Interaction on Human-Centric Communication Platforms: Modelling and Analysis using Algebraic High-Level Nets and Processes

vorgelegt von
Dipl.-Inform.
Karsten Gabriel
aus Berlin

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation.

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Stefan Jähnichen
Gutachter:	Prof. Dr. Hartmut Ehrig
	Prof. Dr. Ina Schieferdecker
	Prof. Dr. Julia Padberg

Tag der wissenschaftlichen Aussprache: 14. Januar 2014

Berlin 2014
D83

Abstract

Human-centric communication platforms, like social networks such as Facebook or Twitter, weblogs, Wiki-systems or Apache Wave, are systems that support humans to communicate, collaborate interactively, and exchange content. Over the last years, modern communication systems have evolved to provide different more interactive ways of communication, where several users have the possibility to concurrently contribute to the communication at the same time. These non-sequential ways of communication lead to new requirements for the modelling and analysis of the interactive communication of users on communication platforms, where it does not suffice to model the communication platform itself, but it is also necessary to be able to model scenarios and histories of interactions on a platform. Up to now there are no adequate techniques for the formal modelling of these kinds of communication platforms and the interactions of users on these platforms that could serve as a basis for their formal analysis.

In this thesis, in order to close this gap, we present a modelling framework for communication platforms and scenarios that is an integration of Petri net, algebraic data type, process and graph transformation techniques. Within this framework, we obtain main conceptual results concerning communication platforms and scenarios of interactions that rely on technical results concerning the modelling and evolution of algebraic high-level nets and -processes. Based on the formal modelling of communication platforms and scenarios in our integrated framework, we obtain the following main analysis results:

Employing general results from the categorical framework of $\mathcal{M}$-adhesive categories, we are able to analyse the independence of different evolutions of communication platforms, and to analyse and synthesise parallel as well as concurrent evolutions of platforms. Based on these main results for the analysis of communication platforms, we develop corresponding technical results that on the conceptual layer allow also the analysis of scenario evolutions, and the investigation of user interactions on communication platforms. Due to our integrated framework, we are also able to analyse the compatibility of scenario and platform evolutions. Further, a comprehensive concept of different levels of abstraction of scenarios allows to investigate a whole set of coinciding scenarios at once, and we show that all concrete realisations of an abstract scenario can be obtained as solutions of a term equation system. The implementation of our algebraic high-level net and process editor (APE) is in progress and will support the modelling and analysis of communication platforms and scenarios using the modelling and analysis results in this thesis.

The practical application of all results concerning the modelling and analysis of communication platforms and scenarios are exemplified in the context of the communication platform Apache Wave which serves as a running example throughout the thesis.

Zusammenfassung

Kommunikationsplattformen, die auf die Bedürfnisse von Menschen ausgerichtet sind, wie die sozialen Netzwerke Facebook und Twitter, Weblogs, Wiki-Systeme oder Apache Wave, sind Systeme, die Menschen dabei unterstützen, zu kommunizieren, interaktiv zusammenzuarbeiten und Inhalte auszutauschen. In den letzten Jahren haben sich die Kommunikationssysteme stetig dahingehend weiter entwickelt, neue interaktivere Methoden der Kommunikation zu unterstützen. Dabei ist es oft möglich, dass mehrere Teilnehmer gleichzeitig etwas zur Kommunikation beitragen. Diese nicht-sequentiellen Arten der Kommunikation von Benutzern auf Kommunikationsplattformen führen zu neuen Anforderungen bei der Modellierung und Analyse der interaktiven Kommunikation zwischen Nutzern der Plattformen. Hierbei reicht es nicht, nur die Kommunikationsplattform selbst zu modellieren, sondern es ist erforderlich auch Szenarien von Interaktionen und die Kommunikationsverläufe auf den Plattformen zu modellieren. Bisher gibt es noch keine geeigneten Techniken für die formale Modellierung dieser Arten von Kommunikationsplattformen und der Benutzerinteraktionen auf den Plattformen, die als Basis für deren formaler Analyse dienen könnten.

Um diese Lücke zu schließen, stellen wir in dieser Dissertation ein Modellierungsframework für Kommunikationsplattformen und Szenarien vor, welches durch die Verknüpfung von Petrinetzen, algebraischen Datentypen, Prozessen und Graphtransformation entsteht. Innerhalb dieses Frameworks erhalten wir konzeptuelle Resultate für Kommunikationsplattformen und Interaktionsszenarien, welche auf technischen Resultaten der Modellierung und Evolution von algebraischen High-Level-Netzen und -Prozessen aufbauen. Basierend auf der formalen Modellierung von Kommunikationsplattformen und Szenarien in unserem integrierten Framework erhalten wir die folgenden Analyse-Resultate:

Unter Verwendung allgemeiner Resultate des kategoriellen Frameworks der $\mathcal{M}$-adhäsiven Kategorien sind wir in der Lage, die Unabhängigkeit verschiedenener Evolutionen von Kommunikationsplattformen zu anlysieren, sowie parallele und nebenläufige Plattformevolutionen zu analysieren und zu synthetisieren. Basierend auf diesen Hauptresultaten für die Analyse von Kommunikationsplattformen entwickeln wir entsprechende technische Resultate, die auf konzeptueller Ebene die Analyse der Evolutionen von Szenarien und die Untersuchung von Benutzerinteraktionen auf Kommunikationsplattformen erlauben. Dank unseres integrierten Frameworks sind wir zudem in der Lage, die Kompatibilität von Szenario- und Plattformevolutionen zu analysieren. Desweiteren stellen wir ein umfangreiches Konzept verschiedener Abstraktionsebenen von Szenarien vor, welches es erlaubt, eine Sammlung übereinstimmender Szenarien zugleich zu untersuchen, und wir zeigen, dass es möglich ist, alle konkreten Realisierungen eines abstrakten Szenarios als Lösungen eines Termgleichungssystems zu berechnen. Die Implementierung unseres Algebraischen High-Level-Netz- und Prozess-Editors (APE) befindet sich in Arbeit. Dieser wird die Modellierung und Analyse von Kommunikationsplattformen und Szenarien unter Verwendung der Modellierungs- und Analyseergebnisse in dieser Arbeit unterstützen.

Die praktische Anwendung aller Modellierungs- und Analyseergebnisse für Kommunikationsplattformen und Szenarien wird im Rahmen der Kommunikationsplattform Apache Wave veranschaulicht, welche uns durch die gesamte Arbeit hindurch als Beispiel dient.

Acknowledgements

First of all, I want to thank my supervisor Hartmut Ehrig for all his support. During my work at the TFS (Theoretische Informatik/Formale Spezifikation) research group at Technische Universität Berlin, he gave me the opportunity to learn a lot in the interesting research areas of Petri nets, graph transformation and category theory. I had the freedom to explore and focus on my own fields of interest, while at the same time he made sure that I do not get too far off the path towards my overall goals. I am deeply grateful for that.

I also want to thank my supervisor Ina Schieferdecker for agreeing to supervise my thesis, and for giving me valuable insights in other research fields during my employment at the Fraunhofer FOCUS.

Furthermore, I want to thank the referee Julia Padberg for agreeing to examine my thesis.

Moreover, this work was made possible due to the support of the Integrated Graduate Program in Human-Centric Communication (IGP H-C3) by granting me a PhD scholarship for three years, and the DFG-research project "Formal modeling and analysis of flexible processes in mobile ad-hoc networks" ($forMA_lNET$) at the TFS group.

I was never alone with my work, thanks to all the nice fellow workers in the TFS research group. In particular, I want to thank Tony Modica for all the fruitful and inspiring discussions. Further, I want to thank the rest of the TFS research group Claudia Ermel, Frank Hermann, Mascha Maximova, Olga Runge and Hanna Schölzel.

Last but not least, I owe special thanks to my parents, my wife Antje and my son Juri for always supporting me and for keeping me motivated.

Contents

1 Introduction

A starting point for the subject matter of this thesis is the research on human-centric communication platforms and their interactions within the Integrated Graduate Program on Human-Centric Communication (IGP H-C3)[1] at Technische Universität Berlin, including works in the fields of, electrical engineering, computer graphics, human-computer interfaces and theoretical computer science. This work is a contribution to the modelling and analysis of human-centric communication platforms and their interactions.

Human-centric communication platforms are systems that support humans to communicate, collaborate interactively, and exchange content. Prominent examples for this are social networks, like Facebook [Fac13] or Twitter [Twi13], weblogs, Wiki-systems [Wik13b, Wik13a], or communication software and services, like Skype [Sky13] and Apache Wave [Wav13a, Wav13b].

