) = f'_{V_G}(p) = f_P(p) \\ &\Rightarrow p' = f_P(p). \end{aligned}$$

This implies altogether that $f'_{E_G}(x, p) = (x', p') = (f_I(x), f_P(p))$.

Part 2:

Let $f'_{E_G}(p, t) = (p', t')$ with $s_{2G}(p', t') = p'$ and $t_{2G}(p', t') = t'$. Then we have:

$$\begin{aligned} p' &= s_{2G}(p', t') = s_{2G}(f'_{E_G}(p, t)) = f'_{V_G}(s_{1G}(p, t)) = f'_{V_G}(p) = f_P(p) \\ &\Rightarrow p' = f_P(p), \\ t' &= t_{2G}(p', t') = t_{2G}(f'_{E_G}(p, t)) = f'_{V_G}(t_{1G}(p, t)) = f'_{V_G}(t) = f_T(t) \\ &\Rightarrow t' = f_T(t). \end{aligned}$$

This implies altogether that $f'_{E_G}(p, t) = (p', t') = (f_P(p), f_T(t))$.

Part 3:

Similar to Part 2 replacing p by t and t by p .

Part 4:

Let $f'_{E_{NA}}(t, n, x) = (t', n', x')$ with $s_{2NA}(t', n', x') = t'$ and $x = x'$ by compatibility with typing morphisms. Then we have:

$$\begin{aligned} t' &= s_{2NA}(t', n', x') = s_{2NA}(f'_{E_{NA}}(t, n, x)) = f'_{V_G}(s_{1NA}(t, n, x)) = f'_{V_G}(t) = f_T(t) \\ &\Rightarrow t' = f_T(t), \\ n' &= t_{2NA}(t', n', x') = t_{2NA}(f'_{E_{NA}}(t, n, x)) = f'_{V_D}(t_{1NA}(t, n, x)) \\ &= \text{id}_N(t_{1NA}(t, n, x)) = \text{id}_N(n) = n \\ &\Rightarrow n' = n. \end{aligned}$$

This implies altogether that $f'_{E_{NA}}(t, n, x) = (t', n', x') = (f_T(t), n, x)$.

Part 5:

Let $f'_{E_{EA}}(p, t, n) = (p', t', n')$ with $s_{2EA}(p', t', n') = (p', t')$. Then we have:

$$\begin{aligned}
p' &= s_{2G}(s_{2EA}(p', t', n')) = s_{2G}(p', t') \\
&\xRightarrow{\text{Part 2}} p' = f_P(p), \\
t' &= t_{2G}(s_{2EA}(p', t', n')) = t_{2G}(p', t') \\
&\xRightarrow{\text{Part 2}} t' = f_T(t), \\
n' &= t_{2EA}(p', t', n') = t_{2EA}(f'_{EA}(p, t, n)) = f'_{VD}(t_{1EA}(p, t, n)) \\
&= \text{id}_N(t_{1EA}(p, t, n)) = \text{id}_N(n) = n \\
&\Rightarrow n' = n.
\end{aligned}$$

This implies altogether that $f'_{EA}(p, t, n) = (p', t', n') = (f_P(p), f_T(t), n)$.

Part 6:

Similar to Part 5 replacing p by t and t by p . □

Lemma 45: (\mathcal{F}_{PTI} Creates Injective Morphisms [207], see page 185)

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2)$, typed attributed graphs $\mathcal{F}_{PTI}(NI_1)$, $\mathcal{F}_{PTI}(NI_2)$, and an injective typed attributed graph morphism $f' : \mathcal{F}_{PTI}(NI_1) \rightarrow \mathcal{F}_{PTI}(NI_2)$ that is compatible with typing morphisms. Then the restricted \mathcal{M} -functor $\mathcal{F}_{PTI} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\mathbf{PNTG}}|_{\mathcal{M}_2}$ creates a unique injective PTI net morphism $f : NI_1 \rightarrow NI_2$ such that $\mathcal{F}_{PTI}(f) = f'$ or formally written:

$$\exists! f : NI_1 \rightarrow NI_2 \text{ in } \mathcal{M}_1. \mathcal{F}_{PTI}(f) = f'.$$

Proof.

- **Construction of $f : NI_1 \rightarrow NI_2$:**

Consider an injective typed attributed graph morphism $f' : \mathcal{F}_{PTI}(NI_1) \rightarrow \mathcal{F}_{PTI}(NI_2)$ with $\mathcal{F}_{PTI}(NI_i) = (V_{iG}, V_{iD} = \mathbb{N}, E_{iG}, E_{iNA}, E_{iEA}, (s_{ij}, t_{ij})_{j \in \{G, NA, EA\}})$ for $i \in \{1, 2\}$ by $f' = (f'_{VG}, f'_{VD}, f'_{EG}, f'_{ENA}, f'_{EEA})$ with injective components

$$\begin{aligned}
f'_{VG} &: V_{1G} \rightarrow V_{2G} \text{ with } V_{iG} = P_i \uplus T_i \uplus I_i \text{ for } i \in \{1, 2\}, \\
f'_{VD} &: \mathbb{N} \rightarrow \mathbb{N} \text{ with } f'_{VD} = \text{id}_{\mathbb{N}}, \\
f'_{EG} &: E_{1G} \rightarrow E_{2G} \text{ with } E_{iG} = E_{i \text{ to } 2p} \uplus E_{i \text{ p } 2t} \uplus E_{i \text{ t } 2p} \text{ for } i \in \{1, 2\}, \\
f'_{ENA} &: E_{1NA} \rightarrow E_{2NA} \text{ with } E_{iNA} = E_{i \text{ in}} \uplus E_{i \text{ out}} \text{ for } i \in \{1, 2\}, \\
f'_{EEA} &: E_{1EA} \rightarrow E_{2EA} \text{ with } E_{iEA} = E_{i \text{ wpre}} \uplus E_{i \text{ wpost}} \text{ for } i \in \{1, 2\}.
\end{aligned}$$

Define the corresponding PTI net morphism $f : NI_1 \rightarrow NI_2$ with $NI_j = (P_j, T_j, \text{pre}_j, \text{post}_j, I_j, m_j)$ for $j \in \{1, 2\}$ by $f = (f_P : P_1 \rightarrow P_2, f_T : T_1 \rightarrow T_2, f_I : I_1 \rightarrow I_2)$ with

$$\begin{aligned}
f_P(p) &= f'_{VG}(p) \text{ for } p \in P_1 \subseteq V_{1G}, \\
f_T(t) &= f'_{VG}(t) \text{ for } t \in T_1 \subseteq V_{1G}, \\
f_I(x) &= f'_{VG}(x) \text{ for } x \in I_1 \subseteq V_{1G}.
\end{aligned}$$

- **Injectivity of $f : NI_1 \rightarrow NI_2$:**

We have to show that $f = (f_P : P_1 \rightarrow P_2, f_T : T_1 \rightarrow T_2, f_I : I_1 \rightarrow I_2)$ is an injective PTI net morphism, i.e., f_P , f_T and f_I are injective.

1. $f_P : P_1 \rightarrow P_2$ is injective:

Fix $p_1, p_2 \in P_1$. It holds the following:

$$\begin{aligned}
f_P(p_1) &= f_P(p_2) \\
&\stackrel{\text{Def. } f'_{V_G}}{\Rightarrow} f'_{V_G}(p_1) = f'_{V_G}(p_2) \\
&\stackrel{f'_{V_G} \text{ inj.}}{\Rightarrow} p_1 = p_2
\end{aligned}$$

2. $f_T : T_1 \rightarrow T_2$ **is injective:**

The proof for this case works analogously to the proof of Case 1.

3. $f_I : I_1 \rightarrow I_2$ **is injective:**

The proof for this case works analogously to the proof of Case 1.

• **Well-definedness of $f : NI_1 \rightarrow NI_2$:**

We have to show the following three steps:

1. $f_P(p) = f'_{V_G}(p) \in P_2$ for $p \in P_1$, $f_T(t) = f'_{V_G}(t) \in T_2$ for $t \in T_1$, $f_I(x) = f'_{V_G}(x) \in I_2$ for $x \in I_1$,
2. f is an injective PTI net morphism, i.e., squares (1) and (2) below commute with injective f_P, f_T, f_I , and
3. $\mathcal{F}_{PTI}(f) = f'$.

$$\begin{array}{ccc}
T_1 & \xrightarrow{\text{pre}_1} & P_1 \oplus \\
\downarrow f_T & \text{post}_1 & \downarrow f_P \oplus \\
& (1) & \\
T_2 & \xrightarrow{\text{pre}_2} & P_2 \oplus \\
& \text{post}_2 &
\end{array}
\quad
\begin{array}{ccc}
I_1 & \xrightarrow{m_1} & P_1 \\
\downarrow f_I & & \downarrow f_P \\
I_2 & \xrightarrow{m_2} & P_2 \\
& (2) &
\end{array}$$

The following holds:

1. To show: $f_P(p) = f'_{V_G}(p) \in P_2$ for $p \in P_1$.
 - a) $f'_{V_G}(p) \in V_{2G} = P_2 \uplus T_2 \uplus I_2$ by construction. The type-compatibility of f' given by assumption implies that $(\text{type}_{V_G}^2 \circ f'_{V_G})(p) = \text{type}_{V_G}^1(p) = \text{Place}$ using $p \in P_1$. This implies that $f_P(p) = f'_{V_G}(p) \in P_2$ using $\text{type}_{V_G}^2(f'_{V_G}(p)) = \text{Place}$ and $(\text{type}_{V_G}^2)^{-1}(\text{Place}) = P_2$.
 - b) $f_T(t) = f'_{V_G}(t) \in T_2$ for $t \in T_1$: similarly to the proof of Case 1a.
 - c) $f_I(x) = f'_{V_G}(x) \in I_2$ for $x \in I_1$: similarly to the proof of Case 1a.
2. To show: f is an injective PTI net morphism, i.e., squares (1) and (2) commute with injective f_P, f_T, f_I .

For this purpose, we have first to verify the following four conditions:

- (1) $\forall t \in T_1. p \in \bullet t \Rightarrow f_P(p) \in \bullet f_T(t)$ and $\forall t \in T_1. p \in t \bullet \Rightarrow f_P(p) \in f_T(t) \bullet$,
- (2) $\forall (p, t) \in P_1 \otimes T_1 = E_{1p2t}. ((p, t), n) \in E_{1w_{pre}} \Rightarrow ((f_P(p), f_T(t)), n) \in E_{2w_{pre}}$ and $\forall (t, p) \in T_1 \otimes P_1 = E_{1t2p}. ((t, p), n) \in E_{1w_{post}} \Rightarrow ((f_T(t), f_P(p)), n) \in E_{2w_{post}}$,
- (3) $\forall t \in T_1. |\bullet t| = n \Rightarrow |\bullet f_T(t)| = n$ and $|t \bullet| = n \Rightarrow |f_T(t) \bullet| = n$ with $\bullet t = \{p \in P_1 \mid \text{pre}_1(t)(p) > 0\}$ and $t \bullet = \{p \in P_1 \mid \text{post}_1(t)(p) > 0\}$,
- (4) $\forall x \in I_1. (x, p) \in E_{1to2p} \Rightarrow (f_I(x), f_P(p)) \in E_{2to2p}$.

We have the following:

- (1) $\forall t \in T_1. p \in \bullet t \Rightarrow f_P(p) \in \bullet f_T(t)$ and $\forall t \in T_1. p \in t \bullet \Rightarrow f_P(p) \in f_T(t) \bullet$:
Fix $p \in P_1$ and $t \in T_1$ such that $p \in \bullet t$. Then it holds:

$$\begin{aligned}
p \in \bullet t &\Leftrightarrow (p, t) \in E_{1p2t} \\
&\Rightarrow f'_{E_G}(p, t) \in E_{2p2t} \\
&\stackrel{(*)}{\Rightarrow} (f_P(p), f_T(t)) \in E_{2p2t} \\
&\Leftrightarrow f_P(p) \in \bullet f_T(t)
\end{aligned}$$

Where for the step (*) it holds that:

$f'_{E_G}(p, t) = (f_P(p), f_T(t))$, because f'_{E_G}, f'_{V_G} are compatible with s_{iG}, t_{iG} for $i \in \{1, 2\}$ (see diagram below) since f' is a graph morphism.

This implies that:

$$(s_{2G} \circ f'_{E_G})(p, t) = (f'_{V_G} \circ s_{1G})(p, t) = f'_{V_G}(p) = f_P(p) \text{ and} \\ (t_{2G} \circ f'_{E_G})(p, t) = (f'_{V_G} \circ t_{1G})(p, t) = f'_{V_G}(t) = f_T(t)$$

$$\begin{array}{ccc} E_{1G} & \xrightarrow{s_{1G}} & V_{1G} \\ f'_{E_G} \downarrow & \begin{array}{c} t_{1G} \\ = \\ \end{array} & \downarrow f'_{V_G} \\ E_{2G} & \xrightarrow[t_{2G}]{} & V_{2G} \end{array}$$

Similarly we have that $\forall t \in T_1. p \in \mathbf{t} \bullet \Leftrightarrow f_P(p) \in f_T(t) \bullet$.

- (2) $\forall (p, t) \in P_1 \otimes T_1 = E_{1p2t}. ((p, t), n) \in E_{1w_{pre}} \Rightarrow ((f_P(p), f_T(t)), n) \in E_{2w_{pre}}$ and
 $\forall (t, p) \in T_1 \otimes P_1 = E_{1t2p}. ((t, p), n) \in E_{1w_{post}} \Rightarrow ((f_T(t), f_P(p)), n) \in E_{2w_{post}}$:

Fix $(p, t) \in P_1 \otimes T_1 = E_{1p2t}$ such that $((p, t), n) \in E_{1w_{pre}}$. Since $f' : \mathcal{F}_{PTI}(NI_1) \rightarrow \mathcal{F}_{PTI}(NI_2)$ is an **AGraphs_{PNTG}**-morphism we have:

$$\begin{aligned} ((p, t), n) \in E_{1w_{pre}} &\Rightarrow f'_{E_{EA}}((p, t), n) \in E_{2w_{pre}} \\ &\Rightarrow (f'_{E_G}(p, t), n) \in E_{2w_{pre}} \\ &\stackrel{(*)}{\Rightarrow} ((f_P(p), f_T(t)), n) \in E_{2w_{pre}} \end{aligned}$$

where we have in the second step $f'_{E_{EA}}((p, t), n) = (f'_{E_G}(p, t), n)$ using the diagram below.

$$\begin{array}{ccc} & E_{1EA} & \xrightarrow{s_{1EA}} E_{1G} \\ t_{1EA} \swarrow & \downarrow f'_{E_{EA}} & \downarrow f'_{E_G} \\ \mathbb{N} & = & \\ t_{2EA} \swarrow & E_{2EA} & \xrightarrow{s_{2EA}} E_{2G} \end{array}$$

Similarly we have that $\forall (t, p) \in T_1 \otimes P_1 = E_{1t2p}. ((t, p), n) \in E_{1w_{post}} \Rightarrow ((f_T(t), f_P(p)), n) \in E_{2w_{post}}$.

- (3) $\forall t \in T_1. |\bullet t| = n \Rightarrow |\bullet f_T(t)| = n$ and $|t \bullet| = n \Rightarrow |f_T(t) \bullet| = n$ with
 $\bullet t = \{p \in P_1 \mid \text{pre}_1(t)(p) > 0\}$ and $\mathbf{t} \bullet = \{p \in P_1 \mid \text{post}_1(t)(p) > 0\}$:

The proof for this case works similarly to the proof of Case (2) using the following diagram.

$$\begin{array}{ccc} & E_{1NA} & \xrightarrow{s_{1NA}} V_{1G} \\ t_{1NA} \swarrow & \downarrow f'_{E_{NA}} & \downarrow f'_{V_G} \\ \mathbb{N} & = & \\ t_{2NA} \swarrow & E_{2NA} & \xrightarrow{s_{2NA}} V_{2G} \end{array}$$

- (4) $\forall x \in I_1. (x, p) \in E_{1t_02p} \Rightarrow (f_I(x), f_P(p)) \in E_{2t_02p}$:

The proof for this case works similarly to the proof of Case (1) using the diagram of Case (1).

Furthermore, we show in the following that f is an injective PTI net morphism if $f = (f_P, f_T, f_I)$ is injective satisfying the conditions (1) – (4).

We start with showing that $\forall x \in I_1. f_P \circ m_1(x) = m_2 \circ f_I(x)$.

Fix $x \in I_1$. Then it holds:

$$\begin{aligned} x \in I_1 &\Rightarrow (x, m_1(x)) \in E_{1 \text{ to } 2P} \\ &\stackrel{(4)}{\Rightarrow} (f_I(x), (f_P \circ m_1)(x)) \in E_{2 \text{ to } 2P} \\ &\Rightarrow (m_2 \circ f_I)(x) = (f_P \circ m_1)(x) \end{aligned}$$

As the next step we show that $\forall t \in T_1. f_P^\oplus \circ \text{pre}_1(t) = \text{pre}_2 \circ f_T(t)$.

Fix $t \in T_1$ and let $\text{pre}_1(t) = \sum_{i=1}^m \lambda_i p_i$, where p_i -s are pairwise disjoint and $\lambda_i > 0$, $\text{pre}_2(f_T(t)) = \sum_{j=1}^{m'} \lambda'_j p'_j$, where p'_j -s are pairwise disjoint and $\lambda'_j > 0$. Then it holds for $i \in \{1, \dots, m\}$:

$$p_i \in \bullet t \stackrel{(1)}{\Rightarrow} f_P(p_i) \in \bullet f_T(t) \Rightarrow \exists j \in \{1, \dots, m'\}. p_j = f_P(p_i)$$

Furthermore we have:

$$\begin{aligned} ((p_i, t), \lambda_i) \in E_{1 \text{ wpre}} &\stackrel{(2)}{\Rightarrow} ((f_P(p_i), f_T(t)), \lambda_i) \in E_{2 \text{ wpre}} \\ &\Rightarrow \exists j \in \{1, \dots, m'\}. p_j = f_P(p_i) \wedge \lambda_i = \lambda'_j \end{aligned}$$

Finally, for a fixed $t \in T_1$ holds:

$$\begin{aligned} |\bullet t| = m &\stackrel{(3)}{\Rightarrow} |\bullet f_T(t)| = m = m', \text{ where } f_P \text{ is injective and } p_i\text{-s and } p_j\text{-s are} \\ &\text{pairwise disjoint} \\ &\Rightarrow \exists \text{ a permutation } \pi \text{ of } \{1, \dots, m\} \text{ with } f_P(p_i) = p'_{\pi(i)} \\ &\Rightarrow \text{pre}_2(f_T(t)) = \sum_{j=1}^{m'} \lambda'_j p'_j = \sum_{i=1}^m \lambda_i f_P(p_i) \\ &= f_P^\oplus \left(\sum_{i=1}^m \lambda_i p_i \right) = f_P^\oplus(\text{pre}_1(t)) \end{aligned}$$

For similar reasons we have that $\text{post}_2(f_T(t)) = f_P^\oplus(\text{post}_1(t))$.

3. To show: $\mathcal{F}_{\text{PTI}}(f) = f'$.

Let $\mathcal{F}_{\text{PTI}}(f) = f'' = (f''_{V_G}, f''_{V_D}, f''_{E_G}, f''_{E_{NA}}, f''_{E_{EA}})$. This implies that $f''_{V_G} \stackrel{\text{Def. 64}}{=} f_P \uplus f_T \uplus f_I$ and $f''_{V_D} = \text{id}_N$. But by construction above we have that:

$$\begin{aligned} f'_{V_G} &= f_P \uplus f_T \uplus f_I \\ &\Rightarrow f'_{V_G} = f''_{V_G} \wedge f'_{V_D} = f''_{V_D} = \text{id}_N \\ &\stackrel{\text{Lem. 44}}{\Rightarrow} f' = f'' \\ &\Rightarrow \mathcal{F}_{\text{PTI}}(f) = f'' = f' \end{aligned}$$

- **Uniqueness of $f : \text{NI}_1 \rightarrow \text{NI}_2$:**

Let $\mathcal{F}_{\text{PTI}}(f) = f'$. Assume that we have another \mathcal{M}_1 -morphism $g : \text{NI}_1 \rightarrow \text{NI}_2$ with $\mathcal{F}_{\text{PTI}}(g) = f'$. We have to show that $f = g$.

Since it holds that $\mathcal{F}_{\text{PTI}}(f) = f' = \mathcal{F}_{\text{PTI}}(g)$ and using Lemma 44 we have the following three cases:

Case 1:

$$\begin{aligned} \mathcal{F}_{\text{PTI}}(g) &= \mathcal{F}_{\text{PTI}}(f) \\ &\Rightarrow \forall t \in V_{1G}. \mathcal{F}_{\text{PTI}}(g)_{V_G}(t) = \mathcal{F}_{\text{PTI}}(f)_{V_G}(t) \end{aligned}$$

$$\Rightarrow \forall t \in T_1. g_T(t) = f_T(t)$$

$$\Rightarrow g_T = f_T$$

Case 2: Similarly to Case 1 applying $\mathcal{F}_{PTI}(g)$ and $\mathcal{F}_{PTI}(f)$ to an arbitrary place $p \in P_1$.

Case 3: Similarly to Case 1 applying $\mathcal{F}_{PTI}(g)$ and $\mathcal{F}_{PTI}(f)$ to an arbitrary individual token $x \in I_1$.

□

Lemma 46: (Application of \mathcal{F}_{PTI} to a PTI Net Boundary Object over an Injective Morphism $f : L \rightarrow G$, see page 186)

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2)$, a boundary object $B = (P^B, T^B, \text{pre}^B, \text{post}^B, I^B, m^B)$ over an injective morphism $f : L \rightarrow G$ in $(\mathbf{PTINet}, \mathcal{M}_1)$ constructed according to Remark 9, and the restricted \mathcal{M} -functor $\mathcal{F}_{PTI} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\mathbf{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then the application of \mathcal{F}_{PTI} to the boundary object B results in the following typed attributed graph:

$$\mathcal{F}_{PTI}(B) = ((B_0, NAT), \text{type}^{\mathcal{F}_{PTI}(B)}) \text{ with}$$

$$B_0 = (V_G^{B_0}, V_D^{B_0} = \mathbb{N}, E_G^{B_0}, E_{NA}^{B_0}, E_{EA}^{B_0}, (s_j^{B_0}, t_j^{B_0})_{j \in \{G, NA, EA\}}) \text{ where}$$

$$V_G^{B_0} = P^B \uplus T^B \uplus I^B = DP_T \cup DP_I \text{ with}$$

$$DP_T = \{p \in P^L \mid \exists t \in T^G \setminus f_T(T^L). f_P(p) \in (\bullet t \cup t \bullet)\},$$

$$DP_I = \{p \in P^L \mid \exists x \in I^G \setminus f_I(I^L). f_P(p) = m^G(x)\},$$

$$E_G^{B_0} = E_{to2p}^{B_0} \uplus E_{t2p}^{B_0} \uplus E_{p2t}^{B_0} = \emptyset \text{ since}$$

$$E_{to2p}^{B_0} = \{(x, p) \in I^B \times P^B \mid m^B(x) = p\} = \emptyset \text{ using } I^B = \emptyset,$$

$$E_{t2p}^{B_0} = \emptyset \text{ using } T^B = \emptyset,$$

$$E_{p2t}^{B_0} = \emptyset \text{ using } T^B = \emptyset,$$

$$E_{NA}^{B_0} = E_{in}^{B_0} \uplus E_{out}^{B_0} = \emptyset \text{ using } T^B = \emptyset,$$

$$E_{EA}^{B_0} = E_{w_{pre}}^{B_0} \uplus E_{w_{post}}^{B_0} = \emptyset \text{ since}$$

$$E_{w_{pre}}^{B_0} = \emptyset \text{ using } E_{p2t}^{B_0} = \emptyset,$$

$$E_{w_{post}}^{B_0} = \emptyset \text{ using } E_{t2p}^{B_0} = \emptyset,$$

and $\mathcal{F}_{PTI}(b) : \mathcal{F}_{PTI}(B) \rightarrow \mathcal{F}_{PTI}(L)$ is an inclusion.