We use the notion of communication platform in analogy to that of a hardware platform: While the type and condition of a hardware platform determines the capabilities of programs that run on that platform, a communication platform determines the capabilities of users that interact on that platform.

Conventional means of communication via computers, like e-mail and chat, are usually unidirectional and sequential. This means that at a specific moment in time, only one single person is contributing something to the communication. Another person can receive that contribution and in turn contribute something else at a later point. The result of this sequential communication is a document that is incrementally growing over the time.

This way of communication is not really natural for humans, and therefore, modern communication systems provide different more interactive ways of communication, where several users have the possibility to concurrently contribute to the communication at the same time. Considering this non-sequential way to communicate, the communication data does not grow in a linear way, but instead it is interactively changing, and the result of the communication may not even contain all previous contributions. Accordingly, these non-sequential ways of communication lead to new requirements for the modelling and analysis of the interactive communication of users on communication platforms, where it does not suffice to model the communication platform itself, but it is also necessary to be able to model scenarios and histories of interactions on a platform.

Formal techniques for the modelling of systems have been clearly proved to be successful in many different fields of computer science, including also a variety of particular aspects of communicating systems. There are unquestionable benefits of the formal modelling, since they are a necessary basis for well-grounded static and dynamic analysis, validation, simulation and testing of systems – which are of increasing importance, due to the ever-growing complexity of systems and corresponding problems in all areas of computer science. However, considering communication platforms in their entirety, there is not really an integrated for-

[1] IGP H-C3 website – www.h-c3.org/IGP/index_en.html

mal approach that covers the majority of their important aspects and characteristics, except for e. g. [Mod12] that introduces an integrated framework of algebraic higher-order nets with individual tokens (AHOI nets) for the modelling, simulation and validation of communication platforms like Skype [Sky13].

1.1 Aim of the Thesis

The main goal of this thesis is the development of a framework of formal techniques for the modelling and analysis of interactions on human-centric communication platforms. The focus on *human-centric* communication platforms means that we concentrate on the communication between humans rather than on the communication between machines. This has important consequences on the requirements for suitable modelling and analysis techniques. Whereas the communication between machines, programs or devices usually has a functional behaviour that relies on specific deterministic algorithms and protocols, the outcome of the communication between humans is always to some extent uncertain and non-deterministic, due to the human participant's free will. This has to be taken into account, when modelling human-centric communication platforms. A human-centric communication platform provides means for the communication and interaction between users without predetermining when or how they are used exactly. Nonetheless, it is an aim of this thesis to provide means for the modelling of abstract patterns of user behaviour.

There are formal modelling techniques, like Petri nets [Pet62, Roz87, Rei85, MM90], that allow the modelling of systems with an essentially non-deterministic operational behaviour. Moreover, modelling techniques for processes allow the modelling of scenarios with concurrent behaviour that is crucial for a suitable modelling of interaction scenarios. Apart from their dynamic behaviour, these modelling techniques usually have a static structure. Considering the fact that modern communication platforms are permanently advancing, it is important to model also evolutions of communication platforms and corresponding scenarios. So, it is a goal of this thesis, to combine suitable modelling techniques for communication platforms and scenarios with formal techniques for the reconfiguration of systems, like rule-based graph transformation [Ehr79, Roz97, EHKPP91a, EEPT06b].

Considering the vast number of possible scenarios of different interactions in a communication platform, it is important for the analysis of scenarios to provide means to unify scenarios that share a similar form. Therefore, an aim of this thesis is the development of a mechanism for different levels of abstraction for scenarios, where on a lower level it is possible to consider the single concrete scenarios, and on a higher level, we have abstract patterns that represent a set of different scenarios with a common structure. It is intended to also provide ways to switch between these levels of abstraction. For a concrete scenario it should be possible to obtain an abstraction of that scenario, and vice versa, for each abstraction it should be possible to compute the set of all its concrete realisations.

Regarding the growing complexity of modern communication platforms, a further goal of this thesis are techniques to provide different views on communication platforms. This shall allow to concentrate only on those parts of a communication platform that are of a particular interest, while masking out all other parts. Accordingly, considering a specific view on a communication platform, we also want to provide a way to obtain a corresponding view on scenarios of that platform. Moreover, it is an aim to develop techniques to combine different views on communication platforms and scenarios to a larger view that encompasses all aspects of the smaller views. Views on communication platforms are particularly interesting in the context of cross-platform communication, where users on different platforms might communicate with each other. In this case, we are interested in a view that sees the different

single platforms as one integrated communication platform, and scenarios of cross-platform interaction are scenarios of that platform.

An integrated framework that is based on the combination of existing well-researched techniques may already provide several techniques for the analysis of models and their evolutions. We intend to apply existing analysis techniques to the models of communication platforms and scenarios, and otherwise extend the techniques, where this is possible, or develop new techniques. Considering the evolutions of communication platforms and scenarios, respectively, we are primarily interested in techniques to answer the question whether different evolutions are in conflict with each other. Two evolutions can be in conflict with each other, if they can only be performed exclusively, or maybe if they can only be performed in a specific order. Otherwise, we can say that the evolutions are independent. In the case of the evolution of communication platforms, independent evolutions correspond to compatible updates of a platform. Non-independent evolutions, on the other hand, may indicate that the communication platform has to be updated with great care, because it is possible that one evolution is not applicable to a later state of the platform. In the case of the evolution of scenarios, independent evolutions may correspond to compatible interactions, and dependent evolutions may indicate that there are conflicts between interactions. Considering the possibility of non-atomic evolutions that correspond to different changes at once, we are also interested in ways to analyse complex evolutions into smaller ones, in order to allow the analysis and refinement of these smaller evolutions. Vice versa, we are also interested in the synthesis of complex evolutions from compatible smaller ones, in order to analyse also the independence of more complex updates or interactions, respectively.

Moreover, considering existing scenarios of evolved communication platforms, it is an aim of this thesis to analyse the compatibility of scenarios and platform evolutions. We are interested in techniques to analyse whether a scenario of a communication platform is still consistent with that platform if the corresponding platform has evolved. On the other hand, we are interested in ways to evolve scenarios consistently to the evolution of the corresponding platform, in order to obtain again a valid scenario, where all modifications to the platform are also reflected in corresponding modifications on interactions in the scenario.

All techniques for the modelling and analysis of communication platforms have to be applied to an actual existing communication platform, in order to verify their practical benefits. As a case study of a communication platform, we choose the open-source project Apache Wave [Wav13a]. It consists of many properties of a typical modern communication platform that we are interested in, like non-sequentiality and the possibility for cross-platform communication.

Finally, we plan to provide a tool support for the modelling and analysis of communication platforms and scenarios, using the techniques that are developed and investigated in this thesis.

1.2 Main Results

In this section, we give a brief overview on the results of this thesis. For a comprehensive overview on the modelling results of this thesis, we refer to Section 8.1, and the analysis results are summarised in Section 8.2.

Modelling and Evolution of Communication Platforms For the modelling of communication platforms, we use the well-researched modelling technique of algebraic high-level (AHL-)nets [EPR92, PER95]. This is an integration of Petri nets and data types in the sense of algebraic specification [EM85]. It allows the modelling of the structure of a communication platform together with an operational behaviour.

The structural evolution of communication platforms is modelled by rule-based transformation of AHL-nets in the sense of graph transformation [Ehr79, Roz97]. The resulting transformation system fits into the abstract categorical framework of $\mathcal{M}$-adhesive categories [EGH10], allowing to employ a large set of already available analysis techniques [EEPT06b, EGH10] for the analysis of communication platforms. Moreover, for the evolution of the data type part of a communication platform, we introduce the concept of data-images of AHL-nets.

Modelling and Evolution of Scenarios With regard to the targeted levels of abstraction for scenarios, mentioned in the previous section, we distinguish between the modelling of abstract, concrete and semi-concrete scenarios. Abstract scenarios specify a logical and causal relation of interactions, but they do not specify any concrete data values that are used during the interaction. For the modelling of abstract scenarios, we use AHL-processes [EHP+02, Ehr05] that conform to an AHL-net model of the corresponding communication platform. Furthermore, concrete scenarios capture a logical and causal relation of interactions together with all the concrete data values that are used during the process of interaction. This is modelled using instantiated AHL-processes [Ehr05, Gab10]. Semi-concrete scenarios lie between abstract and concrete scenarios, in the sense that data values are abstractly specified. For this purpose, we introduce the new concept of abstract instantiations, and semi-concrete scenarios are modelled as abstractly instantiated AHL-processes.

For the modelling of the evolution of scenarios, we extend the concepts of rule-based transformation and data-images of AHL-nets to corresponding concepts that allow also the structural and data evolution of AHL-processes and instantiations.