Proof.

Applying Definition 64 to the boundary object $B = (P^B, T^B, \text{pre}^B, \text{post}^B, I^B, m^B)$ constructed according to Remark 9, we directly get $\mathcal{F}_{PTI}(B)$ as given above. Furthermore, by Lemma 42, we have for the inclusion $b : B \rightarrow L$ that also $\mathcal{F}_{PTI}(b) : \mathcal{F}_{PTI}(B) \rightarrow \mathcal{F}_{PTI}(L)$ is an inclusion. □

Lemma 47: (Boundary Object in $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2)$ over an Injective Morphism $f' : L' \rightarrow G'$, see page 186)

Consider an injective typed attributed graph morphism $f' : L' \rightarrow G'$. Then the boundary object B' is given by $B' = ((B'_0, NAT), \text{type}^{B'})$ with the boundary points B'_0 that correspond to the dangling points and are defined as follows:

$$B'_0 = (V_G^{B'_0}, V_D^{B'_0} = \mathbb{N}, E_G^{B'_0}, E_{NA}^{B'_0}, E_{EA}^{B'_0}, (s_j^{B'_0}, t_j^{B'_0})_{j \in \{G, NA, EA\}}) \text{ given by}$$

$$B'_0 = \bigcap \{B'' \subseteq L' \mid V_D^{L'} = V_D^{B''} \wedge V_G^{B''} \subseteq V_G^{B''} \wedge E_G^{B''} \subseteq E_G^{B''} \wedge E_{NA}^{B''} \subseteq E_{NA}^{B''} \wedge E_{EA}^{B''} \subseteq E_{EA}^{B''}\}.$$

The components of $B''_0 = (V_G^{B''}, V_D^{B''} = \mathbb{N}, E_G^{B''}, E_{NA}^{B''}, E_{EA}^{B''}, (s_j^{B''}, t_j^{B''})_{j \in \{G, NA, EA\}})$ are defined as given below:

$$E_{NA}^{B''} = E_{EA}^{B''} = \emptyset,$$

$$V_G^{B''} = \{a \in V_G^{L'} = P^L \uplus T^L \uplus I^L \mid$$

$$[\exists a' \in E_{NA}^{G'} \setminus f'_{E_{NA}}(E_{NA}^{L'}) = (E_{in}^{G'} \uplus E_{out}^{G'}) \setminus f'_{E_{NA}}(E_{in}^{L'} \uplus E_{out}^{L'}) . f'_{V_G}(a) = s_{NA}^{G'}(a')]$$

$$\vee [\exists a' \in E_G^{G'} \setminus f'_{E_G}(E_G^{L'}) = (E_{to2p}^{G'} \uplus E_{p2t}^{G'} \uplus E_{t2p}^{G'}) \setminus f'_{E_G}(E_{to2p}^{L'} \uplus E_{p2t}^{L'} \uplus E_{t2p}^{L'}) .$$

$$f'_{V_G}(a) = s_G^{G'}(a') \vee f'_{V_G}(a) = t_G^{G'}(a')],$$

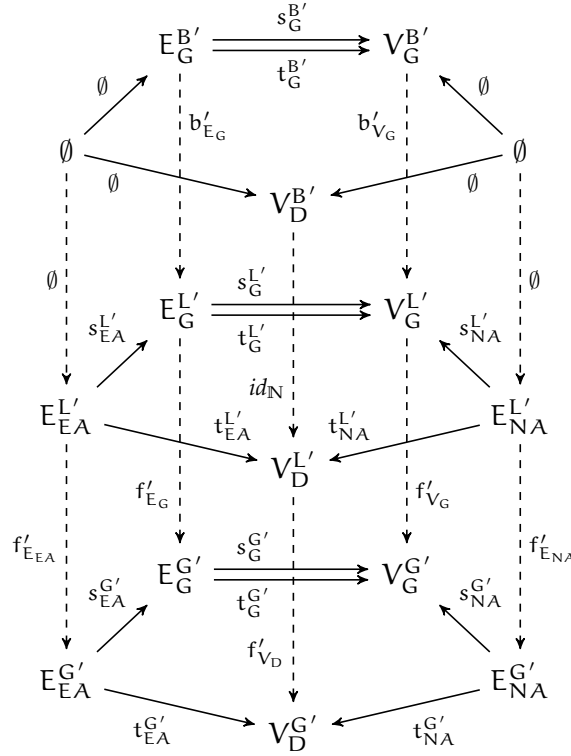
$$E_G^{B''} = \{a \in E_G^{L'} = E_{to2p}^{L'} \uplus E_{t2p}^{L'} \uplus E_{p2t}^{L'} \mid$$

$$\exists a' \in E_{EA}^{G'} \setminus f'_{E_{EA}}(E_{EA}^{L'}) = (E_{w_{pre}}^{G'} \uplus E_{w_{post}}^{G'}) \setminus f'_{E_{EA}}(E_{w_{pre}}^{L'} \uplus E_{w_{post}}^{L'}) .$$

$$f'_{E_G}(a) = s_{EA}^{G'}(a')],$$

$$s_G^{B''}, t_G^{B''} : E_G^{B''} \rightarrow V_G^{B''} \text{ are restrictions of } s_G^{L'}, t_G^{L'} : E_G^{L'} \rightarrow V_G^{L'},$$

and $b' : B' \rightarrow L'$ is an inclusion.



Proof.

The boundary object construction from Fact 6 is simplified to the form above, because the identification points for nodes, edges, node attributes and edge attributes are empty when assuming an injective typed attributed graph morphism $f' : L' \rightarrow G'$. \square

Lemma 48: (Boundary Object in $(\mathbf{AGraphs}_{\text{PTNG}}, \mathcal{M}_2)$ over a Morphism $f' = \mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ for an Injective PTI Net Morphism $f : L \rightarrow G$, see page 187)

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{PTNG}}, \mathcal{M}_2)$, an injective PTI net morphism $f : L \rightarrow G$, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PTNG}}|_{\mathcal{M}_2}$

from Definition 64. Then the boundary object B' of the initial pushout over a translated morphism $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ is constructed in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ as follows:

$B' = ((B'_0, \text{NAT}), \text{type}^{B'})$ is essentially given by the boundary points

$$\begin{aligned} B'_0 &= (V_G^{B'_0}, V_D^{B'_0} = \mathbb{N}, E_G^{B'_0}, E_{\text{NA}}^{B'_0}, E_{\text{EA}}^{B'_0}, (s_j^{B'_0}, t_j^{B'_0})_{j \in \{G, \text{NA}, \text{EA}\}}) \text{ with} \\ E_G^{B'_0} &= E_{\text{NA}}^{B'_0} = E_{\text{EA}}^{B'_0} = \emptyset, \\ V_G^{B'_0} &= \{a \in V_G^{L'} = P^L \uplus T^L \uplus I^L \mid \\ &[\exists a' \in E_{\text{NA}}^{G'} \setminus f'_{\text{EA}}(E_{\text{NA}}^{L'}) = (E_{\text{in}}^{G'} \uplus E_{\text{out}}^{G'}) \setminus f'_{\text{EA}}(E_{\text{in}}^{L'} \uplus E_{\text{out}}^{L'}) \cdot \\ &f'_{V_G}(a) = s_{\text{NA}}^{G'}(a')] \\ &\vee [\exists a' \in E_G^{G'} \setminus f'_{E_G}(E_G^{L'}) = (E_{\text{to2p}}^{G'} \uplus E_{\text{p2t}}^{G'} \uplus E_{\text{t2p}}^{G'}) \setminus f'_{E_G}(E_{\text{to2p}}^{L'} \uplus E_{\text{p2t}}^{L'} \uplus E_{\text{t2p}}^{L'}) \cdot \\ &f'_{V_G}(a) = s_G^{G'}(a') \vee f'_{V_G}(a) = t_G^{G'}(a')] \}, \end{aligned}$$

and $b'' : B' \rightarrow \mathcal{F}_{\text{PTI}}(L)$ is an inclusion.

$$\begin{array}{ccc} B' & \xrightarrow{b''} & \mathcal{F}_{\text{PTI}}(L) = L' \\ & & \downarrow \mathcal{F}_{\text{PTI}}(f) = f' \\ & & \mathcal{F}_{\text{PTI}}(G) = G' \end{array}$$

Proof.

First, we show that the set of edges $E_G^{B'_0}$ (or $E_G^{B''}$ as defined in Lemma 47) is empty. Obviously, the morphism $f' = \mathcal{F}_{\text{PTI}}(f)$ preserves the edge attributes of its source object. Furthermore, the edges in the source and the target objects of f' have unique edge attributes, because f' is an \mathcal{F}_{PTI} -image. Therefore, there can not be an edge attribute a' in the target object of f' , which is not reached by f' if the edge $s_{\text{EA}}^{G'}(a')$ is reached by f' . Thus, $E_G^{B'_0}$ is empty.

From this fact we conclude that B'_0 is not required to be constructed by intersection, because the graph B'' defined in Lemma 47 is a well-defined graph consisting only of nodes (the node attributes and edge attributes have already been empty in Lemma 47). \square

Lemma 49: (Initial Pushout in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ over a Morphism $f' = \mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ for an Injective PTI Net Morphism $f : L \rightarrow G$, see page 190)

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, an injective PTI net morphism $f : L \rightarrow G$, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then the diagram (1) given below with the boundary object B' constructed according to Lemma 48, the context object C' constructed according to Fact 7, inclusions $b' : B' \rightarrow L'$, $c' : C' \rightarrow G'$, and the morphism $g' : B' \rightarrow C'$ given by $g'_j(x) = (f'_j \circ b'_j)(x)$ for $j \in \{V_G, V_D, E_G, E_{\text{NA}}, E_{\text{EA}}\}$ is an initial pushout in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ over $f' = \mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$.

$$\begin{array}{ccc} B' & \xrightarrow{b'} & L' \\ g' \downarrow & (1) & \downarrow f' = \mathcal{F}_{\text{PTI}}(f) \\ C' & \xrightarrow{c'} & G' \end{array}$$

Proof.

According to Fact 7 and the proofs of Lemmas 47 and 48, the construction of initial pushouts in $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M}_2)$ as given in [88] (see Fact 8) coincides with the construction of initial pushouts in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ as given in this lemma for the case if the initial pushout is constructed over an injective \mathcal{F}_{PTI} -image morphism f' . \square

Lemma 50: (\mathcal{F}_{PTI} Preserves Initial Pushouts over Injective Morphisms [207], see page 190)

Consider \mathcal{M} -adhesive categories $(\mathbf{PTINet}, \mathcal{M}_1)$, $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64, and let (1) be an initial pushout over an injective morphism $f : L \rightarrow G$ in $(\mathbf{PTINet}, \mathcal{M}_1)$. Then (2) is an initial pushout over an injective morphism $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$.

$$\begin{array}{ccc} B & \xrightarrow{b} & L \\ \downarrow & (1) & \downarrow f \\ C & \longrightarrow & G \end{array} \quad \Rightarrow \quad \begin{array}{ccc} \mathcal{F}_{\text{PTI}}(B) & \xrightarrow{\mathcal{F}_{\text{PTI}}(b)} & \mathcal{F}_{\text{PTI}}(L) \\ \downarrow & (2) & \downarrow \mathcal{F}_{\text{PTI}}(f) \\ \mathcal{F}_{\text{PTI}}(C) & \longrightarrow & \mathcal{F}_{\text{PTI}}(G) \end{array}$$

Proof.

We assume that (1) is an initial pushout in $(\mathbf{PTINet}, \mathcal{M}_1)$ with boundary object $B = (P^B, T^B = \emptyset, \text{pre}^B = \emptyset, \text{post}^B = \emptyset, I^B = \emptyset, m^B = \emptyset)$ defined according to Remark 9, context object $C = (P^C, T^C, \text{pre}^C = \text{pre}^G, \text{post}^C = \text{post}^G, I^C, m^C = m^G)$ defined according to Fact 10 and an injective PTI net morphism $f : L \rightarrow G$.

$$\begin{array}{ccc} & & b' \\ & \swarrow & \searrow \\ B' & \xrightarrow{i} & \mathcal{F}_{\text{PTI}}(B) \xrightarrow{\mathcal{F}_{\text{PTI}}(b)} \mathcal{F}_{\text{PTI}}(L) = L' \\ & \downarrow (4) & \downarrow (2) \downarrow \mathcal{F}_{\text{PTI}}(f) = f' \\ C' & \xrightarrow{j} & \mathcal{F}_{\text{PTI}}(C) \rightarrow \mathcal{F}_{\text{PTI}}(G) = G' \\ & \nwarrow & \nearrow \\ & & c' \end{array} \quad \begin{array}{ccc} B' & \xrightarrow{b'} & \mathcal{F}_{\text{PTI}}(L) = L' \\ & \downarrow (3) & \downarrow \mathcal{F}_{\text{PTI}}(f) = f' \\ C' & \xrightarrow{c'} & \mathcal{F}_{\text{PTI}}(G) = G' \end{array}$$

Since \mathcal{F}_{PTI} preserves pushouts of injective morphisms according to Lemma 43, we have that (2) is a pushout of injective morphisms in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$.

Now we construct the initial pushout (3) over the morphism $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ according to Lemma 49 with inclusion $b' : B' \rightarrow \mathcal{F}_{\text{PTI}}(L)$ and boundary object $B' = ((B'_0, \text{NAT}), \text{type}^{B'})$ with boundary points $B'_0 = (V_G^{B'_0}, V_D^{B'_0} = \mathbb{N}, E_G^{B'_0} = \emptyset, E_{\text{NA}}^{B'_0} = \emptyset, E_{\text{EA}}^{B'_0} = \emptyset, (s_j^{B'_0}, t_j^{B'_0})_{j \in \{G, \text{NA}, \text{EA}\}})$ defined according to Lemma 48. Initiality of (3) implies unique morphisms $i : B' \rightarrow \mathcal{F}_{\text{PTI}}(B)$ and $j : C' \rightarrow \mathcal{F}_{\text{PTI}}(C)$ such that (4) is a pushout in $\mathbf{AGraphs}_{\text{PNTG}}$ and (5), (6) commute with $i \in \mathcal{M}_2$. Consider now the typed attributed graph $\mathcal{F}_{\text{PTI}}(B) = ((B_0, \text{NAT}), \text{type}^{\mathcal{F}_{\text{PTI}}(B)})$ with $B_0 = (V_G^{B_0}, V_D^{B_0} = \mathbb{N}, E_G^{B_0} = \emptyset, E_{\text{NA}}^{B_0} = \emptyset, E_{\text{EA}}^{B_0} = \emptyset, (s_j^{B_0}, t_j^{B_0})_{j \in \{G, \text{NA}, \text{EA}\}})$ given according to Lemma 46.

In the next step we have to show that $i : B' \rightarrow \mathcal{F}_{\text{PTI}}(B)$ is surjective, i.e., $\mathcal{F}_{\text{PTI}}(B) \subseteq B'$. Since $(E_G^{B_0} = \emptyset) \subseteq E_G^{B'_0}$, $(E_{\text{NA}}^{B_0} = \emptyset) \subseteq E_{\text{NA}}^{B'_0}$ and $(E_{\text{EA}}^{B_0} = \emptyset) \subseteq E_{\text{EA}}^{B'_0}$, it remains only to show that $(V_G^{B_0} = \text{DP}_T \cup \text{DP}_I) \subseteq V_G^{B'_0}$. For an arbitrary $p \in V_G^{B_0}$ we have the following two cases:

1. Let $p \in DP_T$:

By definition of DP_T we have that there is $t \in T^G \setminus f_T(T^L)$ with $f_P(p) \in (\bullet t \cup t \bullet)$. It holds without loss of generality that $f_P(p) \in \bullet t$ for $p \in P^L$.

It suffices to show for $p \in P^L$ that $\exists a' \in E_{p2t}^{G'} \setminus f'_{E_G}(E_{p2t}^{L'})$ with $f'_{V_G}(p) = s_G^{G'}(a')$.

Let $a' = (f_P(p), t) \in (E_G^{G'} = E_{t \circ 2p}^{G'} \uplus E_{p2t}^{G'} \uplus E_{t2p}^{G'})$, because $E_{p2t}^{G'} = \{(p', t) \in P^G \times T^G \mid p' \in \bullet t\}$ and $f_P(p) \in \bullet t$.

Assume now that $a' = (f_P(p), t) \in f'_{E_G}(E_{p2t}^{L'})$. This implies that $\exists (p'', t'') \in E_{p2t}^{L'}$ such that $p'' \in \bullet t''$ with $f'_{E_G}(p'', t'') = a' = (f_P(p), t)$. Then it holds: $f'_{E_G}(p'', t'') \stackrel{\text{Def. 64}}{=} (f_P(p''), f_T(t'')) = (f_P(p), t)$. Since f_P is injective, we have that $p'' = p$ and $f_T(t'') = t \in f_T(T^L)$, which contradicts to the assumption that $t \notin f_T(T^L)$. This implies that $a' \in E_G^{G'} \setminus f'_{E_G}(E_{p2t}^{L'})$ with $s_G^{G'}(a') = s_G^{G'}(f_P(p), t) \stackrel{\text{Def. 64}}{=} f_P(p) \stackrel{\text{Def. 64}}{=} f'_{V_G}(p)$. Thus, we get that $p \in V_G^{B_0}$, which was to be shown.

2. Let $p \in DP_I$:

By definition of DP_I we have that there is $x \in I^G \setminus f_I(I^L)$ with $f_P(p) = m^G(x)$ for $p \in P^L$. It suffices to show for $p \in P^L$ that $\exists a' \in E_{t \circ 2p}^{G'} \setminus f'_{E_G}(E_{t \circ 2p}^{L'})$ with $f'_{V_G}(p) = t_G^{G'}(a')$.

Let $a' = (x, f_P(p)) \in E_{t \circ 2p}^{G'}$ with $f'_{V_G}(p) \stackrel{\text{Def. 64}}{=} f_P(p) \stackrel{\text{Def. 64}}{=} t_G^{G'}(x, f_P(p)) = t_G^{G'}(a')$.

Similarly to above we show that $a' \notin f'_{E_G}(E_{t \circ 2p}^{L'})$ using $x \notin f_I(I^L)$, which altogether implies that $p \in V_G^{B_0}$.

This concludes the part of surjectivity of $i : B' \rightarrow \mathcal{F}_{PTI}(B)$. Now i is injective and surjective, so we get that i is an isomorphism. Since (4) is a pushout, also $j : C' \rightarrow \mathcal{F}_{PTI}(C)$ is an isomorphism and hence (2) is isomorphic to (3). So we get that also (2) is an initial pushout over an injective morphism $\mathcal{F}_{PTI}(f) : \mathcal{F}_{PTI}(L) \rightarrow \mathcal{F}_{PTI}(G)$. \square

Lemma 51: ($\mathcal{E} - \mathcal{M}$ -Factorization in PTINet, see [page 194](#))

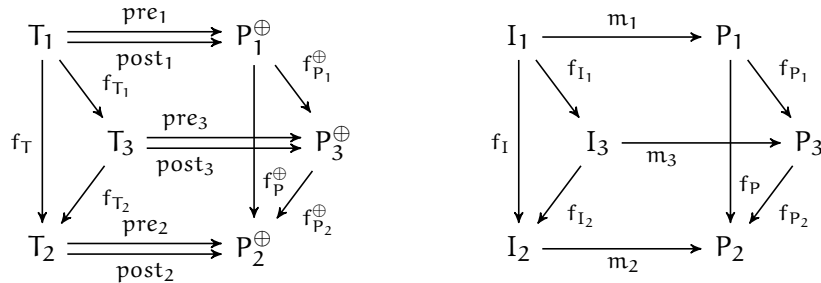
The \mathcal{M} -adhesive category $(PTINet, \mathcal{M}_1)$ has an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization according to Definition 10 where \mathcal{M}_1 is the class of all injective PTI net morphisms and \mathcal{E}_1 is the class of all surjective PTI net morphisms.

Proof.