Modelling of User Behaviour Abstract patterns for the behaviour of users can be modelled as sequential productions for instantiated AHL-processes. Histories and current states of a communication can be modelled as special cases of scenarios. Accordingly, the application of the abstract behavioural patterns, that corresponds to the progression of the history and an adaptation of the current state of a communication, can be performed as a special case of rule-based transformation of instantiated AHL-processes.

Levels of Abstraction Our approach for the modelling of abstract, semi-concrete and concrete scenarios establishes relations of abstraction and concretisation between different levels of abstraction of scenarios. The abstraction of Semi-concrete and concrete scenarios can be obtained by the underlying AHL-process model. Moreover, the concretisation of semi-concrete scenarios is based on the concept of data-images, and concrete realisations of abstract scenarios can be obtained as solutions of a term equation system.

Views on Communication Platforms and Scenarios Based on the modelling of communication platforms using algebraic high-level nets, it is possible to model the view on a platform as an embedding of an AHL-net into a larger context. We introduce techniques for the restriction and amalgamation of AHL-processes along these embeddings, allowing to restrict the view on a scenario to a corresponding view of the associated communication platform or combine views on scenarios according to the composition of platforms.

Compatibility of Platform Evolutions Due to the fact that our modelling approach for communication platforms using AHL-nets fits into the framework of $\mathcal{M}$-adhesive transformation systems, we obtain suitable analysis techniques for the independence of platform evolutions by instantiation of corresponding general results from the abstract

categorical framework [EGH10, EEPT06b]. We also get techniques for the analysis and synthesis of complex evolutions.

Compatibility of Scenario Evolutions The modelling of scenarios using (instantiated) AHL-processes does not satisfy the requirements of an $\mathcal{M}$-adhesive transformation system. However, we extend the results that are used for the analysis of platform evolutions such that they are also applicable to the analysis of scenario evolutions. Since user interactions are modelled as special cases of scenario evolutions, the results can also be used to analyse conflicts and independence of user interactions.

Compatibility between Scenarios and Platform Evolutions The concept of the extension of a scenario can be used to model the case that the scenario of a platform is still in compliance to the result of an evolution of that platform. One result of this thesis are sufficient and necessary conditions for the extension of scenarios along platform evolutions. Another result concerns the evolution of scenarios based on the evolution of platforms, allowing to transfer modifications of a communication platform also to corresponding modifications of scenarios of that platform.

Note that preliminary versions of the results presented in this thesis, we have published already in the following papers: The modelling of AHL-processes with instantiations, relying on a given set of initial markings, in [EHGP09, Gab09] can be seen as a technical starting point for this thesis. In [EG11, Gab11], we have shown that AHL-nets and AHL-processes can be used to model the former communication platform Google Wave that was a predecessor of Apache Wave, and in [EG11] we also presented techniques concerning the independence of AHL-net transformations, and the amalgamation of AHL-processes. Further, in [Gab12a], we introduced the extension of AHL-processes, and in [GE12a] we introduced an approach for the evolution of abstract scenarios based on the evolution of single actions of a platform. This approach was extended to the evolution of abstract scenarios based on the evolution of multiple actions of a platform in [Gab12b]. In [GE12b] we outlined the modelling of communication platforms like Apache Wave, using the framework of AHL-nets and -processes presented in this thesis. The technical details of these works can be found in corresponding technical reports.

1.3 Structure of the Thesis

The thesis is structured as follows:

Chapter 2 (Requirements for Communication Platforms and Scenarios) In this chapter, we discuss the requirements for the modelling and analysis of communication platforms and scenarios. First, in Section 2.1 we specify our understanding and focus on the notion of communication platforms and discuss some prominent examples. As a concrete example of a communication platform we introduce in Section 2.2 our Apache Wave case study which serves as a running example throughout the main part of this thesis. All results concerning the modelling and analysis of communication platforms and scenarios are exemplified in the context of this case study. Further, in Section 2.3 we discuss the requirements that have to be met by a suitable approach for the modelling of communication platforms and scenarios, and in Section 2.4 we give an overview on the several analysis problems that are of interest in the focus of this thesis. In Section 2.5 we review different modelling techniques that might be appropriate for the modelling of communication platforms and scenarios. Finally, in Section 2.6 we give an outlook

on the integrated framework of algebraic high-level nets and processes that is used in the subsequent chapters for the modelling of communication platforms and scenarios.

Chapter 3 (Modelling and Evolution of Communication Platforms) This chapter consists of a presentation of the modelling technique of algebraic high-level (AHL-)nets in Section 3.1, and we show how AHL-nets can be used to model communication platforms. Further, in Section 3.2 we present the rule-based transformation of AHL-nets in the sense of graph transformation that can be used for the structural evolution of communication platforms, followed by the introduction of mechanisms in Section 3.3 that can be used for the data evolution and abstraction of communication platforms. As a last point in this chapter, in Section 3.4 we review the skeleton and flattening functors that yield different low-level place/transition nets from AHL-nets. These functors can be used to extract the structure and flat semantics, respectively, of an AHL-net, and they are of vital importance for the definition of instantiations in the subsequent chapter.

Chapter 4 (Modelling and Evolution of Scenarios) First, in Section 4.1 we present the modelling technique of AHL-processes based on AHL-process nets, and we show how this technique can be used for the modelling of abstract scenarios. Then, in Section 4.2 we extend the rule-based transformation of AHL-nets to support also the rule-based transformation of AHL-process nets and -processes, allowing the structural evolution of abstract scenarios. Moreover, the concept of instantiations and instantiated AHL-processes in Section 4.3 is used to model also concrete and semi-concrete scenarios. In the subsequent Section 4.4, we extend the mechanisms for the data evolution and abstraction of AHL-nets to corresponding mechanisms for AHL-process nets, -processes and instantiations, yielding also a concept of different levels of abstraction for scenarios. The restriction and amalgamation techniques for (instantiated) AHL-processes, presented in Section 4.5, support the restriction and union of views on scenarios based on underlying embeddings and composition, respectively, of communication platforms. Further, in Section 4.6 we present the rule-based transformation of instantiations, and, in combination with the rule-based transformation of AHL-processes, an integrated approach of the rule-based transformation of instantiated AHL-processes. This can be used to model also the structural evolution of concrete and semi-concrete scenarios. The modelling techniques for histories and user behaviour, presented in Section 4.7, are special cases of the techniques for the modelling respectively evolution of concrete scenarios. Finally, in Section 4.8 we introduce the evolution of scenarios based on communication platform evolutions. This includes the extension of scenarios as well as techniques to evolve abstract and concrete scenarios according to evolutions of the underlying communication platform.

Chapter 5 (Analysis of Platforms and Scenarios) In this chapter, we present techniques for the analysis of communication platforms and scenarios. The Local Church-Rosser, Parallelism and Concurrency Theorems for AHL-nets in Section 5.1 are instantiations of corresponding general analysis results in the categorical framework of $\mathcal{M}$-adhesive categories. These techniques can be used for the analysis of independent platform evolutions, and they provide means for the analysis and synthesis of parallel as well as concurrent evolutions of communication platforms. Moreover, in Section 5.2, we introduce corresponding extensions of the Local Church-Rosser, Parallelism and Concurrency Theorems that adequately support the analysis of scenario evolutions. In Section 5.3, we present the construction of a term equation system for the model of

an abstract scenario, and we show that there is a 1-to-1 correspondence between concrete realisations of the abstract scenario and the solutions of the corresponding term equation system.

Chapter 6 (Tool Support) In Section 6.1, we give an overview on the requirements for a suitable tool support that allows the modelling and analysis of communication platforms using the techniques presented in this thesis. This is followed in Section 6.2 by a review of already available modelling tools that might be adequate starting points for the implementation of a tool support for AHL-nets and -processes. In Section 6.3, we present our implementation of an algebraic high-level net and process editor (APE). This includes an overview on the implementation details as well as on the capabilities of the current development status, and an outlook on future extensions.

Chapter 7 (Related Work) In this chapter we give an overview on related work, concerning the different technical and conceptual aspects of this thesis. First, in Section 7.1 we review other contributions to the formal modelling of communication platforms. Then, in Section 7.2 we give an overview on related work on the formal modelling using the modelling technique of Petri nets, as it is the basis for the modelling of communication platforms and scenarios in this thesis. Moreover, considering the fact that structural evolution of platforms and scenarios in this work are modelled using rule-based transformation in the sense of graph transformation, in Section 7.3 we review related work in the field of graph and model transformation. Finally, in Section 7.4 we review other modelling techniques that are related to the techniques used in this thesis.