Consider PTI nets $NI_1 = (P_1, T_1, pre_1 : T_1 \rightarrow P_1^\oplus, post_1 : T_1 \rightarrow P_1^\oplus, I_1, m_1 : I_1 \rightarrow P_1)$, $NI_2 = (P_2, T_2, pre_2 : T_2 \rightarrow P_2^\oplus, post_2 : T_2 \rightarrow P_2^\oplus, I_2, m_2 : I_2 \rightarrow P_2)$ and a PTI net morphism $f : NI_1 \rightarrow NI_2$. We construct an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization $(f_1 : NI_1 \rightarrow NI_3, f_2 : NI_3 \rightarrow NI_2)$ of f given to the right with

$$\begin{array}{ccc} NI_1 & \xrightarrow{f_1 = (f_{P_1}, f_{T_1}, f_{I_1}) \text{ surj.}} & NI_3 \\ f \downarrow & = & \downarrow \\ NI_2 & \xleftarrow{f_2 = (f_{P_2}, f_{T_2}, f_{I_2}) \text{ inj.}} & \end{array}$$

$NI_3 = (P_3, T_3, pre_3 : T_3 \rightarrow P_3^\oplus, post_3 : T_3 \rightarrow P_3^\oplus, I_3, m_3 : I_3 \rightarrow P_3)$ componentwise in **Sets** for places, transitions and individual tokens as follows



where $f_{P_1} : P_1 \rightarrow P_3$, $f_{T_1} : T_1 \rightarrow T_3$, $f_{I_1} : I_1 \rightarrow I_3$ are surjective and $f_{P_2} : P_3 \rightarrow P_2$, $f_{T_2} : T_3 \rightarrow T_2$, $f_{I_2} : I_3 \rightarrow I_2$ are injective. For the construction of the pre_3 function consider the following diagram.

$$\begin{array}{ccc}
 T_1 & \xrightarrow{\text{pre}_1} & P_1^\oplus \\
 f_{T_1} \text{ surj.} \downarrow & & \downarrow f_{P_1}^\oplus \\
 T_3 & \xrightarrow{\text{pre}_3} & P_3^\oplus \\
 f_{T_2} \text{ inj.} \downarrow & & \downarrow f_{P_2}^\oplus \\
 T_2 & \xrightarrow{\text{pre}_2} & P_2^\oplus
 \end{array}$$

Since f_{T_1} is surjective and hence an epimorphism in **Sets**, we also know that f_{T_1} is a coequalizer in **Sets**. Moreover, by Definition 2.6 from [251, p. 171] we have that f_{T_1} is a regular epimorphism implying by Lemma 2.10 from [251, p. 171] that f_{T_1} is a strong epimorphism. Finally, by definition of a strong epimorphism (see Definition 2.1 from [251, p. 170]) we get that there is a unique function $\text{pre}_3 : T_3 \rightarrow P_3^\oplus$ making the diagram below commute if $f_{P_2}^\oplus$ is injective, which can be easily shown as follows:

$f_{P_2}^\oplus$ is injective, i.e., $\forall M = x_1 \oplus \dots \oplus x_n, M' = x'_1 \oplus \dots \oplus x'_n \in P_3^\oplus. (f_{P_2}^\oplus(x_1 \oplus \dots \oplus x_n) = f_{P_2}^\oplus(x'_1 \oplus \dots \oplus x'_n)) \Rightarrow (x_1 \oplus \dots \oplus x_n = x'_1 \oplus \dots \oplus x'_n).$

Fix $M, M' \in P_3^\oplus$ and assume that $f_{P_2}^\oplus(x_1 \oplus \dots \oplus x_n) = f_{P_2}^\oplus(x'_1 \oplus \dots \oplus x'_n)$. Then we have:

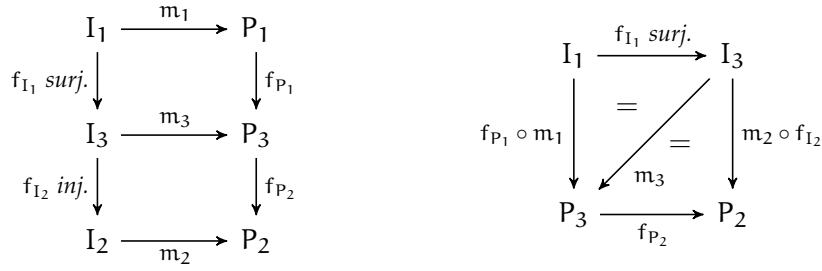
$$\begin{aligned}
 f_{P_2}^\oplus(x_1 \oplus \dots \oplus x_n) &= f_{P_2}^\oplus(x'_1 \oplus \dots \oplus x'_n) \\
 \text{Def. } f_{P_2}^\oplus &\Rightarrow f_{P_2}(x_1) \oplus \dots \oplus f_{P_2}(x_n) = f_{P_2}(x'_1) \oplus \dots \oplus f_{P_2}(x'_n) \\
 f_{P_2} \text{ inj.} + \text{commutativity} &\Rightarrow x_1 \oplus \dots \oplus x_n = x'_1 \oplus \dots \oplus x'_n
 \end{aligned}$$

$$\begin{array}{ccc}
 T_1 & \xrightarrow{f_{T_1} \text{ surj.}} & T_3 \\
 f_{P_1}^\oplus \circ \text{pre}_1 \downarrow & \begin{array}{c} = \\ \swarrow \text{pre}_3 \\ = \end{array} & \downarrow \text{pre}_2 \circ f_{T_2} \\
 P_3^\oplus & \xrightarrow{f_{P_2}^\oplus} & P_2^\oplus
 \end{array}$$

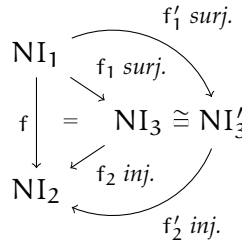
The unique function $\text{post}_3 : T_3 \rightarrow P_3^\oplus$ can be constructed similarly to the case of the pre_3 function considering the following diagrams:

$$\begin{array}{ccc}
 T_1 & \xrightarrow{\text{post}_1} & P_1^\oplus \\
 f_{T_1} \text{ surj.} \downarrow & & \downarrow f_{P_1}^\oplus \\
 T_3 & \xrightarrow{\text{post}_3} & P_3^\oplus \\
 f_{T_2} \text{ inj.} \downarrow & & \downarrow f_{P_2}^\oplus \\
 T_2 & \xrightarrow{\text{post}_2} & P_2^\oplus
 \end{array}
 \qquad
 \begin{array}{ccc}
 T_1 & \xrightarrow{f_{T_1} \text{ surj.}} & T_3 \\
 f_{P_1}^\oplus \circ \text{post}_1 \downarrow & \begin{array}{c} = \\ \swarrow \text{post}_3 \\ = \end{array} & \downarrow \text{post}_2 \circ f_{T_2} \\
 P_3^\oplus & \xrightarrow{f_{P_2}^\oplus} & P_2^\oplus
 \end{array}$$

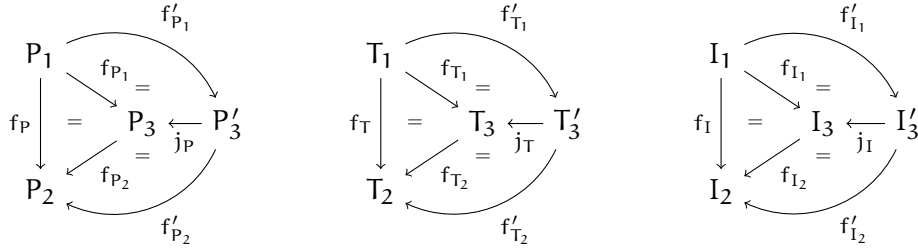
The unique marking function $m_3 : I_3 \rightarrow P_3$ can be constructed similarly to the case of the pre_3 function considering the following diagrams:



It remains to show that the PTI net NI_3 constructed in this way is unique up to isomorphism. For this reason assume that there are another PTI net $NI'_3 = (P'_3, T'_3, \text{pre}'_3 : T'_3 \rightarrow P'_3, \text{post}'_3 : T'_3 \rightarrow P'_3, I'_3, m'_3 : I'_3 \rightarrow P'_3)$ and morphisms $f'_1 : NI_1 \rightarrow NI'_3$ with $f'_1 = (f'_{P_1} : P_1 \rightarrow P'_3, f'_{T_1} : T_1 \rightarrow T'_3, f'_{I_1} : I_1 \rightarrow I'_3)$ in \mathcal{E}_1 , $f'_2 : NI'_3 \rightarrow NI_2$ with $f'_2 = (f'_{P_2} : P'_3 \rightarrow P_2, f'_{T_2} : T'_3 \rightarrow T_2, f'_{I_2} : I'_3 \rightarrow I_2)$ in \mathcal{M}_1 such that it holds that $f = f'_2 \circ f'_1$. We have to show that $NI'_3 \cong NI_3$.



By the componentwise construction of NI_3 , we know that there are isomorphisms $j_P : P'_3 \rightarrow P_3$, $j_T : T'_3 \rightarrow T_3$, $j_I : I'_3 \rightarrow I_3$ in **Sets** such that $f_{P_1} = j_P \circ f'_{P_1}$, $f_{T_1} = j_T \circ f'_{T_1}$, $f_{I_1} = j_I \circ f'_{I_1}$, $f'_{P_2} = f_{P_2} \circ j_P$, $f'_{T_2} = f_{T_2} \circ j_T$ and $f'_{I_2} = f_{I_2} \circ j_I$ as given in the diagrams below.



Thus $j : NI'_3 \rightarrow NI_3$ with $j = (j_P, j_T, j_I)$ is a PTI net isomorphism if it is compatible with the corresponding pre-, post- and marking functions, i.e.,

- $\text{pre}_3 \circ j_T = j_P^\oplus \circ \text{pre}'_3 :$

It holds the following:

$$\begin{aligned} \text{pre}_3 \circ j_T \circ f'_{T_1} & \stackrel{f_{T_1} = j_T \circ f'_{T_1}}{=} \text{pre}_3 \circ f_{T_1} \stackrel{(1) \text{ comm.}}{=} f_{P_1}^\oplus \circ \text{pre}_1 \\ & \stackrel{f_{P_1} = j_P \circ f'_{P_1}}{=} (j_P \circ f'_{P_1})^\oplus \circ \text{pre}_1 = j_P^\oplus \circ f_{P_1}'^\oplus \circ \text{pre}_1 \stackrel{(2) \text{ comm.}}{=} j_P^\oplus \circ \text{pre}'_3 \circ f'_{T_1} \end{aligned}$$

Thus, we have that $\text{pre}_3 \circ j_T \circ f'_{T_1} = j_P^\oplus \circ \text{pre}'_3 \circ f'_{T_1}$ and since f'_{T_1} is surjective by assumption, we get that $\text{pre}_3 \circ j_T = j_P^\oplus \circ \text{pre}'_3$.

- $\text{post}_3 \circ j_T = j_P^\oplus \circ \text{post}'_3 :$

The proof for this case works analogously to the proof of the case above.

$$\begin{array}{ccc}
\begin{array}{ccc} T'_3 & \xrightarrow{\text{pre}'_3} & P'_3{}^\oplus \\ j_T \downarrow & \text{post}'_3 = & \downarrow j_P{}^\oplus \\ T_3 & \xrightarrow{\text{pre}_3} & P_3{}^\oplus \\ & \text{post}_3 & \end{array} &
\begin{array}{ccc} T_1 & \xrightarrow{\text{pre}_1} & P_1{}^\oplus \\ f_{T_1} \downarrow & \text{post}_1 = & \downarrow f_{P_1}{}^\oplus \\ T_3 & \xrightarrow{\text{pre}_3} & P_3{}^\oplus \\ & \text{post}_3 & \end{array} &
\begin{array}{ccc} T_1 & \xrightarrow{\text{pre}_1} & P_1{}^\oplus \\ f_{T_1} \downarrow & \text{post}_1 = & \downarrow f_{P_1}{}^\oplus \\ T'_3 & \xrightarrow{\text{pre}'_3} & P'_3{}^\oplus \\ & \text{post}'_3 & \end{array}
\end{array}$$

- $\mathbf{m}_3 \circ \mathbf{j}_I = \mathbf{j}_P \circ \mathbf{m}'_3$:

It holds the following:

$$\begin{aligned}
\mathbf{m}_3 \circ \mathbf{j}_I \circ f'_{I_1} & \stackrel{f_{I_1} = j_I \circ f'_{I_1}}{=} \mathbf{m}_3 \circ f_{I_1} \stackrel{(3) \text{ comm.}}{=} f_{P_1} \circ \mathbf{m}_1 \\
f_{P_1} & \stackrel{f_{P_1} = j_P \circ f'_{P_1}}{=} j_P \circ f'_{P_1} \circ \mathbf{m}_1 \stackrel{(4) \text{ comm.}}{=} j_P \circ \mathbf{m}'_3 \circ f'_{I_1}
\end{aligned}$$

Thus, we have that $\mathbf{m}_3 \circ \mathbf{j}_I \circ f'_{I_1} = j_P \circ \mathbf{m}'_3 \circ f'_{I_1}$ and since f'_{I_1} is surjective by assumption, we get that $\mathbf{m}_3 \circ \mathbf{j}_I = j_P \circ \mathbf{m}'_3$.

$$\begin{array}{ccc}
\begin{array}{ccc} I'_3 & \xrightarrow{\mathbf{m}'_3} & P'_3 \\ j_I \downarrow & = & \downarrow j_P \\ I_3 & \xrightarrow{\mathbf{m}_3} & P_3 \end{array} &
\begin{array}{ccc} I_1 & \xrightarrow{\mathbf{m}_1} & P_1 \\ f_{I_1} \downarrow & (3) & \downarrow f_{P_1} \\ I_3 & \xrightarrow{\mathbf{m}_3} & P_3 \end{array} &
\begin{array}{ccc} I_1 & \xrightarrow{\mathbf{m}_1} & P_1 \\ f'_{I_1} \downarrow & (4) & \downarrow f'_{P_1} \\ I'_3 & \xrightarrow{\mathbf{m}'_3} & P'_3 \end{array}
\end{array}$$

This implies that $j : \mathbf{NI}'_3 \rightarrow \mathbf{NI}_3$ is a PTI net isomorphism and hence $\mathbf{NI}'_3 \cong \mathbf{NI}_3$. \square

Lemma 52: (\mathcal{F}_{PTI} Preserves Coproducts, see page 194)

Consider a PTI net A , a family of PTI nets $(A_j)_{j \in I}$, a family of PTI net morphisms $(i_j : A_j \rightarrow A)_{j \in I}$, a coproduct $(A, (i_j)_{j \in I})$ of $(A_j)_{j \in I}$ in \mathbf{PTINet} , and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then $(\mathcal{F}_{\text{PTI}}(A), (\mathcal{F}_{\text{PTI}}(i_j))_{j \in I})$ is a coproduct of $(\mathcal{F}_{\text{PTI}}(A_j))_{j \in I}$ in $\mathbf{AGraphs}_{\text{PNTG}}$.

$$\begin{array}{ccc}
\begin{array}{ccc} A_j & \xrightarrow{i_j} & A \\ & \searrow f_j & \downarrow f \\ & & B \end{array} & \xrightarrow{\mathcal{F}_{\text{PTI}}} &
\begin{array}{ccc} \mathcal{F}_{\text{PTI}}(A_j) & \xrightarrow{\mathcal{F}_{\text{PTI}}(i_j)} & \mathcal{F}_{\text{PTI}}(A) \\ & \searrow \mathcal{F}_{\text{PTI}}(f_j) & \downarrow \mathcal{F}_{\text{PTI}}(f) \\ & & \mathcal{F}_{\text{PTI}}(B) \end{array}
\end{array}$$

Proof.

To prove the preservation of coproducts by \mathcal{F}_{PTI} it is sufficient to show, that \mathcal{F}_{PTI} preserves pushouts and initial objects, because pushouts over initial objects are coproducts. In Lemma 43 we have already shown that \mathcal{F}_{PTI} preserves pushouts. It remains to show that \mathcal{F}_{PTI} preserves initial objects, i.e., for the initial PTI net \emptyset , $\mathcal{F}_{\text{PTI}}(\emptyset) = \emptyset^{\mathbb{N}}$ is initial in $\mathbf{AGraphs}_{\text{PNTG}}^{\mathbb{N}}$. This holds obviously according to the explanation in Section 8.1. Thus, \mathcal{F}_{PTI} preserves coproducts. \square

Lemma 53: (\mathcal{F}_{PTI} Preserves Surjective Morphisms, see page 194)

Consider two PTI nets $\mathbf{NI}_1 = (P_1, T_1, \text{pre}_1, \text{post}_1, I_1, \mathbf{m}_1)$, $\mathbf{NI}_2 = (P_2, T_2, \text{pre}_2, \text{post}_2, I_2, \mathbf{m}_2)$, a surjective PTI net morphism $f : \mathbf{NI}_1 \rightarrow \mathbf{NI}_2$ with $f = (f_P : P_1 \rightarrow P_2, f_T : T_1 \rightarrow T_2, f_I : I_1 \rightarrow I_2)$, and the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$. Then the corresponding typed attributed graph morphism $\mathcal{F}_{\text{PTI}}(f) : \mathcal{F}_{\text{PTI}}(\mathbf{NI}_1) \rightarrow \mathcal{F}_{\text{PTI}}(\mathbf{NI}_2)$ with $\mathcal{F}_{\text{PTI}}(f) = f' = (f'_{V_G} : V_{1G} \rightarrow V_{2G}, f'_{V_D} : \mathbb{N} \rightarrow \mathbb{N}, f'_{E_G} : E_{1G} \rightarrow E_{2G}, f'_{E_{NA}} : E_{1NA} \rightarrow E_{2NA}, f'_{E_{EA}} : E_{1EA} \rightarrow E_{2EA})$ is also surjective.

Proof.

Consider surjective morphisms f_P , f_T and f_I , i.e., $\forall p' \in P_2. \exists p \in P_1. p' = f_P(p), \forall t' \in T_2. \exists t \in T_1. t' = f_T(t)$ and $\forall x' \in I_2. \exists x \in I_1. x' = f_I(x)$.

1. To show: f'_{V_G} is surjective, i.e., $\forall y' \in V_{2G}. \exists y \in V_{1G}. (y' = f_P(y)) \vee (y' = f_T(y)) \vee (y' = f_I(y))$.
 $f'_{V_G} : V_{1G} \rightarrow V_{2G}$ with $f'_{V_G} = f_P \uplus f_T \uplus f_I$ for $V_{iG} = P_i \uplus T_i \uplus I_i$ where $i \in \{1, 2\}$ is surjective, because the components f_P , f_T and f_I are surjective.
2. To show: f'_{V_D} is surjective, i.e., $\forall n' \in \mathbb{N}. \exists n \in \mathbb{N}. n' = \text{id}_{\mathbb{N}}(n)$.
 f'_{V_D} is obviously surjective.
3. To show: f'_{E_G} is surjective.
 $f'_{E_G} : E_{1G} \rightarrow E_{2G}$ for $E_{iG} = E_{i \text{ to } 2P} \uplus E_{iP2t} \uplus E_{iT2p}$ with $i \in \{1, 2\}$,
where $E_{i \text{ to } 2P} = \{(x, p) \in I_i \times P_i \mid m_i(x) = p\}$,
 $E_{iP2t} = \{(p, t) \in P_i \times T_i \mid \text{pre}_i(t)(p) > 0\}$ and
 $E_{iT2p} = \{(t, p) \in T_i \times P_i \mid \text{post}_i(t)(p) > 0\}$.
a) Let $(x', p') \in E_{2 \text{ to } 2P}$ with $m_2(x') = p'$.
It remains to show: $\exists (x, p) \in E_{1 \text{ to } 2P}. f'_{E_G}(x, p) = (x', p')$.
Since f_I and f_P are surjective, we have that $\exists x \in I_1. f_I(x) = x', \exists p \in P_1. f_P(p) = p'$
and it holds: $f'_{E_G}(x, p) \stackrel{\text{Def. 64}}{=} (f_I(x), f_P(p)) = (x', p')$. Furthermore, we have:

$$\begin{aligned}
m_2(x') = p' &\Rightarrow m_2(f_I(x)) = f_P(p) \\
&\stackrel{(1) \text{ comm.}}{\Rightarrow} f_P(m_1(x)) = f_P(p) \\
&\stackrel{f_P \text{ inj.}}{\Rightarrow} m_1(x) = p \\
&\stackrel{\text{Def. } E_{1 \text{ to } 2P}}{\Rightarrow} (x, p) \in E_{1 \text{ to } 2P}
\end{aligned}$$

$$\begin{array}{ccc}
I_1 & \xrightarrow{m_1} & P_1 \\
f_I \downarrow & (1) & \downarrow f_P \\
I_2 & \xrightarrow{m_2} & P_2
\end{array}$$

- b) Let $(p', t') \in E_{2P2t}$ with $\text{pre}_2(t')(p') > 0$.
It remains to show: $\exists (p, t) \in E_{1P2t}. f'_{E_G}(p, t) = (p', t')$.
Since f_P and f_T are surjective, we have that $\exists p \in P_1. f_P(p) = p', \exists t \in T_1. f_T(t) = t'$
and it holds: $f'_{E_G}(p, t) \stackrel{\text{Def. 64}}{=} (f_P(p), f_T(t)) = (p', t')$. Furthermore, we have:

$$\begin{aligned}
\text{pre}_2(t')(p') > 0 &\Rightarrow \text{pre}_2(f_T(t))(f_P(p)) > 0 \\
&\stackrel{(2) \text{ comm.}}{\Rightarrow} f_P^\oplus(\text{pre}_1(t))(f_P(p)) > 0 \\
&\stackrel{f_P \text{ inj.}}{\Rightarrow} \text{pre}_1(t)(p) > 0 \\
&\stackrel{\text{Def. } E_{1P2t}}{\Rightarrow} (p, t) \in E_{1P2t}
\end{aligned}$$

$$\begin{array}{ccc}
T_1 & \xrightarrow{\text{pre}_1} & P_1^\oplus \\
f_T \downarrow & (2) & \downarrow f_P^\oplus \\
T_2 & \xrightarrow{\text{pre}_2} & P_2^\oplus
\end{array}$$

- c) Let $(t', p') \in E_{2t2P}$ with $\text{post}_2(t')(p') > 0$. The proof for this case is similar to the proof of the Case 3b above.

4. To show: $f'_{E_{NA}}$ is surjective.

$f'_{E_{NA}} : E_{1NA} \rightarrow E_{2NA}$ for $E_{iNA} = E_{i_{in}} \uplus E_{i_{out}}$ with $i \in \{1, 2\}$,
 where $E_{i_{in}} = \{(t, n, in) \mid (t, n) \in T_i \times \mathbb{N} \wedge |\bullet t| = n\}$
 and $E_{i_{out}} = \{(t, n, out) \mid (t, n) \in T_i \times \mathbb{N} \wedge |t \bullet| = n\}$.

a) Let $(t', n', in) \in E_{2in}$ with $|\bullet t'| = n'$.

It remains to show: $\exists (t, n, in) \in E_{1in}. f'_{E_{NA}}(t, n, in) = (t', n', in)$.

Because of surjectivity of f_T holds: $\exists t \in T_1. f_T(t) = t'$ and let $n = |\bullet t|$. Then we get that $(t, n, in) \in E_{1in}$, because $|\bullet t| = n$. Furthermore, we have:

$$\begin{aligned} n' &= |\bullet t'| = |\bullet f_T(t)| \\ &\stackrel{\text{Def. } \bullet f_T(t)}{=} |\{p' \in P_2 \mid \text{pre}_2(f_T(t))(p') > 0\}| \\ &= |\{f_P(p) \in P_2 \mid \text{pre}_2(f_T(t))(f_P(p)) > 0\}| \\ &\stackrel{f_P \text{ inj., surj.}}{=} |\{p \in P_1 \mid \text{pre}_2(f_T(t))(f_P(p)) > 0\}| \\ &\stackrel{(2) \text{ comm.}}{=} |\{p \in P_1 \mid f_P \oplus (\text{pre}_1(t))(f_P(p)) > 0\}| \\ &\stackrel{f_P \text{ inj.}}{=} |\{p \in P_1 \mid \text{pre}_1(t)(p) > 0\}| \\ &\stackrel{\text{Def. } \bullet t}{=} |\bullet t| = n \end{aligned}$$

This implies: $f'_{E_{NA}}(t, n, in) \stackrel{\text{Def. 64}}{=} (f_T(t), n, in) = (t', n', in)$.

b) Let $(t', n', out) \in E_{2out}$ with $|t' \bullet| = n'$. The proof for this case is similar to the proof of the Case 4a above.

5. To show: $f'_{E_{EA}}$ is surjective.

$f'_{E_{EA}} : E_{1EA} \rightarrow E_{2EA}$ for $E_{iEA} = E_{i_{w_{pre}}} \uplus E_{i_{w_{post}}}$ with $i \in \{1, 2\}$,
 where $E_{i_{w_{pre}}} = \{((p, t), n) \in E_{i_{p2t}} \times \mathbb{N} \mid \text{pre}_i(t)(p) = n\}$
 and $E_{i_{w_{post}}} = \{((t, p), n) \in E_{i_{t2p}} \times \mathbb{N} \mid \text{post}_i(t)(p) = n\}$.

a) Let $((p', t'), n') \in E_{2_{w_{pre}}}$ with $\text{pre}_2(t')(p') = n'$.

It remains to show: $\exists ((p, t), n) \in E_{1_{w_{pre}}}. f'_{E_{EA}}((p, t), n) = ((p', t'), n')$.

Because of surjectivity of f_P and f_T holds: $\exists p \in P_1. f_P(p) = p', \exists t \in T_1. f_T(t) = t'$ and let $n = \text{pre}_1(t)(p)$. Then we get that $((p, t), n) \in E_{1_{w_{pre}}}$, because $\text{pre}_1(t)(p) = n$. Furthermore, we have:

$$\begin{aligned} n' &= \text{pre}_2(t')(p') = \text{pre}_2(f_T(t))(f_P(p)) \\ &\stackrel{(2) \text{ comm.}}{=} f_P \oplus (\text{pre}_1(t))(f_P(p)) \\ &\stackrel{f_P \text{ inj.}}{=} \text{pre}_1(t)(p) = n \end{aligned}$$

This implies: $f'_{E_{EA}}((p, t), n) \stackrel{\text{Def. 64}}{=} ((f_P(p), f_T(t)), n) = ((p', t'), n')$.

b) Let $((t', p'), n') \in E_{2_{w_{post}}}$ with $\text{post}_2(t')(p') = n'$. The proof for this case is similar to the proof of the Case 5a above.

□

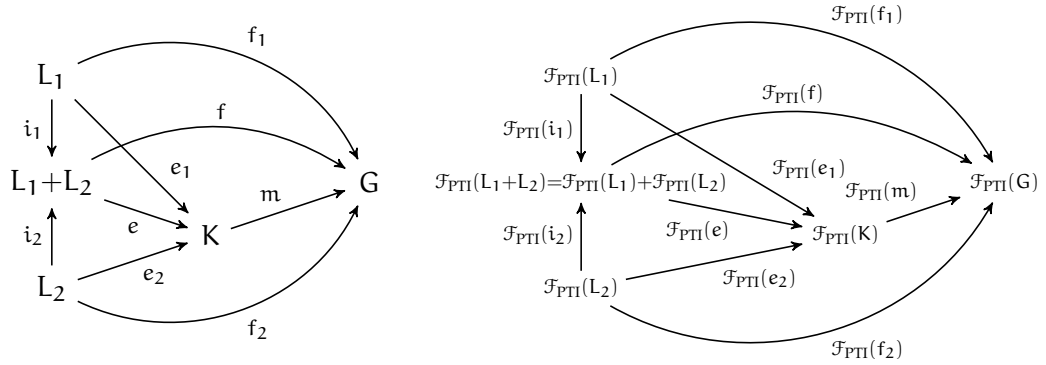
Lemma 54: (\mathcal{F}_{PTI} is Compatible with Pair Factorization, see page 195)

Consider a PTI net transformation system $(\mathbf{PTINet}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2, \mathcal{F}_{PTI}(P))$, and the restricted \mathcal{M} -functor $\mathcal{F}_{PTI} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\mathbf{PNTG}}|_{\mathcal{M}_2}$ from Definition 64. Then \mathcal{F}_{PTI} is compatible with pair factorization.

Proof.

According to Definition 51 and as already discussed before, we have to show that the categories $(\mathbf{PTINet}, \mathcal{M}_1)$ and $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2)$ have $\mathcal{E}' - \mathcal{M}$ pair factorizations and for each $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization $(f_1 = m \circ e_1, f_2 = m \circ e_2)$ in $(\mathbf{PTINet}, \mathcal{M}_1)$ also $(\mathcal{F}_{\mathbf{PTI}}(f_1) = \mathcal{F}_{\mathbf{PTI}}(m) \circ \mathcal{F}_{\mathbf{PTI}}(e_1), \mathcal{F}_{\mathbf{PTI}}(f_2) = \mathcal{F}_{\mathbf{PTI}}(m) \circ \mathcal{F}_{\mathbf{PTI}}(e_2))$ is an $\mathcal{E}'_2 - \mathcal{M}_2$ pair factorization in $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2)$ (see the diagram below). According to Lemma 1, an $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization of $(f_1 : L_1 \rightarrow G, f_2 : L_2 \rightarrow G)$ based on $\mathcal{E}_1 - \mathcal{M}_1$ -factorization and coproducts is given by $(f_1 = m \circ e_1, f_2 = m \circ e_2)$, where $f : L_1 + L_2 \rightarrow G$ is the induced morphism of $f_i : L_i \rightarrow G$ for $i \in \{1, 2\}$, $f = m \circ e$ is an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization according to Definition 10 and $e_1 = e \circ i_1, e_2 = e \circ i_2$ are defined via the coproduct morphisms $i_1 : L_1 \rightarrow L_1 + L_2$ and $i_2 : L_2 \rightarrow L_1 + L_2$ in $(\mathbf{PTINet}, \mathcal{M}_1)$. Similar we obtain a pair factorization in $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2)$. Lemma 1 is applicable to the categories $(\mathbf{PTINet}, \mathcal{M}_1)$ and $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2)$, because these categories have $\mathcal{E}_i - \mathcal{M}_i$ -factorizations for $i \in \{1, 2\}$ according to Lemma 51 and [88] as well as coproducts constructed componentwise as disjoint union, because the empty PTI net \emptyset resp. almost empty typed attributed graph $\mathcal{F}_{\mathbf{PTI}}(\emptyset)$ are initial in the categories \mathbf{PTINet} resp. $\mathbf{AGraphs}_{\mathbf{PNTG}}$ and we have pushouts in both categories.

In order to show that $\mathcal{F}_{\mathbf{PTI}}$ is compatible with pair factorization it remains to show that $\mathcal{F}_{\mathbf{PTI}}$ preserves pair factorization, i.e., for each pair factorization $(f_1 = m \circ e_1, f_2 = m \circ e_2)$ in $(\mathbf{PTINet}, \mathcal{M}_1)$ also $(\mathcal{F}_{\mathbf{PTI}}(f_1) = \mathcal{F}_{\mathbf{PTI}}(m) \circ \mathcal{F}_{\mathbf{PTI}}(e_1), \mathcal{F}_{\mathbf{PTI}}(f_2) = \mathcal{F}_{\mathbf{PTI}}(m) \circ \mathcal{F}_{\mathbf{PTI}}(e_2))$ is a pair factorization in $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2)$. This can be concluded, if $\mathcal{F}_{\mathbf{PTI}}$ preserves coproducts and surjective morphisms according to Lemma 16. Both properties are shown to hold in Lemmas 52 and 53 above. Thus, we get that $\mathcal{F}_{\mathbf{PTI}}$ is compatible with pair factorization.



□

Lemma 55: ($\mathcal{F}_{\mathbf{PTI}}$ Translates Jointly Surjective Morphisms, see page 196)

Consider $\mathcal{E}'_1 - \mathcal{M}_1$ pair factorization in $(\mathbf{PTINet}, \mathcal{M}_1)$ and $\mathcal{E}'_2 - \mathcal{M}_2$ pair factorization in $(\mathbf{AGraphs}_{\mathbf{PNTG}}, \mathcal{M}_2)$. Then the restricted \mathcal{M} -functor $\mathcal{F}_{\mathbf{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\mathbf{PNTG}}|_{\mathcal{M}_2}$ from Definition 64 translates jointly surjective PTI net morphisms (a', b') into the corresponding jointly surjective typed attributed graph morphisms (a'', b'') with $a'' = \mathcal{F}_{\mathbf{PTI}}(a')$ and $b'' = \mathcal{F}_{\mathbf{PTI}}(b')$.

$$\begin{array}{ccc}
 & P' & \\
 & \downarrow a' & \\
 C & \xrightarrow{b'} & C'
 \end{array}
 \qquad
 \begin{array}{ccc}
 & \mathcal{F}_{\mathbf{PTI}}(P') & \\
 & \downarrow a'' = \mathcal{F}_{\mathbf{PTI}}(a') & \\
 \mathcal{F}_{\mathbf{PTI}}(C) & \xrightarrow{b'' = \mathcal{F}_{\mathbf{PTI}}(b')} & C'' = \mathcal{F}_{\mathbf{PTI}}(C')
 \end{array}$$

Proof.

We have to show the following according to Definition 61:

$$\forall (a', b') \in \mathcal{E}'_1. \exists (a'', b'') \in \mathcal{E}'_2. a'' = \mathcal{F}_{\text{PTI}}(a') \wedge b'' = \mathcal{F}_{\text{PTI}}(b')$$

Let (a', b') be a pair of jointly surjective morphisms in \mathcal{E}'_1 . According to Lemma 66 from Appendix A, we know that $((a', b'), \text{id})$ is an \mathcal{E}'_1 - \mathcal{M}_1 pair factorization of (a', b') in $(\text{PTINet}, \mathcal{M}_1)$. Furthermore, we get by application of Lemma 54 that $((\mathcal{F}_{\text{PTI}}(a'), \mathcal{F}_{\text{PTI}}(b')), \mathcal{F}_{\text{PTI}}(\text{id}))$ is an \mathcal{E}'_2 - \mathcal{M}_2 pair factorization of $(\mathcal{F}_{\text{PTI}}(a'), \mathcal{F}_{\text{PTI}}(b'))$ in $(\text{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, which implies by the definition of \mathcal{E}' - \mathcal{M} pair factorization that $(\mathcal{F}_{\text{PTI}}(a'), \mathcal{F}_{\text{PTI}}(b')) \in \mathcal{E}'_2$. \square

Lemma 56: (\mathcal{F}_{PTI} Creates Jointly Surjective Morphisms, see page 196)

Consider \mathcal{E}'_1 - \mathcal{M}_1 pair factorization in $(\text{PTINet}, \mathcal{M}_1)$, \mathcal{E}'_2 - \mathcal{M}_2 pair factorization in $(\text{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$, and PTI net morphisms $a : P \rightarrow C$, $b : P \rightarrow P'$. Then the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \text{PTINet}|_{\mathcal{M}_1} \rightarrow \text{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ from Definition 64 creates jointly surjective PTI net morphisms (a', b') from the corresponding typed attributed graph morphisms (a'', b'') in $\bar{\mathcal{E}}'_2 = \mathcal{F}_{\text{PTI}}(\mathcal{E}'_1)$ if for all spans $(C \xleftarrow{a} P \xrightarrow{b} P')$ holds that the diagram (1) below commutes, $\mathcal{F}_{\text{PTI}}(a') = a''$, $\mathcal{F}_{\text{PTI}}(b') = b''$, and the injectivity of b'' implies the injectivity of b' .

$$\begin{array}{ccc} P & \xrightarrow{b} & P' \\ a \downarrow & (1) & \downarrow a' \\ C & \xrightarrow{b'} & C' \end{array} \quad \begin{array}{ccc} \mathcal{F}_{\text{PTI}}(P) & \xrightarrow{\mathcal{F}_{\text{PTI}}(b)} & \mathcal{F}_{\text{PTI}}(P') \\ \mathcal{F}_{\text{PTI}}(a) \downarrow & (2) & \downarrow a'' = \mathcal{F}_{\text{PTI}}(a') \\ \mathcal{F}_{\text{PTI}}(C) & \xrightarrow{b'' = \mathcal{F}_{\text{PTI}}(b')} & C'' = \mathcal{F}_{\text{PTI}}(C') \end{array}$$

Proof.

We have to show the following according to Definition 61:

$$\begin{aligned} & \forall (a'', b'') \in \bar{\mathcal{E}}'_2. (2) \text{ commutes} \wedge b'' \in \mathcal{M}_2 \Rightarrow \\ & \exists (a', b') \in \mathcal{E}'_1. a'' = \mathcal{F}_{\text{PTI}}(a') \wedge b'' = \mathcal{F}_{\text{PTI}}(b') \wedge (1) \text{ commutes} \wedge b' \in \mathcal{M}_1. \end{aligned}$$

Since $\bar{\mathcal{E}}'_2 = \mathcal{F}_{\text{PTI}}(\mathcal{E}'_1)$, we have that there is $(a', b') \in \mathcal{E}'_1$ with $a'' = \mathcal{F}_{\text{PTI}}(a')$ and $b'' = \mathcal{F}_{\text{PTI}}(b')$. Furthermore, by commutativity of (2) holds:

$$\begin{aligned} & \mathcal{F}_{\text{PTI}}(a') \circ \mathcal{F}_{\text{PTI}}(b) = \mathcal{F}_{\text{PTI}}(b') \circ \mathcal{F}_{\text{PTI}}(a) \\ & \xRightarrow{\text{funct. prop.}} \mathcal{F}_{\text{PTI}}(a' \circ b) = \mathcal{F}_{\text{PTI}}(b' \circ a) \\ & \xRightarrow{\text{Lem. 10} + \text{Rem. 10}} a' \circ b = b' \circ a \end{aligned}$$

Finally, we have that $b' \in \mathcal{M}_1$, because $b'' = \mathcal{F}_{\text{PTI}}(b')$ in \mathcal{M}_2 and \mathcal{F}_{PTI} creates injective morphisms by Lemma 45. \square

Lemma 57: (\mathcal{F}_{PTI} Preserves Pullbacks of Injective Morphisms, see page 197)

Consider a PTI net transformation system $(\text{PTINet}, \mathcal{M}_1, P)$, a typed attributed graph transformation system $(\text{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$, the restricted \mathcal{M} -functor $\mathcal{F}_{\text{PTI}} : \text{PTINet}|_{\mathcal{M}_1} \rightarrow \text{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$, PTI nets NI_i for $i \in \{0, 1, 2, 3\}$ with PTI net morphisms $b = (b_P, b_T, b_I)$, $c = (c_P, c_T, c_I)$, $g = (g_P, g_T, g_I)$, $h = (h_P, h_T, h_I)$, and typed attributed graphs $\mathcal{F}_{\text{PTI}}(\text{NI}_i)$ for $i \in \{0, 1, 2, 3\}$ with typed attributed graph morphisms $\mathcal{F}_{\text{PTI}}(b) =$

$b' = (b'_{V_G}, b'_{V_D}, b'_{E_G}, b'_{E_{NA}}, b'_{E_{EA}})$, $\mathcal{F}_{PTI}(c) = c' = (c'_{V_G}, c'_{V_D}, c'_{E_G}, c'_{E_{NA}}, c'_{E_{EA}})$, $\mathcal{F}_{PTI}(g) = g' = (g'_{V_G}, g'_{V_D}, g'_{E_G}, g'_{E_{NA}}, g'_{E_{EA}})$, $\mathcal{F}_{PTI}(h) = h' = (h'_{V_G}, h'_{V_D}, h'_{E_G}, h'_{E_{NA}}, h'_{E_{EA}})$. If (1) is a pullback in **PTINet** with $g, h \in \mathcal{M}_1$ then we have that (2) is a pullback in **AGraphs_{PNTG}** with $\mathcal{F}_{PTI}(g), \mathcal{F}_{PTI}(h) \in \mathcal{M}_2$.

$$\begin{array}{ccc}
 NI_0 & \xrightarrow{b} & NI_1 \\
 c \downarrow & (1) & \downarrow g \\
 NI_2 & \xrightarrow{h} & NI_3
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathcal{F}_{PTI}(NI_0) & \xrightarrow{\mathcal{F}_{PTI}(b)=b'} & \mathcal{F}_{PTI}(NI_1) \\
 \mathcal{F}_{PTI}(c)=c' \downarrow & (2) & \downarrow \mathcal{F}_{PTI}(g)=g' \\
 \mathcal{F}_{PTI}(NI_2) & \xrightarrow{\mathcal{F}_{PTI}(h)=h'} & \mathcal{F}_{PTI}(NI_3)
 \end{array}$$

Proof.

For the given morphisms $g, h \in \mathcal{M}_1$ we have that $\mathcal{F}_{PTI}(g) = g', \mathcal{F}_{PTI}(h) = h' \in \mathcal{M}_2$ according to Lemma 42.

Let (1) be a pullback in **PTINet** with $g, h \in \mathcal{M}_1$, i.e., P-, T- and I-components of (1) are pullbacks in **Sets**, because pullbacks in **PTINet** are constructed componentwise as shown in Fact 4.1.5 in [223, p. 73].

$$\begin{array}{ccccc}
 P_0 & \xrightarrow{b_P} & P_1 & & T_0 & \xrightarrow{b_T} & T_1 & & I_0 & \xrightarrow{b_I} & I_1 \\
 c_P \downarrow & (PB) & \downarrow g_P \text{ inj.} & & c_T \downarrow & (PB) & \downarrow g_T \text{ inj.} & & c_I \downarrow & (PB) & \downarrow g_I \text{ inj.} \\
 P_2 & \xrightarrow{h_P \text{ inj.}} & P_3 & & T_2 & \xrightarrow{h_T \text{ inj.}} & T_3 & & I_2 & \xrightarrow{h_I \text{ inj.}} & I_3
 \end{array}$$

To show: V_G^- , V_D^- , E_G^- , E_{NA}^- and E_{EA}^- -components of (2) are pullbacks of injective morphisms in **Sets**, because according to [88] pullbacks in **AGraphs_{ATG}** and hence also in **AGraphs_{PNTG}** are constructed componentwise as well.

1. V_G -component of (2) is a pullback of injective morphisms in **Sets** (see diagram (3)) with $f'_{V_G} = f_P \uplus f_T \uplus f_I$ for $f \in \{b, c, g, h\}$, because the components of (3) are pullbacks and pushouts of injective morphisms in **Sets** since $g = (g_P, g_T, g_I), h = (h_P, h_T, h_I) \in \mathcal{M}_1$ by assumption and pushouts are compatible with coproducts (and coproduct in **Sets** is \uplus).

$$\begin{array}{ccc}
 V_{0G} & \xrightarrow{b_P \uplus b_T \uplus b_I} & V_{1G} \\
 c_P \uplus c_T \uplus c_I \downarrow & (3) & \downarrow g_P \uplus g_T \uplus g_I \\
 V_{2G} & \xrightarrow{h_P \uplus h_T \uplus h_I} & V_{3G}
 \end{array}$$

2. V_D -component of (2) is obviously a pullback of injective morphisms in **Sets** (see diagram (4)) with $f'_{V_D} = \text{id}_{\mathbb{N}}$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
 V_{0D}=\mathbb{N} & \xrightarrow{\text{id}_{\mathbb{N}}} & V_{1D}=\mathbb{N} \\
 \text{id}_{\mathbb{N}} \downarrow & (4) & \downarrow \text{id}_{\mathbb{N}} \\
 V_{2D}=\mathbb{N} & \xrightarrow{\text{id}_{\mathbb{N}}} & V_{3D}=\mathbb{N}
 \end{array}$$

3. For the E_G -component we have to show that (5) is a pullback of injective morphisms in **Sets** which follows if (5a), (5b) and (5c) are pullbacks of injective morphisms.

$$\begin{array}{ccc}
E_{0G} & \xrightarrow{b'_{EG}} & E_{1G} \\
c'_{EG} \downarrow & (5) & \downarrow g'_{EG} \\
E_{2G} & \xrightarrow{h'_{EG}} & E_{3G} \\
E_{0p2t} & \xrightarrow{b_P \times b_T} & E_{1p2t} \\
c_P \times c_T \downarrow & (5b) & \downarrow g_P \times g_T \\
E_{2p2t} & \xrightarrow{h_P \times h_T} & E_{3p2t}
\end{array}
\quad
\begin{array}{ccc}
E_{0to2p} & \xrightarrow{b_I \times b_P} & E_{1to2p} \\
c_I \times c_P \downarrow & (5a) & \downarrow g_I \times g_P \\
E_{2to2p} & \xrightarrow{h_I \times h_P} & E_{3to2p} \\
E_{0t2p} & \xrightarrow{b_T \times b_P} & E_{1t2p} \\
c_T \times c_P \downarrow & (5c) & \downarrow g_T \times g_P \\
E_{2t2p} & \xrightarrow{h_T \times h_P} & E_{3t2p}
\end{array}$$

Diagrams (5a), (5b) and (5c) commute, because for each product component we have a pullback in **Sets** by assumption. So it remains to show that (5a), (5b) and (5c) are pullbacks.

a) For diagram (5a) we have to show that the diagram (5a') is a pullback in **Sets** with

$$I_i \otimes P_i = \{(x, p) \in I_i \times P_i \mid m_i(x) = p\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_I \otimes f_P$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
I_0 \otimes P_0 & \xrightarrow{b_I \otimes b_P} & I_1 \otimes P_1 \\
c_I \otimes c_P \downarrow & (5a') & \downarrow g_I \otimes g_P \\
I_2 \otimes P_2 & \xrightarrow{h_I \otimes h_P} & I_3 \otimes P_3
\end{array}$$

Since b, c, g, h are PTI net morphisms, we have that all $f_I \otimes f_P$ morphisms for $f \in \{b, c, g, h\}$ are well-defined. Furthermore, we have that the components of (5a') are pullbacks in **Sets** by assumption. Hence, also (5a') (as well as (5a)) is a pullback, because pullbacks are compatible with products.

b) For diagram (5b) we have to show that the diagram (5b') is a pullback in **Sets** with

$$P_i \otimes T_i = \{(p, t) \in P_i \times T_i \mid \text{pre}_i(t)(p) > 0\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_P \otimes f_T$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
P_0 \otimes T_0 & \xrightarrow{b_P \otimes b_T} & P_1 \otimes T_1 \\
c_P \otimes c_T \downarrow & (5b') & \downarrow g_P \otimes g_T \\
P_2 \otimes T_2 & \xrightarrow{h_P \otimes h_T} & P_3 \otimes T_3
\end{array}$$

The proof for this case is similar to the Case 3a. Hence, (5b') and (5b) are pullbacks.

c) For diagram (5c) we have to show that the diagram (5c') is a pullback in **Sets** with

$$T_i \otimes P_i = \{(t, p) \in T_i \times P_i \mid \text{post}_i(t)(p) > 0\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_T \otimes f_P$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
T_0 \otimes P_0 & \xrightarrow{b_T \otimes b_P} & T_1 \otimes P_1 \\
c_T \otimes c_P \downarrow & (5c') & \downarrow g_T \otimes g_P \\
T_2 \otimes P_2 & \xrightarrow{h_T \otimes h_P} & T_3 \otimes P_3
\end{array}$$

The proof for this case is similar to the Case 3a. Hence, (5c') and (5c) are pullbacks.