Chapter 8 (Conclusion) This chapter concludes the thesis with a summary of the results and an outlook on future research. In Section 8.1, we give an overview on the realisations of the modelling requirements for communication platforms and scenarios, as discussed in Section 2.3. This is followed in Section 8.2 by an overview of techniques for solving the analysis problems, discussed in Section 2.4. Section 8.3 is concerned with additional abstract categorical results of this thesis. Finally, in Section 8.4 we discuss further aspects of research that remain for future work.

The Appendix A contains a variety of technical details that are of use for the results in the main part of this thesis, whereas they are not of an immediate conceptual relevance. Moreover, most of the facts and theorems in the main part of this thesis are only accompanied with a proof-idea, and the complete detailed proofs can be found in Appendix B. The bibliography and an index of important and frequently used notions can be found at the end of this thesis.

2 Requirements for Communication Platforms and Scenarios

In this chapter, we discuss the requirements for the modelling and analysis of communication platforms and scenarios. First, in Section 2.1 we specify our understanding and focus on the notion of communication platforms and discuss some prominent examples.

As a concrete example of a communication platform we introduce in Section 2.2 our Apache Wave case study which serves as a running example throughout the main part of this thesis. All results concerning the modelling and analysis of communication platforms and scenarios are exemplified in the context of this case study.

Further, in Section 2.3 we discuss the requirements that have to be met by a suitable approach for the modelling of communication platforms and scenarios, and in Section 2.4 we give an overview on the several analysis problems that are of interest in the focus of this thesis.

In Section 2.5 we review different modelling techniques that might be appropriate for the modelling of communication platforms and scenarios, and in Section 2.6 we give an outlook on the integrated framework of algebraic high-level nets and processes that is used in the subsequent chapters for the modelling of communication platforms and scenarios.

2.1 Communication Platforms and Scenarios

The notion of a platform is usually used to describe the basic architecture that allows a specific set of developments or activities. For instance, a computing platform can denote a specific hardware architecture like a personal computer or a mobile telephone, but it can also denote a software architecture such as an operating system like Windows or Linux, or a software framework like the Java Platform or the .NET framework. A platform determines a specific set of applications or even other platforms that can be launched on the platform.

The notion of a communication platform is also used in different ways, denoting hardware architectures like ethernet or wireless techniques, or software architectures such as client-server, peer-to-peer (P2P) or ad-hoc networks. These usages of the notion are rather technology- or machine-centric as a client, a server or a peer usually denotes a specific hardware component or computer program. Accordingly, such a machine-centric communication platform determines the ability of machines to communicate with each other.

In contrast, a human-centric communication platform should describe the ability of humans to communicate with each other. Of course, the ability of humans to communicate using computer systems depends on the abilities of the used technology. However, with today's omnipresence of the internet, the different kinds of user-interaction are usually not determined by the used technology, but rather by the used services. Regardless whether someone is using a personal computer with an ethernet connection or a mobile phone with UMTS, the user can

access the same (web-based) services as long as there is an internet-browser or an alternative application installed on the device.

Thus, for human-centric communication platforms, we use the term in the sense of [Mod12] where a communication platform is defined as

> a (possibly Internet-based) service with the main or only purpose to allow humans to communicate with each other. Essentially, the users are the only perceivable actors in these systems and thus every reaction of the system is triggered directly by the users' actions. Usually, the users can set preferences regarding their privacy and availability to communication, which are respected by the system.

There are several popular internet-based examples of communication platforms, most of them serving different main purposes such as social interaction and networking, telecommunication, collaboration, sharing of knowledge, planning or organization.

Facebook Facebook [Fac13] is the most well-known example of a social networking website with over one billion active users. The website offers users the ability to connect with their friends on-line, and to communicate via private messages, via text, voice or video chat, or by posting messages on the so-called *timeline* of the own or another profile. Messages on the timeline of a profile can contain text, images or videos. Depending on privacy settings, most of the recent interactions of connected users are aggregated in a news feed.

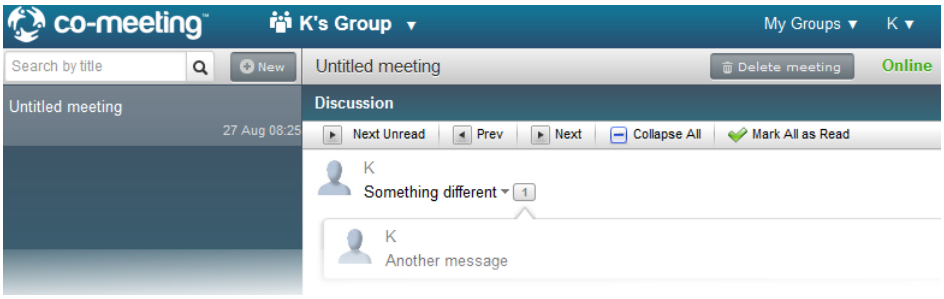
Twitter Another example of a social networking service is Twitter [Twi13] which is a microblogging service. This means that Twitter allows to send text-based messages of up to 140 characters, so-called *tweets*, which by default are publicly visible. Tweets can be restricted to be read only by subscribers (*followers*) of the sender.

Skype Skype [Sky13] is a telephone software application. It offers users the ability to perform voice calls to other users, but also text chatting, video conferences and direct file transfer to other users are possible. The communication software Skype is comprehensively investigated in [Mod12].

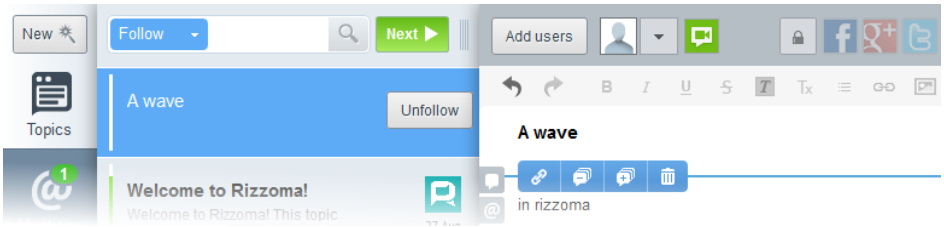
Apache Wave Originally developed by the company Google[2] as Google Wave, the project was handed over in 2010 to the Apache Software Foundation[3] for further development as Apache Wave [Wav13a]. The communication platform allows users to communicate and collaborate via so-called *waves*. A wave is like a document that can be accessed for reading and modification by different users at the same time. Changes to a wave can be recognized by other users almost immediately (near-real-time). For this reason, in Apache Wave for every communication there is a history allowing the user to stepwise replay interactions of the communication. Due to the fact that Apache Wave is open source, there are different independent derivations of the project, like Wave in a box [Wav13b], Google Walkaround [Wal13], Rizzoma [Riz13] or Co-meeting [Cm13]. Apache Wave is considered in more detail in Section 2.2.

Wikis A wiki is a website that allows users to modify its contents. The first wiki was the website WikiWikiWeb[Wik13b], created in 1995, while the most popular example of a wiki is the encyclopaedia Wikipedia[Wik13a]. Wikis are widely used for information

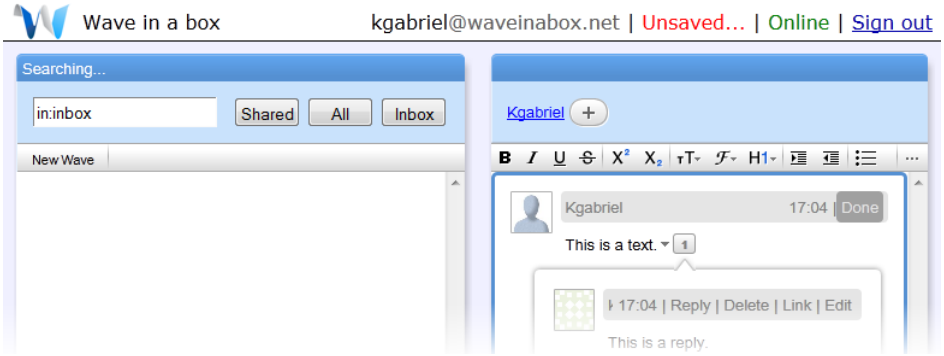
[2] Google website – http://www.google.com
[3] Apache Software Foundation website – http://www.apache.org/



(a) Co-meeting



(b) Rizzoma



(c) Wave in a box

Figure 2.1: Apache Wave implementations

management and note-taking, but they are also used for instance in intranets of organisations for planning the distribution of resources between co-workers. Control over a wiki can be permitted to different levels of access, e. g. only some users may change, add or delete material while the content can be accessed for reading by all users.

Facebook and Skype are proprietary closed-source communication platforms, while there are open-source Twitter, Wave and Wiki projects that can be used by anyone to build an own customized communication platform. One aspect that all these examples have in common is that they all offer a variety of different means of interactions to the users.