4. For the E_{NA} -component we have to show that (6) is a pullback of injective morphisms in **Sets**, which follows if (6a) and similarly (6b) are pullbacks of injective morphisms for $X = \{in\}$ and $Y = \{out\}$.

$$\begin{array}{ccccc}
 E_{0NA} & \xrightarrow{b'_{E_{NA}}} & E_{1NA} & & \\
 c'_{E_{NA}} \downarrow & & \downarrow g'_{E_{NA}} & & \\
 E_{2NA} & \xrightarrow{h'_{E_{NA}}} & E_{3NA} & & \\
 E_{0in} \xrightarrow{b_T \times id_N \times id_X} E_{1in} & & E_{0out} \xrightarrow{b_T \times id_N \times id_Y} E_{1out} & & \\
 c_T \times id_N \times id_X \downarrow & (6a) & \downarrow g_T \times id_N \times id_X & & \downarrow g_T \times id_N \times id_Y & (6b) \\
 E_{2in} \xrightarrow{h_T \times id_N \times id_X} E_{3in} & & E_{2out} \xrightarrow{h_T \times id_N \times id_Y} E_{3out} & &
 \end{array}$$

Diagrams (6a) and (6b) commute, because for each product component we have a pullback in **Sets** by assumption. So it remains to show that (6a) and (6b) are pullbacks.

- a) For diagram (6a) we have to show that the diagram (6a') is a pullback in **Sets** with

$$T_i \otimes \mathbb{N} \otimes X = \{(t, n, in) \mid (t, n) \in T_i \times \mathbb{N} \wedge |\bullet t| = n\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_T \otimes id_N \otimes id_X$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
 T_0 \otimes \mathbb{N} \otimes X & \xrightarrow{b_T \otimes id_N \otimes id_X} & T_1 \otimes \mathbb{N} \otimes X \\
 c_T \otimes id_N \otimes id_X \downarrow & (6a') & \downarrow g_T \otimes id_N \otimes id_X \\
 T_2 \otimes \mathbb{N} \otimes X & \xrightarrow{h_T \otimes id_N \otimes id_X} & T_3 \otimes \mathbb{N} \otimes X
 \end{array}$$

Also here we know that $f_T \otimes id_N \otimes id_X$ morphisms for $f \in \{b, c, g, h\}$ are well-defined since b, c, g, h are PTI net morphisms. Furthermore, we have that the components of (6a') are pullbacks in **Sets** by assumption. Hence, also (6a') (as well as (6a)) is a pullback, because pullbacks are compatible with products.

- b) For diagram (6b) we have to show that the diagram (6b') is a pullback in **Sets** with

$$T_i \otimes \mathbb{N} \otimes Y = \{(t, n, out) \mid (t, n) \in T_i \times \mathbb{N} \wedge |t \bullet| = n\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_T \otimes id_N \otimes id_Y$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
 T_0 \otimes \mathbb{N} \otimes Y & \xrightarrow{b_T \otimes id_N \otimes id_Y} & T_1 \otimes \mathbb{N} \otimes Y \\
 c_T \otimes id_N \otimes id_Y \downarrow & (6b') & \downarrow g_T \otimes id_N \otimes id_Y \\
 T_2 \otimes \mathbb{N} \otimes Y & \xrightarrow{h_T \otimes id_N \otimes id_Y} & T_3 \otimes \mathbb{N} \otimes Y
 \end{array}$$

The proof for this case is similar to the Case 4a. Hence, (6b') and (6b) are pullbacks.

5. For the E_{EA} -component we have to show that (7) is a pullback of injective morphisms in **Sets**, which follows if (7a) and similarly (7b) are pullbacks of injective morphisms.

$$\begin{array}{ccccc}
E_{0EA} & \xrightarrow{b'_{EA}} & E_{1EA} & & \\
c'_{EA} \downarrow & & \downarrow g'_{EA} & (7) & \\
E_{2EA} & \xrightarrow{h'_{EA}} & E_{3EA} & & \\
\downarrow c_P \times c_T \times \text{id}_N & & \downarrow g_P \times g_T \times \text{id}_N & & \\
E_{0w_{pre}} & \xrightarrow{b_P \times b_T \times \text{id}_N} & E_{1w_{pre}} & & E_{0w_{post}} \xrightarrow{b_T \times b_P \times \text{id}_N} E_{1w_{post}} \\
\downarrow c_P \times c_T \times \text{id}_N & & \downarrow g_P \times g_T \times \text{id}_N & (7a) & \downarrow g_T \times g_P \times \text{id}_N \\
E_{2w_{pre}} & \xrightarrow{h_P \times h_T \times \text{id}_N} & E_{3w_{pre}} & & E_{2w_{post}} \xrightarrow{h_T \times h_P \times \text{id}_N} E_{3w_{post}}
\end{array}$$

Diagrams (7a) and (7b) commute, because also here we have a pullback in **Sets** for each product component by assumption. So it remains to show that (7a) and (7b) are pullbacks.

a) For diagram (7a) we have to show that the diagram (7a') is a pullback in **Sets** with

$$P_i \otimes T_i \otimes \mathbb{N} = \{((p, t), n) \in E_{iP2t} \times \mathbb{N} \mid \text{pre}_i(t)(p) = n\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_P \otimes f_T \otimes \text{id}_N$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
P_0 \otimes T_0 \otimes \mathbb{N} & \xrightarrow{b_P \otimes b_T \otimes \text{id}_N} & P_1 \otimes T_1 \otimes \mathbb{N} \\
c_P \otimes c_T \otimes \text{id}_N \downarrow & (7a') & \downarrow g_P \otimes g_T \otimes \text{id}_N \\
P_2 \otimes T_2 \otimes \mathbb{N} & \xrightarrow{h_P \otimes h_T \otimes \text{id}_N} & P_3 \otimes T_3 \otimes \mathbb{N}
\end{array}$$

Similarly to the cases above we have that $f_P \otimes f_T \otimes \text{id}_N$ morphisms for $f \in \{b, c, g, h\}$ are well-defined since b, c, g, h are PTI net morphisms. Furthermore, it holds that the components of (7a') are pullbacks in **Sets** by assumption. Hence, also (7a') (as well as (7a)) is a pullback, because pullbacks are compatible with products.

b) For diagram (7b) we have to show that the diagram (7b') is a pullback in **Sets** with

$$T_i \otimes P_i \otimes \mathbb{N} = \{((t, p), n) \in E_{iTP2p} \times \mathbb{N} \mid \text{post}_i(t)(p) = n\} \text{ for } i \in \{0, 1, 2, 3\}$$

and $f_T \otimes f_P \otimes \text{id}_N$ for $f \in \{b, c, g, h\}$.

$$\begin{array}{ccc}
T_0 \otimes P_0 \otimes \mathbb{N} & \xrightarrow{b_T \otimes b_P \otimes \text{id}_N} & T_1 \otimes P_1 \otimes \mathbb{N} \\
c_T \otimes c_P \otimes \text{id}_N \downarrow & (7b') & \downarrow g_T \otimes g_P \otimes \text{id}_N \\
T_2 \otimes P_2 \otimes \mathbb{N} & \xrightarrow{h_T \otimes h_P \otimes \text{id}_N} & T_3 \otimes P_3 \otimes \mathbb{N}
\end{array}$$

The proof for this case is similar to the Case 5a. Hence, (7b') and (7b) are pullbacks. \square

Lemma 64: ($\mathcal{F}_{\text{PTIC}}$ is a Category Equivalence, see page 223)

The categories $\mathbf{PTINet}_{|\mathcal{M}_1}$ and $\mathbf{SubAGraphs}_{\text{PNTG}}$ are equivalent, i.e., there exists a category equivalence $\mathcal{F}_{\text{PTIC}} : \mathbf{PTINet}_{|\mathcal{M}_1} \xrightarrow{\sim} \mathbf{SubAGraphs}_{\text{PNTG}}$.

Proof.

As the first step we construct the restricted functor $\mathcal{F}_{\text{PTIC}} : \mathbf{PTINet}_{|\mathcal{M}_1} \rightarrow \mathbf{SubAGraphs}_{\text{PNTG}}$ exactly as given in Definition 64. $\mathcal{F}_{\text{PTIC}}$ defined in this way is a well-defined restricted functor as it is shown in the proof of Lemma 42. Now we have to define the inverse restricted functor $\mathcal{F}_{\text{PTIC}}^{-1} : \mathbf{SubAGraphs}_{\text{PNTG}} \rightarrow \mathbf{PTINet}_{|\mathcal{M}_1}$ on objects and morphisms as given in the following. Consider a typed attributed graph $((G', \text{NAT}^1), \text{type})$ in $\mathbf{SubAGraphs}_{\text{PNTG}}$ with the E-graph $G' =$

$(V_G^{G'}, V_D^{G'} = \mathbb{N}, E_G^{G'}, E_{NA}^{G'}, E_{EA}^{G'}, (s_j^{G'}, t_j^{G'})_{j \in \{G, NA, EA\}})$ and morphism $type : (G', NAT) \rightarrow (PNTG, D_{fin})$ given by a final morphism of data types from NAT to the final algebra D_{fin} and $type^{G'} : G' \rightarrow PNTG$ is given by E-graph morphism $type^{G'} = (type_{V_G}, type_{V_D}, type_{E_G}, type_{E_{NA}}, type_{E_{EA}})$. Since the category **SubAGraphs_{PNTG}** contains only \mathcal{F}_{PTIC} -images, we have that $((G', NAT), type) = \mathcal{F}_{PTIC}(G)$ for some PTI net $G = (P_G, T_G, pre_G, post_G, I_G, m_G)$. We define the object $\mathcal{F}_{PTIC}^{-1}(G') = (P_{\mathcal{F}_{PTIC}^{-1}(G')}, T_{\mathcal{F}_{PTIC}^{-1}(G')}, pre_{\mathcal{F}_{PTIC}^{-1}(G')}, post_{\mathcal{F}_{PTIC}^{-1}(G')}, I_{\mathcal{F}_{PTIC}^{-1}(G')}, m_{\mathcal{F}_{PTIC}^{-1}(G')})$ as follows:

$$\begin{aligned} P_{\mathcal{F}_{PTIC}^{-1}(G')} &= \{p \in V_G^{G'} \mid type_{V_G}(p) = Place\} \\ T_{\mathcal{F}_{PTIC}^{-1}(G')} &= \{t \in V_G^{G'} \mid type_{V_G}(t) = Trans\} \\ pre_{\mathcal{F}_{PTIC}^{-1}(G')}(t)(p) &= n \text{ if } ((p, t), n) \in E_{EA}^{G'} \\ post_{\mathcal{F}_{PTIC}^{-1}(G')}(t)(p) &= n \text{ if } ((t, p), n) \in E_{EA}^{G'} \\ I_{\mathcal{F}_{PTIC}^{-1}(G')} &= \{x \in V_G^{G'} \mid type_{V_G}(x) = Token\} \\ m_{\mathcal{F}_{PTIC}^{-1}(G')} &= \{(x, p) \in E_G^{G'} \mid type_{E_G}(x, p) = to2p\} \end{aligned}$$

Consider additionally another typed attributed graph $((H', NAT), type')$ in **SubAGraphs_{PNTG}** with the E-graph $H' = (V_G^{H'}, V_D^{H'} = \mathbb{N}, E_G^{H'}, E_{NA}^{H'}, E_{EA}^{H'}, (s_j^{H'}, t_j^{H'})_{j \in \{G, NA, EA\}})$ and the morphism $type' : (H', NAT) \rightarrow (PNTG, D_{fin})$. For each typed attributed graph morphism $f' : ((G', NAT), type) \rightarrow ((H', NAT), type')$ in **SubAGraphs_{PNTG}** with $f' = (f'_{V_G} : V_G^{G'} \rightarrow V_G^{H'}, f'_{V_D} : \mathbb{N} \rightarrow \mathbb{N}, f'_{E_G} : E_G^{G'} \rightarrow E_G^{H'}, f'_{E_{NA}} : E_{NA}^{G'} \rightarrow E_{NA}^{H'}, f'_{E_{EA}} : E_{EA}^{G'} \rightarrow E_{EA}^{H'})$, we define $\mathcal{F}_{PTIC}^{-1}(f') : \mathcal{F}_{PTIC}^{-1}(G') \rightarrow \mathcal{F}_{PTIC}^{-1}(H')$ where $\mathcal{F}_{PTIC}^{-1}(G') = (P_{\mathcal{F}_{PTIC}^{-1}(G')}, T_{\mathcal{F}_{PTIC}^{-1}(G')}, pre_{\mathcal{F}_{PTIC}^{-1}(G')}, post_{\mathcal{F}_{PTIC}^{-1}(G')}, I_{\mathcal{F}_{PTIC}^{-1}(G')}, m_{\mathcal{F}_{PTIC}^{-1}(G')})$ and $\mathcal{F}_{PTIC}^{-1}(H') = (P_{\mathcal{F}_{PTIC}^{-1}(H')}, T_{\mathcal{F}_{PTIC}^{-1}(H')}, pre_{\mathcal{F}_{PTIC}^{-1}(H')}, post_{\mathcal{F}_{PTIC}^{-1}(H')}, I_{\mathcal{F}_{PTIC}^{-1}(H')}, m_{\mathcal{F}_{PTIC}^{-1}(H')})$ by $\mathcal{F}_{PTIC}^{-1}(f') = f = (f_P, f_T, f_I)$ where:

$$\begin{aligned} f_P : P_{\mathcal{F}_{PTIC}^{-1}(G')} &\rightarrow P_{\mathcal{F}_{PTIC}^{-1}(H')} \text{ with } f_P(p) = f'_{V_G}(p) \\ f_T : T_{\mathcal{F}_{PTIC}^{-1}(G')} &\rightarrow T_{\mathcal{F}_{PTIC}^{-1}(H')} \text{ with } f_T(t) = f'_{V_G}(t) \\ f_I : I_{\mathcal{F}_{PTIC}^{-1}(G')} &\rightarrow I_{\mathcal{F}_{PTIC}^{-1}(H')} \text{ with } f_I(x) = f'_{V_G}(x) \end{aligned}$$

In the next step we have to show, that the restricted functor \mathcal{F}_{PTIC}^{-1} introduced above is a well-defined restricted functor. For this reason we have to show the following for an arbitrary object $((G', NAT), type)$ as well as for an arbitrary morphism $f' : ((G', NAT), type) \rightarrow ((H', NAT), type')$ in **SubAGraphs_{PNTG}**:

1. **The components of $\mathcal{F}_{PTIC}^{-1}(G')$ are well-defined w.r.t. codomain.**

Consider a typed attributed graph $((G', NAT), type)$ in **SubAGraphs_{PNTG}** with the E-graph $G' = (V_G^{G'}, V_D^{G'} = \mathbb{N}, E_G^{G'}, E_{NA}^{G'}, E_{EA}^{G'}, (s_j^{G'}, t_j^{G'})_{j \in \{G, NA, EA\}})$ and the morphism $type : (G', NAT) \rightarrow (PNTG, D_{fin})$ with $type^{G'} = (type_{V_G}, type_{V_D}, type_{E_G}, type_{E_{NA}}, type_{E_{EA}})$. We have to show that $\mathcal{F}_{PTIC}^{-1}(G') = (P_{\mathcal{F}_{PTIC}^{-1}(G')}, T_{\mathcal{F}_{PTIC}^{-1}(G')}, pre_{\mathcal{F}_{PTIC}^{-1}(G')}, post_{\mathcal{F}_{PTIC}^{-1}(G')}, I_{\mathcal{F}_{PTIC}^{-1}(G')}, m_{\mathcal{F}_{PTIC}^{-1}(G')})$ is a well-defined PTI net. For this reason it suffices to show that the corresponding components of $\mathcal{F}_{PTIC}^{-1}(G')$ and G are equal.

- $P_{\mathcal{F}_{PTIC}^{-1}(G')} = P_G :$

$$P_{\mathcal{F}_{PTIC}^{-1}(G')} \stackrel{Def. \mathcal{F}_{PTIC}^{-1}}{=} \{p \in V_G^{G'} \mid type_{V_G}(p) = Place\}$$

¹ Similarly to Section 8.2, we consider the category **SubAGraphs_{PNTG}** with the fixed data type NAT and an identical algebra homomorphism, which implies that the V_D -component of morphisms is an identity.

$$\begin{aligned} V_G^{G'} &\stackrel{\text{from Def. 64}}{=} \{p \in (P_G \uplus T_G \uplus I_G) \mid \text{type}_{V_G}(p) = \text{Place}\} \\ \text{type}_{V_G} &\stackrel{\text{from Def. 64}}{=} P_G \end{aligned}$$

- $T_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} = T_G :$

$$\begin{aligned} T_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} &\stackrel{\text{Def. } \mathcal{F}_{\text{PTIC}}^{-1}}{=} \{t \in V_G^{G'} \mid \text{type}_{V_G}(t) = \text{Trans}\} \\ V_G^{G'} &\stackrel{\text{from Def. 64}}{=} \{t \in (P_G \uplus T_G \uplus I_G) \mid \text{type}_{V_G}(t) = \text{Trans}\} \\ \text{type}_{V_G} &\stackrel{\text{from Def. 64}}{=} T_G \end{aligned}$$

- $\text{pre}_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} = \text{pre}_G :$

We have to show the following:

$$\forall t \in T_G, p \in P_G. \text{pre}_{\mathcal{F}_{\text{PTIC}}^{-1}(G')}(t)(p) = \text{pre}_G(t)(p).$$

Fix $t \in T_G$ and $p \in P_G$. Then it holds:

$$\begin{aligned} \text{pre}_G(t)(p) &= \text{pre}_G(t)(p) \\ E_{EA}^{G'} &\stackrel{\text{from Def. 64}}{\Rightarrow} ((p, t), \text{pre}_G(t)(p)) \in E_{EA}^{G'} \\ \text{Def. } \mathcal{F}_{\text{PTIC}}^{-1} &\Rightarrow \text{pre}_{\mathcal{F}_{\text{PTIC}}^{-1}(G')}(t)(p) = \text{pre}_G(t)(p) \end{aligned}$$

- $\text{post}_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} = \text{post}_G :$

We have to show the following:

$$\forall t \in T_G, p \in P_G. \text{post}_{\mathcal{F}_{\text{PTIC}}^{-1}(G')}(t)(p) = \text{post}_G(t)(p).$$

Fix $t \in T_G$ and $p \in P_G$. Then it holds:

$$\begin{aligned} \text{post}_G(t)(p) &= \text{post}_G(t)(p) \\ E_{EA}^{G'} &\stackrel{\text{from Def. 64}}{\Rightarrow} ((t, p), \text{post}_G(t)(p)) \in E_{EA}^{G'} \\ \text{Def. } \mathcal{F}_{\text{PTIC}}^{-1} &\Rightarrow \text{post}_{\mathcal{F}_{\text{PTIC}}^{-1}(G')}(t)(p) = \text{post}_G(t)(p) \end{aligned}$$

- $I_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} = I_G :$

$$\begin{aligned} I_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} &\stackrel{\text{Def. } \mathcal{F}_{\text{PTIC}}^{-1}}{=} \{x \in V_G^{G'} \mid \text{type}_{V_G}(x) = \text{Token}\} \\ V_G^{G'} &\stackrel{\text{from Def. 64}}{=} \{x \in (P_G \uplus T_G \uplus I_G) \mid \text{type}_{V_G}(x) = \text{Token}\} \\ \text{type}_{V_G} &\stackrel{\text{from Def. 64}}{=} I_G \end{aligned}$$

- $m_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} = m_G :$

$$\begin{aligned} m_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} &\stackrel{\text{Def. } \mathcal{F}_{\text{PTIC}}^{-1}}{=} \{(x, p) \in E_G^{G'} \mid \text{type}_{E_G}(x, p) = \text{to2p}\} \\ E_G^{G'} &\stackrel{\text{from Def. 64}}{=} \{(x, p) \in (E_{\text{to2p}}^{G'} \uplus E_{\text{p2t}}^{G'} \uplus E_{\text{t2p}}^{G'}) \mid \text{type}_{E_G}(x, p) = \text{to2p}\} \\ \text{type}_{E_G} &\stackrel{\text{from Def. 64}}{=} E_{\text{to2p}}^{G'} \stackrel{E_{\text{to2p}}^{G'} \text{ from Def. 64}}{=} \{(x, p) \in I_G \times P_G \mid m_G(x) = p\} \\ &= m_G \end{aligned}$$

2. **The components of $\mathcal{F}_{\text{PTIC}}^{-1}(f')$ are well-defined w.r.t. codomain.**

Consider a typed attributed graph morphism $f' : ((G', \text{NAT}), \text{type}) \rightarrow ((H', \text{NAT}), \text{type}')$ in **SubAGraphs_{PNTG}** with $f' = (f'_{V_G} : V_G^{G'} \rightarrow V_G^{H'}, f'_{V_D} : \mathbb{N} \rightarrow \mathbb{N}, f'_{E_G} : E_G^{G'} \rightarrow E_G^{H'}, f'_{E_{NA}} : E_{NA}^{G'} \rightarrow E_{NA}^{H'}, f'_{E_{EA}} : E_{EA}^{G'} \rightarrow E_{EA}^{H'})$. Since the category **SubAGraphs_{PNTG}** contains only $\mathcal{F}_{\text{PTIC}}$ -images, we have that $f' = \mathcal{F}_{\text{PTIC}}(f)$ for some injective PTI net morphism $f = (f_P : P_G \rightarrow P_H, f_T : T_G \rightarrow T_H, f_I : I_G \rightarrow I_H)$. We have to show that $\mathcal{F}_{\text{PTIC}}^{-1}(f') = ((\mathcal{F}_{\text{PTIC}}^{-1}(f'))_P : P_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} \rightarrow P_{\mathcal{F}_{\text{PTIC}}^{-1}(H')}, (\mathcal{F}_{\text{PTIC}}^{-1}(f'))_T : T_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} \rightarrow T_{\mathcal{F}_{\text{PTIC}}^{-1}(H')}, (\mathcal{F}_{\text{PTIC}}^{-1}(f'))_I : I_{\mathcal{F}_{\text{PTIC}}^{-1}(G')} \rightarrow I_{\mathcal{F}_{\text{PTIC}}^{-1}(H')})$ is a well-defined injective PTI net morphism. It suffices to show that the corresponding components of $\mathcal{F}_{\text{PTIC}}^{-1}(f')$ and f are equal.

- $(\mathcal{F}_{\text{PTIC}}^{-1}(f'))_P = f_P :$
Fix $p \in P_{\mathcal{F}_{\text{PTIC}}^{-1}(G')}$.

$$(\mathcal{F}_{\text{PTIC}}^{-1}(f'))_P(p) \stackrel{\text{Def. } \mathcal{F}_{\text{PTIC}}^{-1}}{=} f'_{V_G}(p) \stackrel{f'_{V_G} \text{ from Def. 64}}{=} (f_P \uplus f_T \uplus f_I)(p) = f_P(p)$$

- $(\mathcal{F}_{\text{PTIC}}^{-1}(f'))_T = f_T :$
Fix $t \in T_{\mathcal{F}_{\text{PTIC}}^{-1}(G')}$.

$$(\mathcal{F}_{\text{PTIC}}^{-1}(f'))_T(t) \stackrel{\text{Def. } \mathcal{F}_{\text{PTIC}}^{-1}}{=} f'_{V_G}(t) \stackrel{f'_{V_G} \text{ from Def. 64}}{=} (f_P \uplus f_T \uplus f_I)(t) = f_T(t)$$

- $(\mathcal{F}_{\text{PTIC}}^{-1}(f'))_I = f_I :$
Fix $x \in I_{\mathcal{F}_{\text{PTIC}}^{-1}(G')}$.

$$(\mathcal{F}_{\text{PTIC}}^{-1}(f'))_I(x) \stackrel{\text{Def. } \mathcal{F}_{\text{PTIC}}^{-1}}{=} f'_{V_G}(x) \stackrel{f'_{V_G} \text{ from Def. 64}}{=} (f_P \uplus f_T \uplus f_I)(x) = f_I(x)$$

3. **Compositionality axiom holds for $\mathcal{F}_{\text{PTIC}}^{-1}$.**

Consider morphisms $g' : ((G', \text{NAT}), \text{type}) \rightarrow ((H', \text{NAT}), \text{type}')$ and $f' : ((H', \text{NAT}), \text{type}') \rightarrow ((D', \text{NAT}), \text{type}'')$ in **SubAGraphs_{PNTG}**. We have to show that $\mathcal{F}_{\text{PTIC}}^{-1}(f' \circ g') = \mathcal{F}_{\text{PTIC}}^{-1}(f') \circ \mathcal{F}_{\text{PTIC}}^{-1}(g')$. Since the category **SubAGraphs_{PNTG}** contains only $\mathcal{F}_{\text{PTIC}}$ -images, we have that $g' = \mathcal{F}_{\text{PTIC}}(g)$ and $f' = \mathcal{F}_{\text{PTIC}}(f)$ for injective PTI net morphisms $g : G \rightarrow H$, $f : H \rightarrow D$. Thus, it suffices to show that: $\mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(f) \circ \mathcal{F}_{\text{PTIC}}(g)) = \mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(f)) \circ \mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(g))$. It holds the following:

$$\begin{aligned} & \mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(f) \circ \mathcal{F}_{\text{PTIC}}(g)) \\ & \stackrel{\text{funct. prop.}}{=} \mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(f \circ g)) \\ & \stackrel{\mathcal{F}_{\text{PTIC}}^{-1} \text{ well-def. on morph.}}{=} f \circ g \\ & \stackrel{\mathcal{F}_{\text{PTIC}}^{-1} \text{ well-def. on morph.}}{=} \mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(f)) \circ \mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(g)) \end{aligned}$$

4. **Identity axiom holds for $\mathcal{F}_{\text{PTIC}}^{-1}$.**

Consider an identity morphism $\text{id}_{X'} : ((X', \text{NAT}), \text{type}) \rightarrow ((X', \text{NAT}), \text{type})$ in **SubAGraphs_{PNTG}**. We have to show that $\mathcal{F}_{\text{PTIC}}^{-1}(\text{id}_{X'}) = \text{id}_{\mathcal{F}_{\text{PTIC}}^{-1}(X')}$. Since the category **SubAGraphs_{PNTG}** contains only $\mathcal{F}_{\text{PTIC}}$ -images, we have that $X' = \mathcal{F}_{\text{PTIC}}(X)$ for some $X \in \text{Ob}_{\text{PTINet}[\mathcal{M}_1]}$. Thus, it suffices to show that: $\mathcal{F}_{\text{PTIC}}^{-1}(\text{id}_{\mathcal{F}_{\text{PTIC}}(X)}) = \text{id}_{\mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(X))}$. It holds the following:

$$\begin{aligned} & \mathcal{F}_{\text{PTIC}}^{-1}(\text{id}_{\mathcal{F}_{\text{PTIC}}(X)}) \stackrel{\mathcal{F}_{\text{PTIC}} \text{ is funct.}}{=} \mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(\text{id}_X)) \\ & \stackrel{\mathcal{F}_{\text{PTIC}}^{-1} \text{ well-def. on morph.}}{=} \text{id}_X \stackrel{\mathcal{F}_{\text{PTIC}}^{-1} \text{ well-def. on morph.}}{=} \text{id}_{\mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(X))} \end{aligned}$$

It remains now to show that $\mathcal{F}_{\text{PTIC}} : \mathbf{PTINet}|_{\mathcal{M}_1} \xrightarrow{\sim} \mathbf{SubAGraphs}_{\text{PNTG}}$ is a category equivalence. For this reason, according to Definitions 66 and 67, we have to show that:

$$\begin{aligned} & \forall A \in \text{Ob}_{\mathbf{PTINet}|_{\mathcal{M}_1}}. \exists t_1(A) : (\mathcal{F}_{\text{PTIC}}^{-1} \circ \mathcal{F}_{\text{PTIC}})(A) \rightarrow \text{Id}_{\mathbf{PTINet}|_{\mathcal{M}_1}}(A). \\ & t_1(A) \text{ is an isomorphism in } \mathbf{PTINet}|_{\mathcal{M}_1} \wedge \\ & \forall A' \in \text{Ob}_{\mathbf{SubAGraphs}_{\text{PNTG}}}. \exists t_2(A') : (\mathcal{F}_{\text{PTIC}} \circ \mathcal{F}_{\text{PTIC}}^{-1})(A') \rightarrow \text{Id}_{\mathbf{SubAGraphs}_{\text{PNTG}}}(A'). \\ & t_2(A') \text{ is an isomorphism in } \mathbf{SubAGraphs}_{\text{PNTG}} \end{aligned}$$

Fix objects $A \in \text{Ob}_{\mathbf{PTINet}|_{\mathcal{M}_1}}$, $A' \in \text{Ob}_{\mathbf{SubAGraphs}_{\text{PNTG}}}$ and let $t_1(A) = \text{id}_A$ and $t_2(A') = \text{id}_{A'}$. Obviously, id_A and $\text{id}_{A'}$ are isomorphisms in $\mathbf{PTINet}|_{\mathcal{M}_1}$, $\mathbf{SubAGraphs}_{\text{PNTG}}$, respectively. Furthermore, it holds the following:

$$\begin{aligned} t_1(A) &= \text{id}_A : A \rightarrow A \\ &\stackrel{\mathcal{F}_{\text{PTIC}}^{-1} \text{ well-def. on obj.}}{=} \text{id}_A : (\mathcal{F}_{\text{PTIC}}^{-1} \circ \mathcal{F}_{\text{PTIC}})(A) \rightarrow A \\ &\stackrel{\text{Def. Id}_{\mathbf{PTINet}|_{\mathcal{M}_1}}}{=} \text{id}_A : (\mathcal{F}_{\text{PTIC}}^{-1} \circ \mathcal{F}_{\text{PTIC}})(A) \rightarrow \text{Id}_{\mathbf{PTINet}|_{\mathcal{M}_1}}(A) \end{aligned}$$

and

$$\begin{aligned} t_2(A') &= \text{id}_{A'} : A' \rightarrow A' \\ &\stackrel{\text{Def. Id}_{\mathbf{SubAGraphs}_{\text{PNTG}}}}{=} \text{id}_{A'} : A' \rightarrow \text{Id}_{\mathbf{SubAGraphs}_{\text{PNTG}}}(A') \\ &\stackrel{(*)}{=} \text{id}_{A'} : (\mathcal{F}_{\text{PTIC}} \circ \mathcal{F}_{\text{PTIC}}^{-1})(A') \rightarrow \text{Id}_{\mathbf{SubAGraphs}_{\text{PNTG}}}(A') \end{aligned}$$

(*): This step is possible since we know that the restricted functor $\mathcal{F}_{\text{PTIC}}^{-1}$ is well-defined on objects and for an arbitrary object $B \in \text{Ob}_{\mathbf{PTINet}|_{\mathcal{M}_1}}$ it holds the following:

$$\begin{aligned} & (\mathcal{F}_{\text{PTIC}}^{-1} \circ \mathcal{F}_{\text{PTIC}})(B) = B \\ & \stackrel{\text{funct. prop.}}{\Leftrightarrow} \mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(B)) = B \\ & \Leftrightarrow \mathcal{F}_{\text{PTIC}}(\mathcal{F}_{\text{PTIC}}^{-1}(\mathcal{F}_{\text{PTIC}}(B))) = \mathcal{F}_{\text{PTIC}}(B) \\ & \stackrel{A' = \mathcal{F}_{\text{PTIC}}(B)}{\Leftrightarrow} \mathcal{F}_{\text{PTIC}}(\mathcal{F}_{\text{PTIC}}^{-1}(A')) = A' \\ & \stackrel{\text{funct. prop.}}{\Leftrightarrow} (\mathcal{F}_{\text{PTIC}} \circ \mathcal{F}_{\text{PTIC}}^{-1})(A') = A' \end{aligned}$$

Thus, we get that $\mathcal{F}_{\text{PTIC}} : \mathbf{PTINet}|_{\mathcal{M}_1} \xrightarrow{\sim} \mathbf{SubAGraphs}_{\text{PNTG}}$ is a category equivalence² implying altogether that the categories $\mathbf{PTINet}|_{\mathcal{M}_1}$ and $\mathbf{SubAGraphs}_{\text{PNTG}}$ are equivalent. \square

Lemma 65: (I_{PTI} Satisfies Required Properties, see page 223)

Consider a transformation system $(\mathbf{SubAGraphs}_{\text{PNTG}}, \mathcal{F}_{\text{PTI}}(P))$ with a distinguished class of monomorphisms \mathcal{M}_2^* , an \mathcal{M} -adhesive transformation system $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2, \mathcal{F}_{\text{PTI}}(P))$, and a restricted functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$. Then the restricted inclusion functor $\text{I}_{\text{PTI}} : \mathbf{SubAGraphs}_{\text{PNTG}} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ satisfies the properties listed in Definition 70.

Proof.

Consider \mathcal{M} -adhesive transformation system $(\mathbf{PTINet}, \mathcal{M}_1, P)$, the restricted functor $\mathcal{F}_{\text{PTI}} : \mathbf{PTINet}|_{\mathcal{M}_1} \rightarrow \mathbf{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ (see Definition 64), the subcategory of typed attributed graphs $\mathbf{SubAGraphs}_{\text{PNTG}}$ with $\text{Ob}_{\mathbf{SubAGraphs}_{\text{PNTG}}} = \mathcal{F}_{\text{PTI}}(\text{Ob}_{\mathbf{PTINet}|_{\mathcal{M}_1}})$, $\text{Mor}_{\mathbf{SubAGraphs}_{\text{PNTG}}}(\mathcal{F}_{\text{PTI}}(A),$

² In this case, we even have a category isomorphism according to Definition 68 since it obviously holds that $\mathcal{F}_{\text{PTIC}} \circ \mathcal{F}_{\text{PTIC}}^{-1} = \text{Id}_{\mathbf{SubAGraphs}_{\text{PNTG}}}$ and $\mathcal{F}_{\text{PTIC}}^{-1} \circ \mathcal{F}_{\text{PTIC}} = \text{Id}_{\mathbf{PTINet}|_{\mathcal{M}_1}}$.

$\mathcal{F}_{\text{PTI}}(B)) = \mathcal{F}_{\text{PTI}}(\text{Mor}_{\text{PTINet}|_{\mathcal{M}_1}}(A, B))$ for arbitrary $A, B \in \text{Ob}_{\text{PTINet}|_{\mathcal{M}_1}}$, and restricted functors $\mathcal{F}_{\text{PTIC}} : \text{PTINet}|_{\mathcal{M}_1} \rightarrow \text{SubAGraphs}_{\text{PNTG}}$, $\mathcal{F}_{\text{PTIC}}^{-1} : \text{SubAGraphs}_{\text{PNTG}} \rightarrow \text{PTINet}|_{\mathcal{M}_1}$ building the category equivalence $\mathcal{F}_{\text{PTIC}} : \text{PTINet}|_{\mathcal{M}_1} \xrightarrow{\sim} \text{SubAGraphs}_{\text{PNTG}}$. Let furthermore, $((G, \text{NAT}), \text{type})$ be a typed attributed graph in $\text{SubAGraphs}_{\text{PNTG}}$ with the E-graph $G = (V_G^G, V_D^G = \mathbb{N}, E_G^G, E_{\text{NA}}^G, E_{\text{EA}}^G, (s_j^G, t_j^G)_{j \in \{G, \text{NA}, \text{EA}\}})$ and the morphism $\text{type} : (G, \text{NAT}) \rightarrow (\text{PNTG}, D_{\text{fin}})$ given by a final morphism of data types from NAT to the final algebra D_{fin} and $\text{type}^G : G \rightarrow \text{PNTG}$ with $\text{type}^G = (\text{type}_{V_G}, \text{type}_{V_D}, \text{type}_{E_G}, \text{type}_{E_{\text{NA}}}, \text{type}_{E_{\text{EA}}})$.

In the following we have to show that the restricted inclusion functor $I_{\text{PTI}} : \text{SubAGraphs}_{\text{PNTG}} \rightarrow \text{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ satisfies properties from Definition 70:

1. I_{PTI} **preserves monomorphisms, i.e.**, $I_{\text{PTI}}(\mathcal{M}_2^*) \subseteq \mathcal{M}_2$:

By definition of I_{PTI} , we know that $I_{\text{PTI}}(\mathcal{M}_2^*) = \mathcal{M}_2^*$. Furthermore, we have by construction of \mathcal{M}_2^* that $\mathcal{M}_2^* \subseteq \mathcal{M}_2$.

2. I_{PTI} **preserves pushouts of \mathcal{M} -morphisms:**

Let $(D, g_1 : B \rightarrow D, g_2 : C \rightarrow D)$ be a pushout in $\text{SubAGraphs}_{\text{PNTG}}$ with $f_1 : A \rightarrow B$ and $f_2 : A \rightarrow C$ in \mathcal{M}_2^* . Since pushouts in PTINet are constructed componentwise and the restricted functor $\mathcal{F}_{\text{PTIC}}$ preserves the componentwise construction, pushouts in $\text{SubAGraphs}_{\text{PNTG}}$ are constructed componentwise as well and we have that the V_G^- , V_D^- , E_G^- , E_{NA}^- and E_{EA}^- -components of (1) are pushouts in **Sets**. Thus, also $(I_{\text{PTI}}(D), I_{\text{PTI}}(g_1) : I_{\text{PTI}}(B) \rightarrow I_{\text{PTI}}(D), I_{\text{PTI}}(g_2) : I_{\text{PTI}}(C) \rightarrow I_{\text{PTI}}(D))$ with $I_{\text{PTI}}(D) = D$, $I_{\text{PTI}}(g_1) = g_1$ and $I_{\text{PTI}}(g_2) = g_2$ is constructed componentwise in $\text{AGraphs}_{\text{PNTG}}|_{\mathcal{M}_2}$ and hence also in $\text{AGraphs}_{\text{PNTG}}$ such that the V_G^- , V_D^- , E_G^- , E_{NA}^- and E_{EA}^- -components of (2) are pushouts in **Sets**. This implies that (2) is a pushout in $\text{AGraphs}_{\text{PNTG}}$. Furthermore, since I_{PTI} preserves monomorphisms according to the property shown before, we have for $f_1, f_2 \in \mathcal{M}_2^*$ that $I_{\text{PTI}}(f_1), I_{\text{PTI}}(f_2) \in \mathcal{M}_2$.

$$\begin{array}{ccccc}
 A & \xrightarrow{f_1 \in \mathcal{M}_2^*} & B & & I_{\text{PTI}}(A)=A \xrightarrow{(I_{\text{PTI}}(f_1)=f_1) \in \mathcal{M}_2} I_{\text{PTI}}(B)=B \\
 \downarrow f_2 \in \mathcal{M}_2^* & & \downarrow g_1 & & \downarrow I_{\text{PTI}}(g_1)=g_1 \\
 C & \xrightarrow{g_2} & D & & I_{\text{PTI}}(C)=C \xrightarrow{I_{\text{PTI}}(g_2)=g_2} I_{\text{PTI}}(D)=D \\
 & & & & \uparrow I_{\text{PTI}}(f_2)=f_2 \in \mathcal{M}_2 \\
 & & & & (2)
 \end{array}$$

3. I_{PTI} **creates \mathcal{M} -morphisms:**

Consider an \mathcal{M}_2 -morphism $f' : I_{\text{PTI}}(A) \rightarrow I_{\text{PTI}}(B)$ in $\text{Mor}_{\text{AGraphs}_{\text{PNTG}}}$. We have to show that there is exactly one \mathcal{M}_2^* -morphism $f : A \rightarrow B$ in $\text{Mor}_{\text{SubAGraphs}_{\text{PNTG}}}$ such that $I_{\text{PTI}}(f) = f'$. f' satisfies already the existence property with $f' : A \rightarrow B$ and $I_{\text{PTI}}(f') = f'$, because $I_{\text{PTI}}(A) = A$ and $I_{\text{PTI}}(B) = B$. Since the category $\text{SubAGraphs}_{\text{PNTG}}$ contains only \mathcal{M}_2^* -morphisms, f' is an \mathcal{M}_2^* -morphism. Finally, uniqueness of f' follows from the injectivity of the inclusion functor I_{PTI} .

4. I_{PTI} **preserves initial pushouts over \mathcal{M} -morphisms:**

Consider initial pushout (2) over the \mathcal{M}_2^* -morphism $\mathcal{F}_{\text{PTIC}}(f) : \mathcal{F}_{\text{PTIC}}(L) \rightarrow \mathcal{F}_{\text{PTIC}}(G)$ in $\text{SubAGraphs}_{\text{PNTG}}$ consisting of $\mathcal{F}_{\text{PTIC}}$ -images only with boundary object $\mathcal{F}_{\text{PTIC}}(B)$, context object $\mathcal{F}_{\text{PTIC}}(C)$ and morphisms $\mathcal{F}_{\text{PTIC}}(b), \mathcal{F}_{\text{PTIC}}(c) \in \mathcal{M}_2^*$. This initial pushout is the $\mathcal{F}_{\text{PTIC}}$ -translation of the diagram (1) with boundary object B , context object C constructed over the \mathcal{M}_1 -morphism $f : L \rightarrow G$ as given in Remark 9 resp. Fact 10 and morphisms $b, c \in \mathcal{M}_1$, which is an initial pushout in $\text{PTINet}|_{\mathcal{M}_1}$ and hence also in PTINet since $\mathcal{F}_{\text{PTIC}}^{-1} : \text{SubAGraphs}_{\text{PNTG}} \xrightarrow{\sim} \text{PTINet}|_{\mathcal{M}_1}$ is a category equivalence as well and hence the restricted functor $\mathcal{F}_{\text{PTIC}}^{-1} : \text{SubAGraphs}_{\text{PNTG}} \rightarrow \text{PTINet}|_{\mathcal{M}_1}$ from this category equivalence preserves initial pushouts by the corresponding instantiation of Lemma 60.

We have to show that the diagram (3) is an initial pushout in $\mathbf{AGraphs}_{\mathbf{PNTG}}|_{\mathcal{M}_2}$ and hence also in $\mathbf{AGraphs}_{\mathbf{PNTG}}$.

Let $\mathcal{F}_{\mathbf{PTIC}}(B) = B^* = ((B_0^*, \mathbf{NAT}), \text{type})$ be given according to Lemma 46. Since $\mathbf{I}_{\mathbf{PTI}}$ preserves monomorphisms and pushouts of \mathcal{M} -morphisms as already shown before, we have that (3) is a pushout in $\mathbf{AGraphs}_{\mathbf{PNTG}}|_{\mathcal{M}_2}$ and hence also in $\mathbf{AGraphs}_{\mathbf{PNTG}}$ with \mathcal{M}_2 -morphisms $\mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(b)) = \mathcal{F}_{\mathbf{PTIC}}(b)$ and $\mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(c)) = \mathcal{F}_{\mathbf{PTIC}}(c)$. Furthermore, by definition of $\mathbf{I}_{\mathbf{PTI}}$ we obviously have that the diagrams (2) and (3) are isomorphic. Now we construct the initial pushout (4) over the \mathcal{M}_2 -morphism $\mathcal{F}_{\mathbf{PTIC}}(f) : \mathcal{F}_{\mathbf{PTIC}}(L) \rightarrow \mathcal{F}_{\mathbf{PTIC}}(G)$ in $\mathbf{AGraphs}_{\mathbf{PNTG}}$ with boundary $B' = ((B_0', \mathbf{NAT}), \text{type})$ defined according to Lemma 48 and morphisms $b' : B' \rightarrow \mathcal{F}_{\mathbf{PTIC}}(L)$, $c' : C' \rightarrow \mathcal{F}_{\mathbf{PTIC}}(G)$ in \mathcal{M}_2 .