Moreover, all of the above examples have evolved over the years and are constantly evolving by adding or changing features. A particularly interesting evolution is the introduction of the Facebook-timeline in 2012 which changed the way, the profile of a user is displayed. Before introduction of the timeline, users could only post messages on their profile which were dated to the present, while the timeline also offers the ability to post messages about events in the past. Since there was a possible impact on the privacy of the users, the users were not forced to use the new timeline, but instead, users could choose if they wanted to switch to the new timeline, or using the old profile instead. Due to the different feature sets in Facebook with or without timeline, this can be seen as two different Facebook platforms

running in parallel, allowing users on both platforms to communicate with each other.

2.2 Case Study: Apache Wave

In this section we introduce our main case study Apache Wave which is a communication platform that was originally developed by the company Google as Google Wave. Google itself has stopped the development of Google Wave, but the development is continued by the Apache Software Foundation as Apache Wave.

One of the most interesting aspects of Apache Wave is the possibility to make changes on previous contributions. Therefore, in contrast to email, text chat or forums, due to possible changes the resulting data of the communication does not necessarily give a comprehensive overview on all interactions of the communication. For this reason, in Google Wave for every communication there was a history allowing the users to replay interactions of the communication step by step. Unfortunately, the playback feature is currently not included in any of the Apache Wave implementations, but it is planned that the feature returns in some of the projects. Taking this feature into account, for the modelling of Apache Wave it is necessary that we do not only model the systems and the communication but also the history of the communication.

We have chosen Apache Wave as running example for this thesis because it includes typical modern features of many other communication systems, such as near-real-time communication. This means that different users can simultaneously edit the same document, and changes of one user can be seen almost immediately by the other users.

In Apache Wave users can communicate and collaborate via so-called waves. A wave is like a document which can contain diverse types of data that can be edited by different invited users. The changes that are made to a wave can be simultaneously recognized by the other participating users. In order to keep track of the changes that have been made, every wave contains also a history of all the actions in that wave.

Apache Wave supports different types of extensions which are divided into gadgets and robots. The extensions are programs that can be used inside of a wave. The difference between gadgets and robots is that gadgets are not able to interact with their environment while robots can be seen as automated users that can independently create, read or change waves, invite users or other robots, and so on. This allows robots for example to do real-time translation or highlighting of texts that are written by different users of a wave. Clearly, it is intended to use different robots for different tasks and it is desired that multiple robots interact without conflicts. This makes the modelling and analysis of Apache Wave very important in order to predict possible conflicts or other undesired behaviour of robots.

Figure 2.2 shows a small example of the structure of a basic Apache Wave platform. In order to keep the example simple, we focus only on actions that operate on one single wave. The platform contains basic features like the creation of new waves, modifications to existing waves, and the invitation of users to a wave.

A wavelet is a part of a wave that contains a user ID, a list of XML documents and a set of users which are invited to modify the wavelet. For simplicity we model in our example only the simple case that every wavelet contains only one single document and the documents contain only plain text. In order to obtain a more realistic model one has to extend the model accordingly.

Figure 2.3 shows an example of a concrete scenario of interactions in a wave. The scenario starts with the creation of a wavelet with id 0 by user Alice. Then, while user Alice insert some text into the previously created wavelet, user Bob creates another wavelet with id 1. Afterwards, Alice invites Bob to participate in the wavelet 0, and Bob changes the wavelet

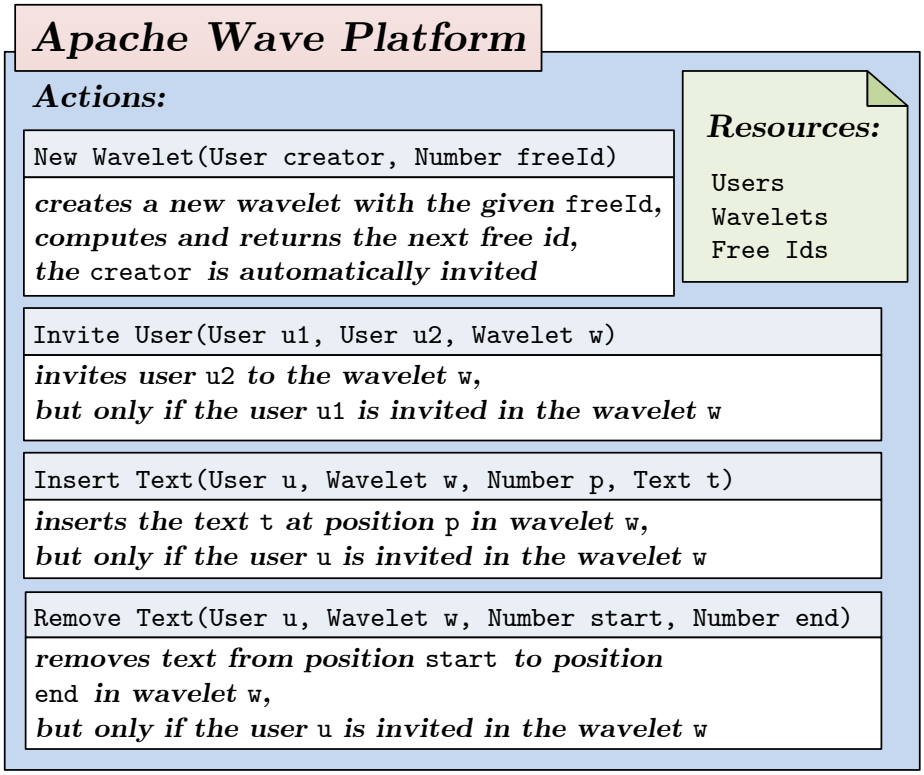


Figure 2.2: Resources and actions of a basic Apache Wave platform

by removing a part of the text. Finally, Bob inserts another text into the wavelet 1, where only he is invited.

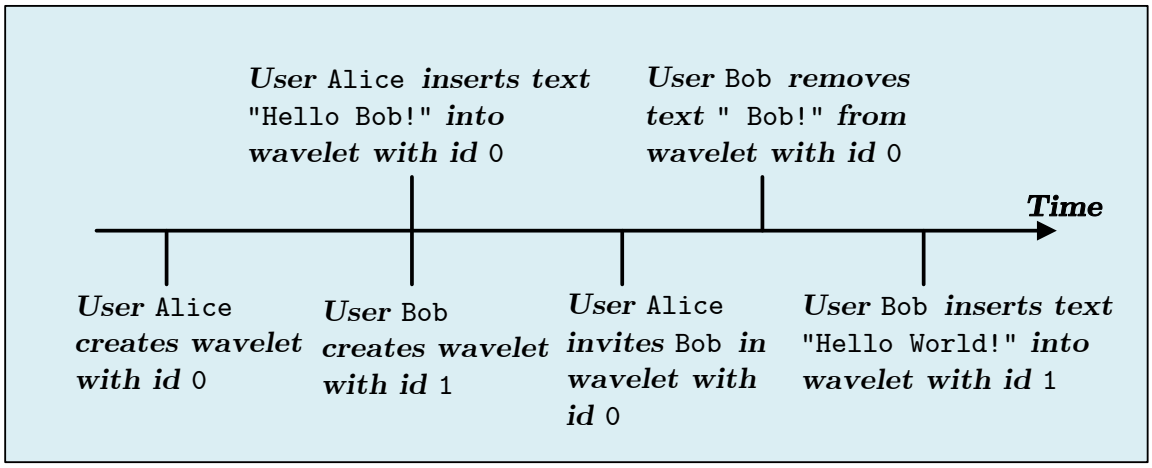


Figure 2.3: Concrete scenario of interactions in a wave

Note that subsequent events on the time line in Figure 2.2 are considered to happen one after another and in no other causal relation. On the other hand, the two events of Alice inserting the text ‘‘Hello Bob’’ and Bob creating the wavelet with id 1 are considered to happen simultaneously. Due to the network communication the events in the communication platform are only perceived in *near*-real time by the system and the other users. Therefore, it may be wise to consider events not only as simultaneous, if they actually happen at the

exact same time, but also, if they *can* happen at the same time. This includes all events for which it is possible that they can occur in every possible order, e. g. it is not important whether Bob creates the wavelet a few milliseconds before Alice inserts the text into the other wavelet, or if it is the other way around.

An abstraction of the concrete scenario is depicted in Figure 2.4, where the correct order of actions from the concrete scenario is retained, but we do not consider the concrete values for the different resources like users, texts and ids.