The initiality of (4) implies the existence of unique morphisms $i : B' \rightarrow \mathcal{F}_{\mathbf{PTIC}}(B)$ and $j : C' \rightarrow \mathcal{F}_{\mathbf{PTIC}}(C)$ such that (5) is a pushout in $\mathbf{AGraphs}_{\mathbf{PNTG}}$ and (6), (7) commute with $i, j \in \mathcal{M}_2$. The surjectivity of $i : B' \rightarrow \mathcal{F}_{\mathbf{PTIC}}(B)$, which can be shown by the fact that $\mathcal{F}_{\mathbf{PTIC}}(B) = B^* \subseteq B'$, follows from the surjectivity proof given for Lemma 50 replacing $\mathcal{F}_{\mathbf{PTI}}$ by $\mathbf{I}_{\mathbf{PTI}} \circ \mathcal{F}_{\mathbf{PTIC}}$. Thus, i is injective and surjective, so we get that i is an isomorphism. Since (5) is a pushout, also $j : C' \rightarrow \mathcal{F}_{\mathbf{PTIC}}(C)$ is an isomorphism and hence (3) is isomorphic to (4). So we get that also (3) is an initial pushout over the \mathcal{M}_2 -morphism $\mathcal{F}_{\mathbf{PTIC}}(f) : \mathcal{F}_{\mathbf{PTIC}}(L) \rightarrow \mathcal{F}_{\mathbf{PTIC}}(G)$ in $\mathbf{AGraphs}_{\mathbf{PNTG}}$.

$$\begin{array}{ccc}
 B & \xrightarrow{b} & L \\
 g \downarrow & (1) & \downarrow f \in \mathcal{M}_1 \\
 C & \xrightarrow{c} & G
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathcal{F}_{\mathbf{PTIC}}(B) & \xrightarrow{\mathcal{F}_{\mathbf{PTIC}}(b)} & \mathcal{F}_{\mathbf{PTIC}}(L) \\
 \mathcal{F}_{\mathbf{PTIC}}(g) \downarrow & (2) & \downarrow \mathcal{F}_{\mathbf{PTIC}}(f) \in \mathcal{M}_2^* \\
 \mathcal{F}_{\mathbf{PTIC}}(C) & \xrightarrow{\mathcal{F}_{\mathbf{PTIC}}(c)} & \mathcal{F}_{\mathbf{PTIC}}(G)
 \end{array}$$

$$\begin{array}{ccc}
 & & b' \\
 & \swarrow & \searrow \\
 B' & \xrightarrow{i} & \mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(B)) \xrightarrow{\mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(b))} \mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(L)) \\
 g' \downarrow & (5) & \downarrow \mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(g)) \quad (3) \quad \downarrow \mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(f)) \in \mathcal{M}_2 \\
 C' & \xrightarrow{j} & \mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(C)) \xrightarrow{\mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(c))} \mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(G)) \\
 & & c'
 \end{array}$$

$$\begin{array}{ccc}
 B' & \xrightarrow{b'} & \mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(L)) \\
 g' \downarrow & (4) & \downarrow \mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(f)) \in \mathcal{M}_2^* \\
 C' & \xrightarrow{c'} & \mathbf{I}_{\mathbf{PTI}}(\mathcal{F}_{\mathbf{PTIC}}(G))
 \end{array}$$

5. $\mathbf{I}_{\mathbf{PTI}}$ preserves pullbacks of \mathcal{M} -morphisms:

Let $(A, f_1 : A \rightarrow B, f_2 : A \rightarrow C)$ be a pullback in $\mathbf{SubAGraphs}_{\mathbf{PNTG}}$ with $g_1 : B \rightarrow D$, $g_2 : C \rightarrow D$ in \mathcal{M}_2^* . Since pullbacks in \mathbf{PTINet} are constructed componentwise and the restricted functor $\mathcal{F}_{\mathbf{PTIC}}$ preserves the componentwise construction, pullbacks in $\mathbf{SubAGraphs}_{\mathbf{PNTG}}$ are constructed componentwise as well and we have that the V_{G^-} , V_{D^-} , E_{G^-} , E_{NA^-} and E_{EA^-} -components of (1) are pullbacks in **Sets**. Thus, also $(\mathbf{I}_{\mathbf{PTI}}(A), \mathbf{I}_{\mathbf{PTI}}(f_1) : \mathbf{I}_{\mathbf{PTI}}(A) \rightarrow \mathbf{I}_{\mathbf{PTI}}(B), \mathbf{I}_{\mathbf{PTI}}(f_2) : \mathbf{I}_{\mathbf{PTI}}(A) \rightarrow \mathbf{I}_{\mathbf{PTI}}(C))$ with $\mathbf{I}_{\mathbf{PTI}}(A) = A$, $\mathbf{I}_{\mathbf{PTI}}(f_1) = f_1$ and $\mathbf{I}_{\mathbf{PTI}}(f_2) = f_2$ is constructed componentwise in $\mathbf{AGraphs}_{\mathbf{PNTG}}|_{\mathcal{M}_2}$ and hence also in $\mathbf{AGraphs}_{\mathbf{PNTG}}$ such that the V_{G^-} , V_{D^-} , E_{G^-} , E_{NA^-} and E_{EA^-} -components of (2) are pullbacks in **Sets**. This implies that (2) is a pullback in $\mathbf{AGraphs}_{\mathbf{PNTG}}$. Fur-

thermore, since I_{PTI} preserves monomorphisms according to the first shown property of this proof, we have for $g_1, g_2 \in \mathcal{M}_2^*$ that $I_{PTI}(g_1), I_{PTI}(g_2) \in \mathcal{M}_2$.

$$\begin{array}{ccc}
 A & \xrightarrow{f_1} & B \\
 f_2 \downarrow & (1) & \downarrow g_1 \in \mathcal{M}_2^* \\
 C & \xrightarrow{g_2 \in \mathcal{M}_2^*} & D
 \end{array}
 \quad
 \begin{array}{ccc}
 I_{PTI}(A)=A & \xrightarrow{I_{PTI}(f_1)=f_1} & I_{PTI}(B)=B \\
 I_{PTI}(f_2)=f_2 \downarrow & (2) & \downarrow (I_{PTI}(g_1)=g_1) \in \mathcal{M}_2 \\
 I_{PTI}(C)=C & \xrightarrow{(I_{PTI}(g_2)=g_2) \in \mathcal{M}_2} & I_{PTI}(D)=D
 \end{array}$$

6. I_{PTI} preserves epimorphisms, i.e., $I_{PTI}(\mathcal{E}_2^*) \subseteq \mathcal{E}_2$:

By definition of I_{PTI} , we know that $I_{PTI}(\mathcal{E}_2^*) = \mathcal{E}_2^*$. Furthermore, we have by construction of \mathcal{E}_2^* that $\mathcal{E}_2^* \subseteq \mathcal{E}_2$.

7. I_{PTI} preserves coproducts of \mathcal{M} -morphisms:

Let $(A, (u_i)_{i \in I})$ be a coproduct in **SubAGraphs_{PNTG}** with $A \in \text{Ob}_{\text{SubAGraphs}_{\text{PNTG}}}$, a family of **SubAGraphs_{PNTG}**-morphisms $u_i : A_i \rightarrow A$ and an index set I . Since coproducts in **PTINet** and **PTINet** $_{|\mathcal{M}_1}$ are constructed componentwise and the restricted functor \mathcal{F}_{PTIC} preserves the componentwise construction, coproducts in **SubAGraphs_{PNTG}** are constructed componentwise as well and we have that the corresponding V_G -, V_D -, E_G -, E_{NA} - and E_{EA} -components of (1) are coproducts in **Sets**. Thus, it holds that also $(I_{PTI}(A), (I_{PTI}(u_i))_{i \in I})$ with $I_{PTI}(A) = A$ and $I_{PTI}(u_i) = u_i$ is constructed componentwise in **AGraphs_{PNTG}** $_{|\mathcal{M}_2}$ and hence also in **AGraphs_{PNTG}** such that the corresponding V_G -, V_D -, E_G -, E_{NA} - and E_{EA} -components of (2) are coproducts in **Sets**. This implies that $(I_{PTI}(A), (I_{PTI}(u_i))_{i \in I})$ is a coproduct in **AGraphs_{PNTG}**.

$$\begin{array}{ccc}
 A_i & \xrightarrow{u_i} & A \\
 & (1) & \downarrow f \\
 & f_i & B
 \end{array}
 \quad
 \begin{array}{ccc}
 I_{PTI}(A_i) & \xrightarrow{I_{PTI}(u_i)} & I_{PTI}(A) \\
 & (2) & \downarrow I_{PTI}(f) \\
 & I_{PTI}(f_i) & I_{PTI}(B)
 \end{array}$$

8. I_{PTI} preserves \mathcal{E}' -instances:

According to Definition 61, we have to show the following:

$$\forall (a', b') \in \mathcal{E}_2''. \exists (a'', b'') \in \mathcal{E}_2'. a'' = I_{PTI}(a') \wedge b'' = I_{PTI}(b')$$

Let (a', b') be a pair of jointly surjective morphisms in \mathcal{E}_2'' and define $a'' = I_{PTI}(a')$, $b'' = I_{PTI}(b')$. Then it remains to show that $(I_{PTI}(a'), I_{PTI}(b')) \in \mathcal{E}_2'$. For this reason it is obviously sufficient to show that I_{PTI} is compatible with pair factorization, which means by Definition 51 and Remark 13 that categories **SubAGraphs_{PNTG}** and **AGraphs_{PNTG}** have pair factorizations and I_{PTI} translates pair factorization in **SubAGraphs_{PNTG}** into the corresponding pair factorization in **AGraphs_{PNTG}**. According to [88], we know that the category **AGraphs_{ATG}** and hence also **AGraphs_{PNTG}** has $\mathcal{E}_2' - \mathcal{M}_2$ pair factorizations. In the next step, we want to show that also the subcategory **SubAGraphs_{PNTG}** has pair factorizations. As already mentioned in Section 10.1, we can construct an $\mathcal{E}_2' - \mathcal{M}_2^*$ pair factorization using binary coproducts and $\mathcal{E}_2^* - \mathcal{M}_2^*$ -factorization in **SubAGraphs_{PNTG}**. We have binary coproducts in **SubAGraphs_{PNTG}** because \emptyset is an initial object in **SubAGraphs_{PNTG}** and **SubAGraphs_{PNTG}** has pushouts of \mathcal{M}_2^* -morphisms since the restricted functor \mathcal{F}_{PTIC} preserves pushouts of \mathcal{M} -morphisms by application of Lemma 60. Moreover, we have $\mathcal{E}_2^* - \mathcal{M}_2^*$ -factorizations in **SubAGraphs_{PNTG}** because **PTINet** has $\mathcal{E}_1 - \mathcal{M}_1$ -factorizations according to Lemma 51, $\mathcal{E}_1 - \mathcal{M}_1$ -factorizations of monomorphisms in **PTINet** are preserved by the restriction to monomorphisms in **PTINet** $_{|\mathcal{M}_1}$ (because it holds that epimorphisms in $\mathcal{E}_1 - \mathcal{M}_1$ -factorizations of \mathcal{M}_1 -morphisms in **PTINet** are isomorphisms due to the decomposition property of \mathcal{M}_1 -morphisms), and the restricted functor

$\mathcal{F}_{\text{PTIC}}$ preserves $\mathcal{E} - \mathcal{M}$ -factorizations, which can be shown as follows: Let $A \xrightarrow{e} X \xrightarrow{m} B$ be an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization of the morphism $f : A \rightarrow B$ in $\mathbf{PTINet}_{|\mathcal{M}_1}$ with $e \in \mathcal{E}_1$ and $m \in \mathcal{M}_1$ according to Definition 10. Applying the restricted functor $\mathcal{F}_{\text{PTIC}}$ to the commuting triangle (1), we get a commuting triangle (2), because functors preserve commuting diagrams by general functor property. Furthermore, according to Lemma 60 the restricted functor $\mathcal{F}_{\text{PTIC}}$ preserves monomorphisms and epimorphisms, which implies that $\mathcal{F}_{\text{PTIC}}(e) \in \mathcal{E}_2^*$ and $\mathcal{F}_{\text{PTIC}}(m) \in \mathcal{M}_2^*$. It remains to show that the object $\mathcal{F}_{\text{PTIC}}(X)$ is unique up to isomorphism. For this reason consider another object $X' \in \text{Ob}_{\text{SubAGraphs}_{\text{PNTG}}}$ and morphisms $e' : \mathcal{F}_{\text{PTIC}}(A) \rightarrow X'$, $m' : X' \rightarrow \mathcal{F}_{\text{PTIC}}(B)$ in $\text{Mor}_{\text{SubAGraphs}_{\text{PNTG}}}$ such that it holds that $e' \in \mathcal{E}_2^*$, $m' \in \mathcal{M}_2^*$ and $\mathcal{F}_{\text{PTIC}}(f) = m' \circ e'$. We have to show that $\mathcal{F}_{\text{PTIC}}(X) \cong X'$. According to the definition of the category $\text{SubAGraphs}_{\text{PNTG}}$, we know that X' , e' and m' are $\mathcal{F}_{\text{PTIC}}$ -images. This means that for some object $X'' \in \text{Ob}_{\mathbf{PTINet}_{|\mathcal{M}_1}}$ and morphisms $e'' : A \rightarrow X''$, $m'' : X'' \rightarrow B$ in $\text{Mor}_{\mathbf{PTINet}_{|\mathcal{M}_1}}$ it holds that $X' = \mathcal{F}_{\text{PTIC}}(X'')$, $e' = \mathcal{F}_{\text{PTIC}}(e'')$ and $m' = \mathcal{F}_{\text{PTIC}}(m'')$. Furthermore, by Lemma 60 we know that the restricted functor $\mathcal{F}_{\text{PTIC}}$ creates \mathcal{M} -morphisms implying that $m'' \in \mathcal{M}_1$. To show that $e'' \in \mathcal{E}_1$, let us first assume that $e'' \notin \mathcal{E}_1$. This implies directly a contradiction, namely that also $(\mathcal{F}_{\text{PTIC}}(e'') = e') \notin \mathcal{E}_2^*$ since we have the following: Let $e'' = (e''_p, e''_t, e''_l)$. Then $e'' \notin \mathcal{E}_1$ means that either e''_p or e''_t or e''_l is not surjective. Let us first assume that e''_p is not surjective, which means that there is $y \in P_{X''}$ such that $\forall x \in P_A. e''_p(x) \neq y$. It remains to show that also $\mathcal{F}_{\text{PTIC}}(e'')$ is not surjective. For this reason it suffices to show that $\mathcal{F}_{\text{PTIC}}(e'')_{V_G}$ is not surjective, which follows from the property that $\forall x \in V_G^{\Lambda'}. \mathcal{F}_{\text{PTIC}}(e'')_{V_G}(x) \neq y$ since y is also in $V_G^{\overline{X'}} = P_{X''} \uplus T_{X''} \uplus I_{X''}$. Fix $x \in (V_G^{\Lambda'} = P_A \uplus T_A \uplus I_A)$. Then we have the following:

$$\mathcal{F}_{\text{PTIC}}(e'')_{V_G}(x) = (e''_p \uplus e''_t \uplus e''_l)(x)$$

Let $x \in P_A$ then it holds:

$$(e''_p \uplus e''_t \uplus e''_l)(x) = e''_p(x) \neq y \text{ by assumption for } y \in P_{X''}$$

Let $x \in T_A$ then it holds:

$$(e''_p \uplus e''_t \uplus e''_l)(x) = e''_t(x) \neq y \text{ for } y \in P_{X''}$$

Let $x \in I_A$ then it holds:

$$(e''_p \uplus e''_t \uplus e''_l)(x) = e''_l(x) \neq y \text{ for } y \in P_{X''}$$

Hence, we get that $e'' \in \mathcal{E}_1$. Finally, we can show that $f = m'' \circ e''$ as follows:

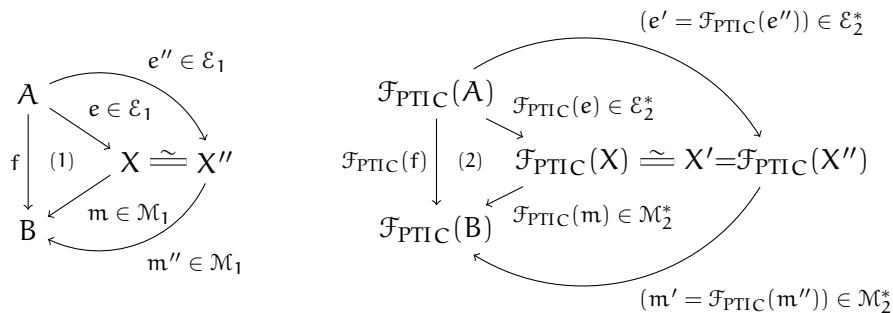
$$\mathcal{F}_{\text{PTIC}}(f) = e' \circ m'$$

$$\Rightarrow \mathcal{F}_{\text{PTIC}}(f) = \mathcal{F}_{\text{PTIC}}(e'') \circ \mathcal{F}_{\text{PTIC}}(m'')$$

$$\Rightarrow \mathcal{F}_{\text{PTIC}}(f) = \mathcal{F}_{\text{PTIC}}(e'' \circ m'')$$

$$\stackrel{\text{Def. } \mathcal{F}_{\text{PTIC}}}{\Rightarrow} \mathcal{F}_{\text{PTIC}}(f) = \mathcal{F}_{\text{PTIC}}(e'' \circ m'')$$

$$\stackrel{\text{Lem. 10} + \text{Rem. 10}}{\Rightarrow} f = e'' \circ m''$$



Thus, we get that $A \xrightarrow{e''} X'' \xrightarrow{m''} B$ is an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization of the morphism $f : A \rightarrow B$ in $\mathbf{PTINet}|_{\mathcal{M}_1}$ with $e'' \in \mathcal{E}_1$, $m'' \in \mathcal{M}_1$ and $f = m'' \circ e''$ according to Definition 10. But since $A \xrightarrow{e} X \xrightarrow{m} B$ is also an $\mathcal{E}_1 - \mathcal{M}_1$ -factorization of the morphism $f : A \rightarrow B$ by assumption, we have that $X'' \cong X$, which implies by application of the restricted functor $\mathcal{F}_{\text{PTIC}}$ that also $X' \cong \mathcal{F}_{\text{PTIC}}(X)$. Thus, we get that $\mathcal{F}_{\text{PTIC}}(A) \xrightarrow{\mathcal{F}_{\text{PTIC}}(e)} \mathcal{F}_{\text{PTIC}}(X) \xrightarrow{\mathcal{F}_{\text{PTIC}}(m)} \mathcal{F}_{\text{PTIC}}(B)$ is an $\mathcal{E}_2^* - \mathcal{M}_2^*$ -factorization of the morphism $\mathcal{F}_{\text{PTIC}}(f)$ in $\mathbf{SubAGraphs}_{\text{PNTG}}$ and hence the restricted functor $\mathcal{F}_{\text{PTIC}}$ preserves $\mathcal{E} - \mathcal{M}$ -factorizations.

Moreover, since I_{PTI} preserves coproducts of \mathcal{M} -morphisms, monomorphisms and epimorphisms according to the proofs of the previous properties, we get using Lemma 16³ and Remark 13 that I_{PTI} translates $\mathcal{E}_2' - \mathcal{M}_2^*$ pair factorization in $\mathbf{SubAGraphs}_{\text{PNTG}}$ into the corresponding $\mathcal{E}_2' - \mathcal{M}_2$ pair factorization in $\mathbf{AGraphs}_{\text{PNTG}}$. Thus, by Definition 51 adapted according to Remark 13, we have that I_{PTI} is compatible with pair factorization implying that I_{PTI} preserves \mathcal{E}' -instances.

9. I_{PTI} creates \mathcal{E}' -instances:

According to Definition 61, we have to show the following:

$$\begin{aligned} & \forall (a'', b'') \in I_{\text{PTI}}(\mathcal{E}_2''). \text{ (2) commutes} \wedge b'' \in \mathcal{M}_2 \Rightarrow \\ & \exists (a', b') \in \mathcal{E}_2''. a'' = I_{\text{PTI}}(a') \wedge b'' = I_{\text{PTI}}(b') \wedge \text{(1) commutes} \wedge b' \in \mathcal{M}_2^* \end{aligned}$$

Let (a'', b'') be a pair of jointly surjective morphisms in $I_{\text{PTI}}(\mathcal{E}_2'')$ such that (2) commutes and $b'' \in \mathcal{M}_2$. Since $I_{\text{PTI}}(\mathcal{E}_2'') = \mathcal{E}_2''$, we have that there is $(a', b') \in \mathcal{E}_2''$, namely the pair (a'', b'') , with $I_{\text{PTI}}(a'') = a''$, $I_{\text{PTI}}(b'') = b''$ and the diagram (1) obviously commutes. It remains to show that $(b' = b'') \in \mathcal{M}_2^*$. Since I_{PTI} creates \mathcal{M} -morphisms as already shown above, we have that for $b'' \in \mathcal{M}_2$ there is exactly one morphism $b^* \in \mathcal{M}_2^*$ such that $I_{\text{PTI}}(b^*) = b''$, which implies directly that $b'' \in \mathcal{M}_2^*$.

$$\begin{array}{ccc} P & \xrightarrow{b} & P' \\ \downarrow a & (1) & \downarrow a' \\ C & \xrightarrow{b' \in \mathcal{M}_2^*} & C' \end{array} \qquad \begin{array}{ccc} I_{\text{PTI}}(P) & \xrightarrow{I_{\text{PTI}}(b)} & I_{\text{PTI}}(P') \\ \downarrow I_{\text{PTI}}(a) & (2) & \downarrow a'' = I_{\text{PTI}}(a') \\ I_{\text{PTI}}(C) & \xrightarrow{(b'' = I_{\text{PTI}}(b')) \in \mathcal{M}_2} & C'' = I_{\text{PTI}}(C') \end{array}$$

□

³ Note that I_{PTI} preserves coproducts because it preserves coproducts of \mathcal{M} -morphisms and $\mathbf{SubAGraphs}_{\text{PNTG}}$ contains only \mathcal{M} -morphisms by definition.

LIST OF DEFINITIONS

1	\mathcal{M} -Adhesive Category	19
2	\mathcal{M} -Adhesive Transformation System	19
3	Direct Transformation	19
4	Initial Pushout	20
5	Gluing Condition in \mathcal{M} -Adhesive Transformation Systems	20
6	Termination of \mathcal{M} -Adhesive Transformation Systems	22
7	Confluence of \mathcal{M} -Adhesive Transformation Systems	23
8	Local Confluence of \mathcal{M} -Adhesive Transformation Systems	24
9	Parallel and Sequential Independence of Direct Transformations	25
10	\mathcal{E} – \mathcal{M} -Factorization	26
11	\mathcal{E}' -Instance, \mathcal{E}' – \mathcal{M}' Pair Factorization	26
12	Critical Pair	27
13	Derived Span	28
14	Strict Confluence of Critical Pairs	29
15	Nested Application Condition	31
16	Satisfaction of Nested Application Conditions	32
17	Rule with Nested Application Conditions	33
18	Applicability of Rules with Nested Application Conditions	33
19	Direct Transformation with Nested Application Conditions	33
20	AC-Disregarding (Direct) Transformation	33
21	Plain Derived Rule	34
22	Derived Nested Application Condition	35
23	Derived Rule	35
24	Parallel and Sequential Independence of Direct Transformations with Nested Application Conditions	37
25	Weak Critical Pair of Transformations with Nested Application Conditions	38
26	Critical Pair of Transformations with Nested Application Conditions	39
27	Compatibility of Nested Application Conditions	40
28	AC-Compatibility of AC-Disregarding Transformations	40
29	Strict AC-Confluence of Critical Pairs	40
30	E-Graphs and E-Graph Morphisms	42
31	Attributed Graphs and Attributed Graph Morphisms	43
32	Attributed Type Graph ATG	43
33	Typed Attributed Graphs over Type Graph ATG and Typed Attributed Graph Morphisms	44
34	Hypergraphs and Hypergraph Morphisms	47
35	Category HyperGraphs	49
36	Construction of Hypergraph Pushouts	50
37	Construction of Hypergraph Pullbacks	52
38	PTI Nets and PTI Net Morphisms	62
39	Firing Behavior of PTI Nets	64

40	Category PTINet	65
41	\mathcal{M} -Functor	82
42	Properties of \mathcal{M} -Functors	82
43	Translation and Creation of Rule Applicability and Direct Transformations with General Match Morphisms	83
44	Restricted \mathcal{M} -Functor	86
45	Properties of Restricted \mathcal{M} -Functors	86
46	Translation and Creation of Rule Applicability and Direct Transformations with \mathcal{M} -Match Morphisms	87
47	Translated Nested Application Condition	89
48	Isomorphism Closure	93
49	R-Simulation, R-Similarity	94
50	R-Bisimulation, R-Bisimilarity	94
51	\mathcal{F} is Compatible with Pair Factorization	102
52	\mathcal{F} -Reachable Critical Pair	103
53	Translated Plain Derived Rule	107
54	Translated Derived Nested Application Condition	108
55	Translated Derived Rule	108
56	\mathcal{F} -Reachable Weak Critical Pair of Transformations with Nested Application Conditions	109
57	\mathcal{F} -Reachable Critical Pair of Transformations with Nested Application Conditions	110
58	$\mathcal{F}(\text{AC})$ -Compatibility of AC-Disregarding Transformations	111
59	$\text{AC}(\mathcal{F})$ -Compatibility of AC-Disregarding Transformations	111
60	Strict $\text{AC}(\mathcal{F})$ -Confluence of Critical Pairs	112
61	Preservation and Creation of \mathcal{E}' -Instances	113
62	\mathcal{F} -Termination of a Translated \mathcal{M} -Adhesive Transformation System	118
63	Functor \mathcal{F}_{HG}	124
64	Restricted Functor \mathcal{F}_{PTI}	174
65	Functor Transformation	213
66	Functor Equivalence	213
67	Category Equivalence	214
68	Category Isomorphism	214
69	Collection of Properties for \mathcal{M} -Functors	214
70	Collection of Properties for Restricted \mathcal{M} -Functors	217
71	Jointly Epimorphic Morphisms	269
72	$\mathcal{M} - \mathcal{M}'$ Pushout-Pullback Decomposition Property	272
73	Extension Diagram	272
74	Gluing Condition in AGraphs _{ATG}	273

LIST OF THEOREMS

1	Translation and Creation of Rule Applicability and (Direct) Transformations for Rules without Nested Application Conditions	84
2	Translation and Creation of Rule Applicability and (Direct) Transformations with \mathcal{M} -Match Morphisms for Rules without Nested Application Conditions	87
3	Translation and Creation of Rule Applicability and (Direct) Transformations for Rules with Nested Application Conditions	90
4	\mathcal{F} -Bisimilarity of \mathcal{M} -Adhesive Transformation Systems	95
5	\mathcal{F} -Transfer of R-Bisimilarity	97
6	Translation and Creation of Local Confluence for Transformations without Nested Application Conditions	99
7	Translation and Creation of Parallel and Sequential Independence of Transformations without Nested Application Conditions	100
8	Creation of Local Confluence Based on \mathcal{F} -Reachable Critical Pairs for Rules without Nested Application Conditions	104
9	Translation and Creation of Local Confluence for Transformations with Nested Application Conditions	105
10	Translation and Creation of Parallel and Sequential Independence of Transformations with Nested Application Conditions	106
11	Creation of Local Confluence Based on \mathcal{F} -Reachable Critical Pairs for Rules with Nested Application Conditions	116
12	\mathcal{F} -Transfer of Termination	118
13	\mathcal{F} -Transfer of Confluence and Functional Behavior	119
14	Translation of Hypergraph Transformations into Typed Attributed Graph Transformations	132
15	Creation of Hypergraph Transformations from Typed Attributed Graph Transformations	142
16	\mathcal{F}_{HG} -Bisimilarity and \mathcal{F}_{HG} -Transfer of R-Bisimilarity	142
17	Local Confluence of Hypergraph Transformation Systems without Nested Application Conditions	151
18	Local Confluence of Hypergraph Transformation Systems with Nested Application Conditions	153
19	\mathcal{F}_{HG} -Transfer of Termination	168
20	\mathcal{F}_{HG} -Transfer of Confluence and Functional Behavior	168
21	Translation of PTI Net Transformations into Typed Attributed Graph Transformations	180
22	Creation of PTI Net Transformations from Typed Attributed Graph Transformations	191
23	\mathcal{F}_{PTI} -Bisimilarity and \mathcal{F}_{PTI} -Transfer of R-Bisimilarity	191
24	Local Confluence of PTI Net Transformation Systems without Nested Application Conditions	195

25	Local Confluence of PTI Net Transformation Systems with Nested Application Conditions	197
26	\mathcal{F}_{PTI} -Transfer of Termination	209
27	\mathcal{F}_{PTI} -Transfer of Confluence and Functional Behavior	209
28	$\mathcal{F} = \text{I} \circ \mathcal{F}_{\text{C}}$ Satisfies Required Properties	216
29	$\mathcal{F}_{\text{R}} = \text{I}_{\text{R}} \circ \mathcal{F}_{\text{RC}}$ Satisfies Required Properties	218
30	\mathcal{F}_{HG} Satisfies Required Properties	222
31	\mathcal{F}_{PTI} Satisfies Required Properties	223

LIST OF LEMMAS

1	$\mathcal{E}' - \mathcal{M}$ Pair Factorization Based on $\mathcal{E} - \mathcal{M}$ -Factorization	27
2	Characterization of Hypergraph Morphisms	49
3	Pushout in HyperGraphs	50
4	Pullback in HyperGraphs	52
5	Boundary Object in $(\mathbf{HyperGraphs}, \mathcal{M})$	53
6	Context Object in $(\mathbf{HyperGraphs}, \mathcal{M})$	54
7	Initial Pushout in $(\mathbf{HyperGraphs}, \mathcal{M})$	54
8	\mathcal{F} Creates Identities and Isomorphisms	83
9	\mathcal{F} is Injective on Objects	83
10	\mathcal{F} is Injective on Morphisms	83
11	Satisfaction of Nested Application Conditions	90
12	R-Simulations are Closed under Isomorphism Closure	94
13	R-Bisimulations are Closed under Isomorphism Closure	95
14	\mathcal{F} -Image Restricted \mathcal{F} -Transfer of R-Bisimilarity	97
15	R-Bisimulation is Preserved under \mathcal{F} -Image Restriction	97
16	\mathcal{F} Preserves $\mathcal{E}' - \mathcal{M}$ Pair Factorization	102
17	\mathcal{F} -Reachable Critical Pairs	103
18	Completeness of \mathcal{F} -Reachable Critical Pairs	103
19	Creation of Compatibility of Nested Application Conditions	110
20	$\mathcal{F}(\text{AC})$ -Compatibility Implies AC-Compatibility	111
21	Preservation and Creation of Plain Strict Confluence	112
22	\mathcal{F} is Compatible with Shift -Transformation	114
23	\mathcal{F} is Compatible with L -Transformation	114
24	$\text{AC}(\mathcal{F})$ -Compatibility Implies $\mathcal{F}(\text{AC})$ -Compatibility	115
25	$\text{AC}(\mathcal{F})$ -Compatibility Implies AC-Compatibility	115
26	Preservation of (Weak) Critical Pairs	116
27	Well-Definedness of Hypergraph Morphism Translation	126
28	\mathcal{F}_{HG} Preserves Pushouts along Injective Morphisms	129
29	Uniquely Determined \mathcal{F}_{HG} -Images	137
30	\mathcal{F}_{HG} Creates Morphisms	137
31	\mathcal{F}_{HG} Creates Injective Morphisms	137
32	Application of \mathcal{F}_{HG} to a Hypergraph Boundary Object B over a Morphism $f : L \rightarrow G$	138
33	Boundary Object in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$ over a Morphism $f' : L' \rightarrow G'$. .	139
34	\mathcal{F}_{HG} Preserves Initial Pushouts	140
35	$\mathcal{E} - \mathcal{M}$ -Factorization in HyperGraphs	150
36	\mathcal{F}_{HG} Preserves Coproducts	150
37	\mathcal{F}_{HG} Preserves Surjective Morphisms	150
38	\mathcal{F}_{HG} is Compatible with Pair Factorization	151
39	\mathcal{F}_{HG} Translates Jointly Surjective Morphisms	152
40	\mathcal{F}_{HG} Creates Jointly Surjective Morphisms	152

41	\mathcal{F}_{HG} Preserves Pullbacks of Injective Morphisms	153
42	Well-Definedness of PTI Net Morphism Translation	177
43	\mathcal{F}_{PTI} Preserves Pushouts of Injective Morphisms	179
44	Uniquely Determined \mathcal{F}_{PTI} -Images	185
45	\mathcal{F}_{PTI} Creates Injective Morphisms	185
46	Application of \mathcal{F}_{PTI} to a PTI Net Boundary Object over an Injective Morphism $f : L \rightarrow G$	186
47	Boundary Object in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ over an Injective Morphism $f' : L' \rightarrow G'$	186
48	Boundary Object in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ over a Morphism $f' = \mathcal{F}_{\text{PTI}}(f) :$ $\mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ for an Injective PTI Net Morphism $f : L \rightarrow G$	187
49	Initial Pushout in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$ over a Morphism $f' = \mathcal{F}_{\text{PTI}}(f) :$ $\mathcal{F}_{\text{PTI}}(L) \rightarrow \mathcal{F}_{\text{PTI}}(G)$ for an Injective PTI Net Morphism $f : L \rightarrow G$	190
50	\mathcal{F}_{PTI} Preserves Initial Pushouts over Injective Morphisms	190
51	$\mathcal{E} - \mathcal{M}$ -Factorization in PTINet	194
52	\mathcal{F}_{PTI} Preserves Coproducts	194
53	\mathcal{F}_{PTI} Preserves Surjective Morphisms	194
54	\mathcal{F}_{PTI} is Compatible with Pair Factorization	195
55	\mathcal{F}_{PTI} Translates Jointly Surjective Morphisms	196
56	\mathcal{F}_{PTI} Creates Jointly Surjective Morphisms	196
57	\mathcal{F}_{PTI} Preserves Pullbacks of Injective Morphisms	197
58	\mathcal{F}_{C} Satisfies Required Properties	215
59	$I \circ \mathcal{F}_{\text{C}}$ Satisfies Required Properties	215
60	\mathcal{F}_{RC} Satisfies Required Properties	218
61	$I_{\text{R}} \circ \mathcal{F}_{\text{RC}}$ Satisfies Required Properties	218
62	\mathcal{F}_{HGC} is a Category Equivalence	221
63	I_{HG} Satisfies Required Properties	222
64	$\mathcal{F}_{\text{PTIC}}$ is a Category Equivalence	223
65	I_{PTI} Satisfies Required Properties	223
66	A Special Case of Pair Factorization	271

LIST OF FACTS

1	Local Church-Rosser Theorem	25
2	Completeness of Critical Pairs Lemma	28
3	Local Confluence Theorem and Critical Pair Lemma	29
4	Local Church-Rosser Theorem for Transformations with Nested Application Conditions	37
5	Local Confluence Theorem for Transformation Systems with Nested Application Conditions	40
6	Boundary Object in $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$	45
7	Context Object in $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$	45
8	Initial Pushout in $(\mathbf{AGraphs}_{\text{ATG}}, \mathcal{M})$	46
9	Boundary Object over a General Match Morphism in $(\mathbf{PTINet}, \mathcal{M})$	65
10	Context Object in $(\mathbf{PTINet}, \mathcal{M})$	67
11	Initial Pushout over an Injective Match Morphism in $(\mathbf{PTINet}, \mathcal{M})$	67

LIST OF REMARKS

1	$\mathcal{E}' - \mathcal{M}$ Pair Factorization	27
2	Functional Behavior of \mathcal{M} -Adhesive Transformation Systems	30
3	Negative and Positive Application Conditions	32
4	Shift of Nested Application Conditions over Morphisms	34
5	Shift of Nested Application Conditions over Rules	35
6	$(\mathbf{AGraphs}_{\mathbf{ATG}}, \mathcal{M})$ is an \mathcal{M} -adhesive Category	44
7	$(\mathbf{HyperGraphs}, \mathcal{M})$ is an \mathcal{M} -adhesive Category	50
8	Relevant Facts about the Category of PTI Nets	65
9	Boundary Object over an Injective Match Morphism in $(\mathbf{PTINet}, \mathcal{M})$	66
10	Additional Properties Holding for Restricted \mathcal{M} -Functors	88
11	Satisfaction of Translated Nested Application Conditions	89
12	Translation and Creation of Rule Applicability and (Direct) Transformations with \mathcal{M} -Match Morphisms for Rules with Nested Application Conditions	92
13	Compatibility of Restricted \mathcal{M} -Functors with Pair Factorization	102
14	\mathcal{F} -Transfer of Termination for the Case of Transformations with Nested Application Conditions	119
15	\mathcal{F} -Transfer of Confluence and Functional Behavior for the Case of Transformations with Nested Application Conditions	120
16	Functor Decomposition Strategy for Verification of Properties Required by the General Theory of \mathcal{M} -Functors	216
17	Functor Decomposition Strategy for Verification of Properties Required by the General Theory of Restricted \mathcal{M} -Functors	219

LIST OF EXAMPLES

1	Nested Application Condition	31
2	Satisfaction of Nested Application Conditions	32
3	Hypergraphs and Hypergraph Morphisms	48
4	Construction of Hypergraph Pushouts	51
5	Construction of Hypergraph Pullbacks	52
6	Construction of Initial Pushouts in (HyperGraphs, \mathcal{M})	55
7	Mobile Processes	56
8	Strict AC-Confluence of a Critical Pair in the Context of the Mobile Processes Scenario	57
9	PTI Nets and PTI Net Morphisms	63
10	Firing of Transitions in PTI Nets	64
11	Construction of Initial Pushouts over Injective Match Morphisms in (PTINet, \mathcal{M})	67
12	Mobile Dining Philosophers	69
13	Strict AC-Confluence of a Critical Pair in the Context of Mobile Dining Philosophers Scenario	72
14	Hypergraph Transformations Translated by \mathcal{F}_{HG}	132
15	Coffee-Tea Machines	143
16	Strict $\text{AC}(\mathcal{F}_{\text{HG}})$ -Confluence of an \mathcal{F}_{HG} -Reachable Critical Pair in the Context of Mobile Processes Scenario	154
17	PTI Net Transformations Translated by \mathcal{F}_{PTI}	181
18	Strict $\text{AC}(\mathcal{F}_{\text{PTI}})$ -Confluence of an \mathcal{F}_{PTI} -Reachable Critical Pair in the Context of Mobile Dining Philosophers Scenario	198

LIST OF FIGURES

1	Abstract perspective on the research problem and the proposed solution . . .	8
2	Example view of AGG	17
3	Nested application condition ac_P	32
4	Diagram for satisfaction of a nested application condition $ac_P = \exists(a, ac_C)$	32
5	Satisfaction of the nested application condition ac_P	33
6	Rule and transformation with a nested application condition ac_L	33
7	Data type signature $\Sigma\text{-nat}$ and algebra NAT	42
8	Pushout in HyperGraphs	51
9	Pullback in HyperGraphs	53
10	Initial Pushout in $(\mathbf{HyperGraphs}, \mathcal{M})$	55
11	Hypergraph defining a network with distributed processes	56
12	Hypergraph transformation rules without nested application conditions modeling the behavior of the Mobile Processes scenario	57
13	Hypergraph transformation rules with nested application conditions modeling the behavior of the extended Mobile Processes scenario	57
14	Weak critical pair of rules $runP$ and $enterServer$	58
15	Extension application condition $ac_K = ac_{K_1} \wedge ac_{K_2}$ induced by the weak critical pair of rules $runP$ and $enterServer$	58
16	Critical pair of rules $runP$ and $enterServer$	59
17	Embedding morphism $m : K \rightarrow G$ satisfies ac_K induced by the weak critical pair of rules $runP$ and $enterServer$	59
18	Plain strict confluence of the critical pair of rules $runP$ and $enterServer$	60
19	Pushout, pullback, and initial pushout in $(\mathbf{PTINet}, \mathcal{M})$	68
20	PTI net modeling five philosophers sitting at a table	70
21	PTI net transformation rules without nested application conditions modeling the behavior of the Mobile Dining Philosophers scenario	70
22	PTI net transformation rules with PACs modeling the behavior of the extended Mobile Dining Philosophers scenario	71
23	Weak critical pair of rules $LeaveTable$ and $LeaveTable$	74
24	Extension application condition $ac_K = ac_{K_1} \wedge ac_{K_2}$ induced by the weak critical pair of rules $LeaveTable$ and $LeaveTable$	75
25	Critical pair of rules $LeaveTable$ and $LeaveTable$	76
26	Embedding morphism $m : K \rightarrow G$ satisfies ac_K induced by the weak critical pair of rules $LeaveTable$ and $LeaveTable$	77
27	Plain strict confluence of the critical pair of rules $LeaveTable$ and $LeaveTable$	78
28	Diagram for satisfaction of a nested application condition $\mathcal{F}(ac_P) = \exists(\mathcal{F}(a), \mathcal{F}(ac_C))$	90
29	Attributed type graph $HGTG$	124
30	Hypergraph G and its corresponding typed attributed graph $\mathcal{F}_{HG}(G)$	126
31	Example for the $\mathbf{AGraphs}_{HGTG}$ -morphism $type$ components $type_{V_G}$ and $type_{E_G}$	126

32	Translation of a hypergraph morphism g into the corresponding typed attributed graph morphism $\mathcal{F}_{\text{HG}}(g)$	127
33	Pushout in HyperGraphs with an injective hypergraph morphism $h : G_1 \rightarrow G'_2$	130
34	Translated pushout in AGraphs _{HGTG} with an injective typed attributed graph morphism $\mathcal{F}_{\text{HG}}(h) : \mathcal{F}_{\text{HG}}(G_1) \rightarrow \mathcal{F}_{\text{HG}}(G'_2)$	131
35	Hypergraph transformation step for the application of the rule enterServer	133
36	Satisfaction of the translated NAC of the rule $\mathcal{F}_{\text{HG}}(\text{enterServer})$	133
37	Translated transformation step for the application of the rule $\mathcal{F}_{\text{HG}}(\text{enterServer})$	134
38	Translated transformation rules without nested application conditions of the Mobile Processes system	135
39	Translated transformation rules with nested application conditions of the extended Mobile Processes system	136
40	Preservation of initial pushouts in $(\mathbf{AGraphs}_{\text{HGTG}}, \mathcal{M}_2)$	141
41	Hypergraph transformation rules modeling the behavior of the Coffee-Tea system CT_1	144
42	Hypergraph transformation rules modeling the behavior of the Coffee-Tea system CT_2	144
43	A hypergraph representing the initial state of the Coffee-Tea systems CT_1 and CT_2	145
44	Typed attributed graph transformation rules modeling the behavior of the Coffee-Tea system CT_3	146
45	Typed attributed graph transformation rules modeling the behavior of the Coffee-Tea system CT_4	147
46	Reachability graphs of CT_3 (given to the left) and CT_4 (given to the right) .	148
47	Preservation of binary coproducts by the \mathcal{M} -functor \mathcal{F}_{HG}	151
48	Workflow for local confluence analysis of hypergraph transformation systems and hypergraph grammars without or with nested application conditions	156
49	Computed critical pairs for the Mobile Processes system without considering any additional invariants	158
50	AGG type graph extended by multiplicities for the Mobile Processes system	158
51	Computed critical pairs for the Mobile Processes system considering the AGG type graph with multiplicities	159
52	A critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{runP}), \mathcal{F}_{\text{HG}}(\text{enterServer}))$ non-reachable from the translated start hypergraph by rule application	159
53	A non- \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{runP}), \mathcal{F}_{\text{HG}}(\text{leaveServer}))$	160
54	An \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{enterServer}), \mathcal{F}_{\text{HG}}(\text{crossC}))$	160
55	An \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{runP}), \mathcal{F}_{\text{HG}}(\text{leaveServer}))$	161
56	A critical pair diagram depicting the \mathcal{F}_{HG} -reachable critical pair for the rule pair $(\mathcal{F}_{\text{HG}}(\text{enterServer}), \mathcal{F}_{\text{HG}}(\text{crossC}))$	162

57	Strictness diagram for the rule pair $(\mathcal{F}_{\text{HG}}(\text{enterServer}), \mathcal{F}_{\text{HG}}(\text{crossC}))$. . .	163
58	Computed critical pairs for the extended Mobile Processes system without considering any additional invariants	164
59	Computed critical pairs for the extended Mobile Processes system considering the AGG type graph with multiplicities	165
60	A critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{crossC}),$ $\mathcal{F}_{\text{HG}}(\text{enterServer}))$ with application conditions non-reachable from the translated start hypergraph by rule application	165
61	A non- \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{leaveServer}), \mathcal{F}_{\text{HG}}(\text{runP}))$ with application conditions	166
62	An \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{leaveServer}), \mathcal{F}_{\text{HG}}(\text{runP}))$ with application conditions	166
63	An \mathcal{F}_{HG} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{HG}}(\text{enterServer}), \mathcal{F}_{\text{HG}}(\text{backC1}))$ with application conditions	167
64	Attributed type graph PNTG	174
65	PTI net L and its corresponding typed attributed graph $\mathcal{F}_{\text{PTI}}(L)$	176
66	Example for the AGraphs _{PNTG} -morphism <i>type</i> components <i>type</i> _{V_G} and <i>type</i> _{E_G}	177
67	Counterexample for translation of general (non-injective) morphisms . . .	177
68	Translated pushout in AGraphs _{PNTG} with injective typed attributed graph morphisms $\mathcal{F}_{\text{PTI}}(a), \mathcal{F}_{\text{PTI}}(b), \mathcal{F}_{\text{PTI}}(c)$, and $\mathcal{F}_{\text{PTI}}(f)$	180
69	PTI net transformation step for the application of the rule JoinTable	181
70	Translated transformation step for the application of the rule $\mathcal{F}_{\text{PTI}}(\text{JoinTable})$	182
71	Translated transformation rules without nested application conditions of the Mobile Dining Philosophers system	183
72	Translated transformation rules with PACs of the extended Mobile Dining Philosophers system	184
73	Preservation of initial pushouts in $(\mathbf{AGraphs}_{\text{PNTG}}, \mathcal{M}_2)$	189
74	Preservation of binary coproducts by the restricted \mathcal{M} -functor \mathcal{F}_{PTI}	195
75	Workflow for local confluence analysis of PTI net transformation systems and PTI net grammars without or with nested application conditions . . .	200
76	AGG type graph for the Mobile Dining Philosophers system extended by multiplicities	202
77	A critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{PTI}}(\text{JoinTable}), \mathcal{F}_{\text{PTI}}(\text{JoinTable}))$ non-reachable from the translated start PTI net by rule application	203
78	An \mathcal{F}_{PTI} -reachable critical pair computed by AGG for the rule pair $(\mathcal{F}_{\text{PTI}}(\text{JoinTable}), \mathcal{F}_{\text{PTI}}(\text{JoinTable}))$	204
79	A critical pair diagram depicting the \mathcal{F}_{PTI} -reachable critical pair for the rule pair $(\mathcal{F}_{\text{PTI}}(\text{JoinTable}), \mathcal{F}_{\text{PTI}}(\text{JoinTable}))$	207
80	Strictness diagram for the rule pair $(\mathcal{F}_{\text{PTI}}(\text{JoinTable}), \mathcal{F}_{\text{PTI}}(\text{JoinTable}))$	208
81	Overview of main results	243
82	Requirements to be verified for a (restricted) functor at hand for the instantiation of the theorems of our abstract framework given in Figure 81 , which allow for the transfer of the semantical properties P_1 – P_6 .	244