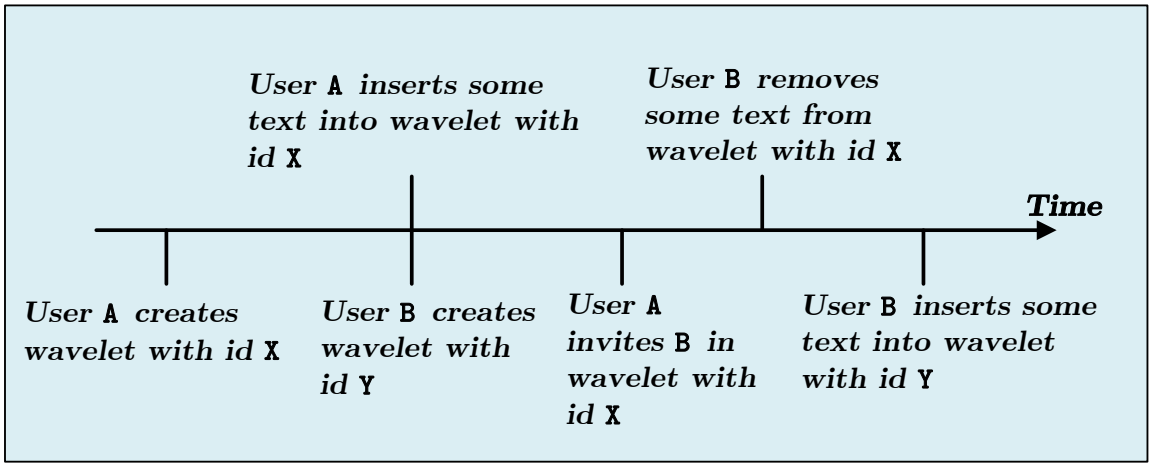


Figure 2.4: Abstract scenario of interactions in a wave

Another interesting aspect of the modelling of Apache Wave are dynamic changes to the structure of the platform. Since Apache Wave is open-source, it can be expected that platforms are constantly modified by potentially different developers.

Figure 2.5 shows a possible modification of the platform presented in Figure 2.2. In this platform, it is not possible to change a wavelet directly, but instead it is possible to modify the copy of a wavelet, because the actions *Insert Text* and *Remove Text* have been replaced by the action *Modify Copy*.

Since it is possible that the communication platform is modified at runtime there may already exist some waves that correspond to the old version of the platform. In some cases that correspondence could be violated by the modification of the platform.

An intuitive solution is to apply the modification of the platform also to the wave, replacing all occurrences of the old features with corresponding new ones if possible, or remove them otherwise. In the case of our scenario in Figure 2.3 and the platform evolution described above, this leads to new concrete and abstract scenarios depicted in Figure 2.6 and Figure 2.7, where the two occurrences of text insertions as well as the occurrence of removing a text have been replaced by occurrences of the modify copy action.

2.3 Requirements for the Modelling

From the given examples in Section 2.1, we can derive a number of requirements for the modelling of modern communication platforms.

Platforms For the modelling of the platforms, it is important to allow the modelling of the *users* as well as all sorts of *data* that is product of the communication between these users. Moreover, the model of a wave has to contain possibly different types of *actions* that can be used by the users to communicate with each other.

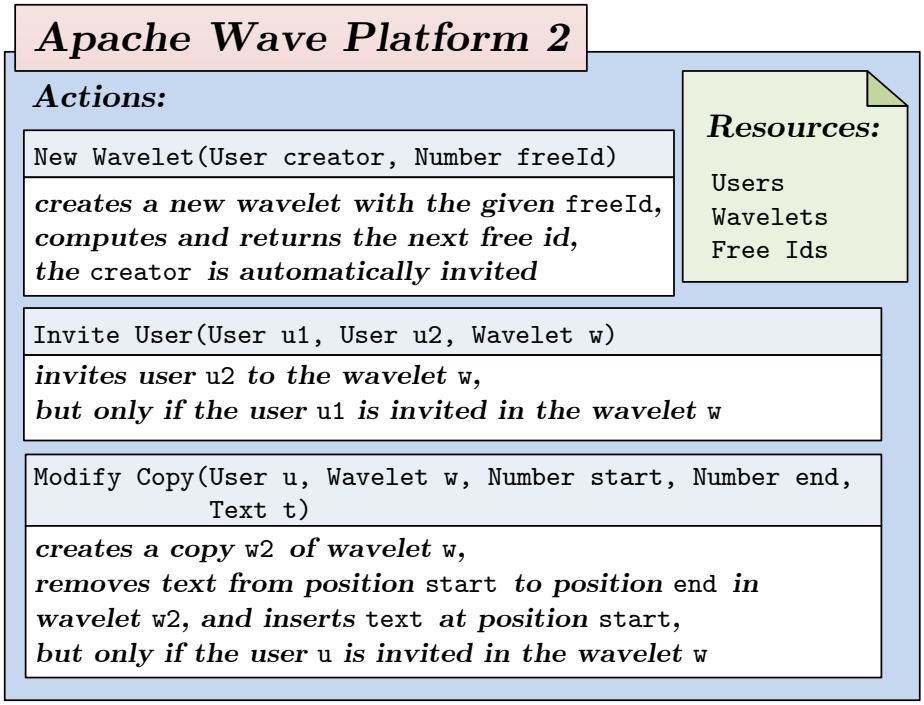


Figure 2.5: Modified Apache Wave platform

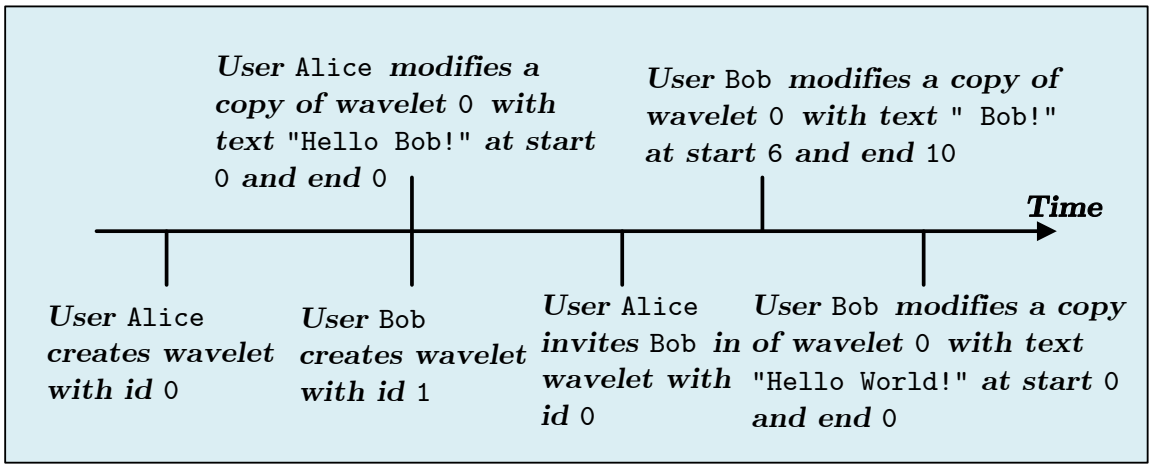


Figure 2.6: Modified concrete scenario of a wave

In some cases such as the different manifestations of Apache Wave, there may be different platforms with a different set of actions or data, but users on different platforms may still communicate with each other due to a common communication protocol. This means that the modelling of communication platforms should be *compositional*, allowing to view a combination of platforms as a platform.

Evolution of Platforms As we have seen in the examples above it should also be possible model the *evolution of platforms*, changing the actions and/or data of the platform.

Scenarios Since we want to investigate the interactions of human-centric communication platforms, we need a way to model these interactions. As the interactions usually

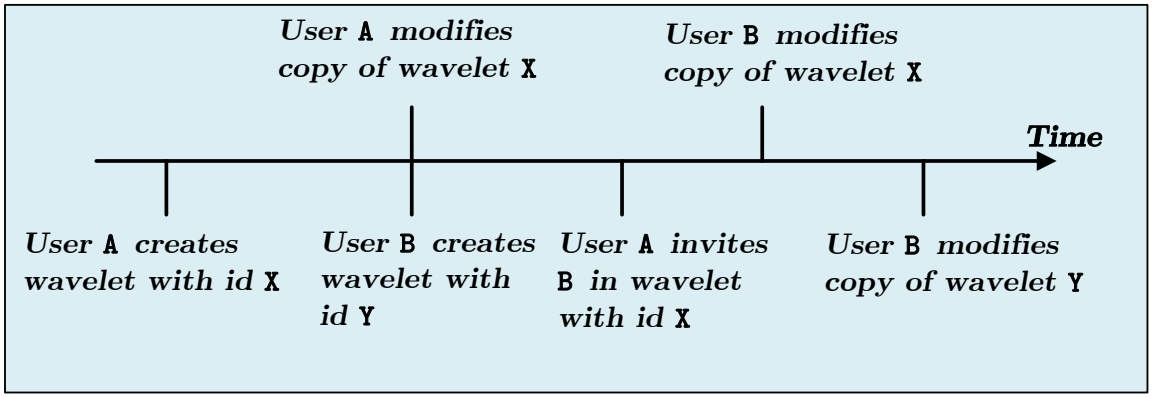


Figure 2.7: Modified abstract scenario of a wave

are performed by humans, we have to consider different *scenarios* of interactions on a platform. A scenario contains a number of actions performed by users, and these actions have to be in compliance with the actions and data that can be used on the corresponding platform. The model of a scenario should reflect the *causal relation* of subsequent actions, but it should also be possible to model the *concurrent* occurrence of actions, since usually it is possible that different users perform actions independently at the same time (see also the discussion on page 23 about a tolerant interpretation of simultaneity in near-real time communication).

Histories With a modelling technique for scenarios of interactions fulfilling these requirements, it should also be possible to adequately model the histories of waves.

Evolution of Scenarios Similar to the evolution of communication platforms it should be possible to model also the evolution of scenarios. Considering histories as special scenarios this should include the modelling of the progression of histories by user interactions.

Levels of Abstraction In order to identify and investigate specific patterns of interactions, we need a way to model different *levels of abstraction*, allowing us to consider concrete scenarios with the different concrete data values and users that are involved, as well as abstract scenarios, where the data and users are not fixed a priori. It should then be possible to relate abstract and concrete scenarios, if the first is an abstraction of the latter, or equivalently, the latter is a concrete realization of the first one.

Views on Scenarios Regarding the compositionality of platforms described above, it should also be possible to restrict scenarios to the current view on the platform. This is particularly interesting in the context of cross-platform communication, where different platforms together form a common larger platform.

Behaviour of Users (Robots) Analysing the interactions in human-centric communication platforms, we require not only a formal model of the platform, but also a suitable model of the behaviour of the users. This includes the behaviour of humans, and, considering robots as a special case of automated users, also the behaviour of robots. Formalisation of behavioural patterns should allow to analyse possible conflicts or dependencies in the interaction of humans and/or robots, making it possible to detect and prevent potential problems at the design-time of the platform.

2.4 Analysis Problems

Modelling techniques satisfying the requirements in the previous section may be used to address the following analysis problems concerning communication platforms and scenarios:

Compatibility and Composability of Platform Evolutions Especially in the case of open-source platforms, it can happen that outgoing from one platform there are different evolutions of that platform, introducing or removing features or resources, respectively. Considering two of these evolutions, it is an interesting question if the evolutions exclude or depend on each other. If this is not the case, it may be possible that the evolutions are compatible with each other, in the sense that one evolution can also be applied to the resulting platform of the other one and vice versa. In that case there should be a way to unite the different evolutions and obtain one evolution, consisting of the benefits of either of the single evolutions, but with the difference that the platform evolves in a single step. In the case of dependencies on the other hand, the preferred result would be a composition of the evolutions in the right order, assuring that no conflicts occur during the cumulative evolution of the platform.

The other way around, it could be helpful to decompose complex evolutions of platforms into smaller parts. Then the analysis of dependencies of the single parts with each other could allow to change the order of single changes in order to increase the performance, or to exclude undesired parts of the evolution.

Compatibility between Scenarios and Platform Evolutions Moreover, due to the fact that platforms can evolve, we need a way to check if scenarios of the original platform are still in compliance with the result of the modification. Otherwise, it is an interesting question, if scenarios can be adapted appropriately, in order to obtain a consistent scenario corresponding to the new platform.

Conflicts between Interactions Considering interactive (near-)real-time communication using modern platforms it is possible that many users perform different action at the same time. In the case that they are using the same resources this may lead to conflicts. This can especially be a problem when a variety of different robots is involved in the communication. It would be helpful to detect these conflicts at design-time to prevent problems that may occur during the communication.

Concrete Realisations of Abstract Scenarios Considering abstract scenarios without any specific data or users, the question arises if there are at all concrete values that can be used to obtain a concrete scenario that is a realization of the abstract scenario.

Cross-Platform Communication Another interesting aspect for the analysis is the possibility of communication participants on different platforms to communicate with each other. Considering the communication interface it may be necessary to exchange specific resources in order to ensure that the communication works correctly.

2.5 Formal Techniques for Modelling and Analysis

Based on the requirements for the modelling of communication platforms as identified in the previous section, we investigate different modelling techniques to which extent they meet these requirements.

Platforms The Unified Modelling Language (UML) provides a large set of different graphical and textual techniques for the modelling of object-oriented systems [BRJ98, BJR99].

For instance, class diagrams describe the static structure of a system by describing the relation between different classes, object diagrams are graphs of instances, describing the static state of a system, whereas collaboration diagrams model the dynamic changes of states.

Moreover, the UML offers a variety of techniques for describing the behaviour of systems, such as sequence diagrams, use case diagrams and activity diagrams. The main drawback of the UML languages is that – although there have been some efforts for a formalisation of single parts of the UML [BHH+97, GPP98, BCD+06] – there does not exist a consistent formal semantics for the whole UML framework.

Another well-known visual modelling technique are Petri nets that have been started by Carl Adam Petri in [Pet62]. In the basic approach of low-level place/transition (P/T) nets [Roz87, Rei85, MM90], the nets consist of places and transitions. The operational semantics of P/T nets, the so-called token-game, involves a marking of indistinguishable tokens that can be present on places, and the marking can be changed by firing transitions under certain conditions.

High-level nets based on low-level P/T nets and data types have been studied in [GL81, Gen87] as Predicate/Transition nets, and with data types in the programming language ML in [Jen91, JR91, Jen97a] as coloured Petri nets, providing the modeller with various verification techniques [Jen97b] and comprehensive tool-support that has been used in many different application areas [Jen97c].

The combination of Petri nets and algebraic specifications[EM85] was mainly initiated by Krämer [Krä87, Krä89] and extended in [Hum89, Rei90, DHP91, EPR92, ER97] leading to the notion of algebraic high-level (AHL) nets. The mathematical formalisation of structuring techniques like the gluing or restriction of algebraic high-level nets allow the compositional modelling and analysis of complex systems [Lil91, GE00].

Evolution of Platforms A very powerful and well researched technique for the evolution of graphs is rule-based graph transformation [Ehr79, Roz97]. There are several general frameworks such as the framework of adhesive [LS04], weak adhesive HLR [EEPT06b] and $\mathcal{M}$-adhesive transformation systems [EGH10], allowing to apply the idea of rule-based transformation not only to pure graphs but to a large set of different graph-like structures. This includes for instance the evolution of UML diagrams [GPP98, KGKK02], of AHL-nets [PER95, Pra07], and even AHL-systems, that is, AHL-nets together with markings [Pra08, MGE+10, MGH10].

Besides the general applicability of the frameworks, they have the benefit to offer results and analysis techniques like the Local Church-Rosser, Local Confluence, Parallelism, Extension and Embedding Theorems, and Critical Pair Analysis.

Scenarios Considering the fact that a scenario has to be in compliance with the respective platform it corresponds to, it may a good idea to regard a scenario as a typed object or graph[EEPT06a, EEPT06b], in the sense that the scenario contains resources (like data elements or users) as well as actions that are instances of corresponding elements that are present in the platform model.

In the case of Petri nets, there is a special notion of typed objects, called Petri net processes. A low-level Petri net process [GR83, Roz87, DMM89, Eng91, MMS97, BCEH01] $p : K \to N$ of a net N consists of a special conflict-free and acyclic Petri net K typed over N, modelling up to concurrency a specific firing sequence of transitions that is compatible with the causal relation of N.

Accordingly, there is a corresponding notion of algebraic high-level (AHL-) processes [EHP+02, Ehr05] defined as special nets typed over an algebraic high-level net. It is important to note that in contrast to the low-level case, one high-level process does not necessarily model a single (concurrent) firing sequence, due to possibly different data-elements that can be involved. A concrete choice of data-elements in a process can be fixed using the notion of instantiations [Ehr05]. Another interesting aspect is that there are also results regarding the composition and rule-based transformation of algebraic high-level processes [EHGP09, EG11, Gab12b]. Moreover, there are several results regarding the views on typed objects, using restriction and amalgamation of typed objects [SEM+12], open low-level Petri net processes [BCEH01] and AHL-processes [EG11].

Another family of techniques for the formal modelling of concurrent systems are the process algebras or process calculi. One example of a process calculus is the Calculus of Communication Systems (CCS) by R. Milner [Mil80]. It allows to model communication process between two participants, and contains basic combinators for the description of parallel composition, choices between actions and scope restriction. Communicating Sequential Processes (CSP) by C.A.R. Hoare [Hoa85] can be used to model independent component processes that interact with each other through message passing. In contrast to these modelling techniques, where the processes have a static topology, the π-calculus [Mil99] allows to dynamically change the network structure of the processes.

2.6 Algebraic High-Level Nets and Processes for Modelling and Analysis

For the formal modelling of communication platforms we choose the modelling technique of algebraic high-level (AHL-)nets [EPR92, PER95]. An AHL-net is a special kind of Petri net [Pet62]. Petri nets can be seen as directed hypergraphs, where the nodes are called places and the edges are called transitions. The places represent local states or resources of the modelled system, and they can equip a number of tokens, meaning that the corresponding state is active, or the resource is present or occupied, respectively. The operational semantics of Petri nets is given by the so-called token-game, where tokens on all places in the predomain (the source) of a transition are consumed, while on all places in the postdomain (the target) of that transition tokens are produced. We then say that the transition fired.

In contrast to classical low-level place/transition (P/T) nets, an AHL-net consists of an algebraic data type part [EM85]. This means that the tokens are data values rather than indistinguishable "black tokens", and the transitions have firing conditions, allowing the firing of a transition only if the tokens that are to be consumed on firing satisfy the conditions of that transition. Considering the platform as the basis of all communication processes, the formal model of a platform has to describe all the resources and actions that are available on that platform. The modelling of communication platforms using AHL-nets is described in Section 3.1.

The rule-based transformation of AHL-nets in the sense of graph transformation [Ehr79, Roz97] is well researched [PER95, Pra07]. Since AHL-nets fit into the general frameworks of weak adhesive high-level replacement [EEPT06b] and $\mathcal{M}$-adhesive systems [EGH10], there are even powerful analysis techniques like the Local Church-Rosser, Local Confluence, Parallelism, Extension and Embedding Theorems, and Critical Pair Analysis [EEPT06b] that can be used for the analysis of AHL-nets. Therefore, we choose to model the evolution of communication platforms using rule-based transformation of AHL-nets, as described in Section 3.2.

As modelling technique for waves and scenarios of interactions in platforms, we choose

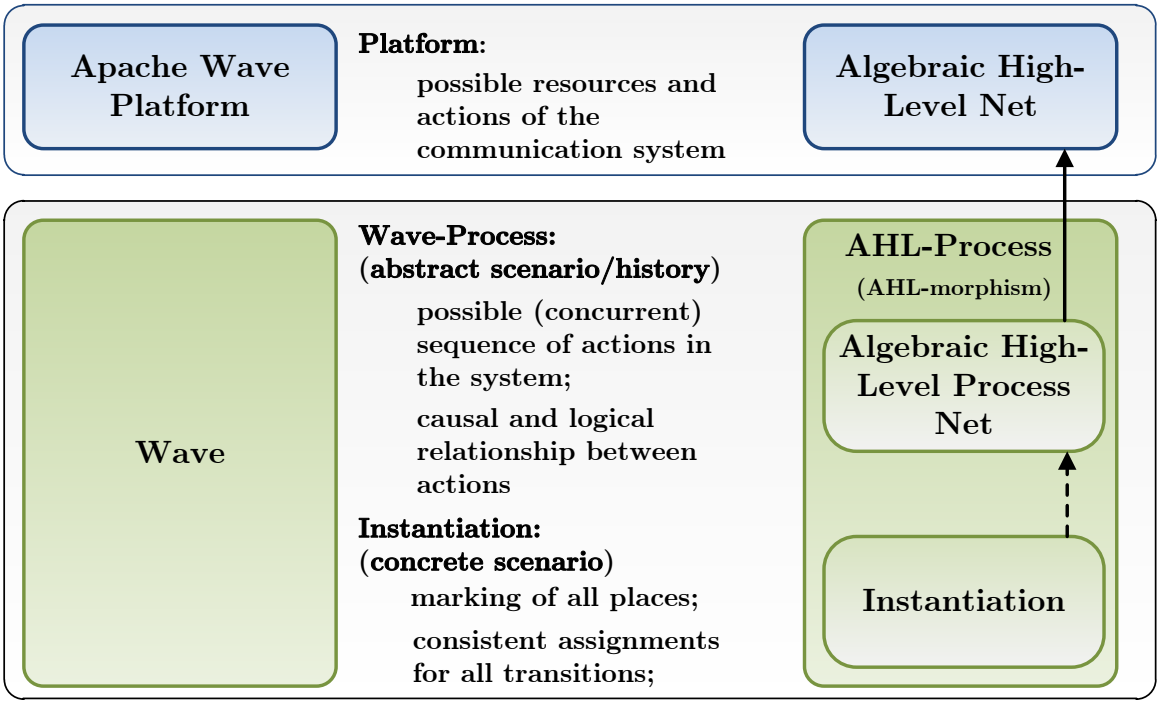


Figure 2.8: Modelling of communication platforms and scenarios with AHL-nets and -processes

algebraic high-level (AHL-)processes with instantiations. An instantiated AHL-process models up to concurrency a specific sequence of actions in an AHL-net. Since the components of an AHL-process are AHL-nets, we can in principal use the same structuring techniques for scenarios as for AHL-nets, however, with some additional requirements in order to preserve the process character of the models. The modelling and evolution of scenarios is described in Chapter 4.

All in all, we obtain an integrated framework for the modelling of communication platforms and scenarios using algebraic high-level nets and instantiated AHL-processes (see Figure 2.8). For an overview over all realisations of modelling requirements, we refer to Section 8.1, and for an overview over the corresponding analysis results see Section 8.2.

3 Modelling and Evolution of Communication Platforms

In Section 3.1 we review the modelling framework of algebraic high-level (AHL-)nets [EPR92, PER95, EEPT06b] which are an integration of (low-level) Petri nets [Pet62] and algebraic specification techniques [EM85]. Moreover, in Section 3.2 we review the rule-based reconfiguration of AHL-nets in the sense of graph transformation [Roz97, PER95]. We show how AHL-nets can be used to model Apache Wave platforms, and we demonstrate the evolution of these platforms by rule-based transformation of the respective AHL-nets.

3.1 Modelling of Communication Platforms Using Algebraic High-Level Nets

In this section we review the concepts of low-level place/transition and algebraic high-level Petri nets. As net formalism we use place/transition nets following the notation of "Petri nets are Monoids" in [MM90], where $X^{\oplus}$ is the free commutative monoid over the set X. Note that $s \in X^{\oplus}$ is a formal sum $s = \sum_{i=1}^{n} \lambda_i x_i$ with $\lambda_i \in \mathbb{N}$ and $x_i \in X$ meaning that we have λ_i copies of x_i in s and for $s' = \sum_{i=1}^{n} \lambda'_i x_i$ we have $s \oplus s' = \sum_{i=1}^{n} (\lambda_i + \lambda'_i) x_i$.

Definition 3.1.1 (Place/Transition Net)**.** A place/transition (P/T) net $N = (P, T, pre, post)$ consists of sets P and T of places and transitions respectively, and pre- and post domain functions $pre, post : T \to P^{\oplus}$ where $P^{\oplus}$ is the free commutative monoid over P.

A P/T-net morphism $f : N_1 \to N_2$ is given by $f = (f_P, f_T)$ with functions $f_P : P_1 \to P_2$ and $f_T : T_1 \to T_2$ satisfying

$$f_P^{\oplus} \circ pre_1 = pre_2 \circ f_T \text{ and } f_P^{\oplus} \circ post_1 = post_2 \circ f_T$$

where the extension $f_P^{\oplus} : P_1^{\oplus} \to P_2^{\oplus}$ of $f_P : P_1 \to P_2$ is defined by $f_P^{\oplus}(\sum_{i=1}^{n} k_i \cdot p_i) = \sum_{i=1}^{n} k_i \cdot f_P(p_i)$. A P/T-net morphism $f = (f_P, f_T)$ is called injective if f_P and f_T are injective and is called isomorphism if f_P and f_T are bijective.

The category defined by P/T-nets and P/T-net morphisms is denoted by **PTNets** where the composition of P/T-net morphisms is defined componentwise for places and transitions. △

Definition 3.1.2 (Marking, Firing Behaviour of Place/Transition Nets)**.** A *marking* $M = \sum_{i=1}^{n} \lambda_i p_i \in P^{\oplus}$ means that of a P/T net N means that place p_i contains λ_i tokens. Given a P/T net N with marking M, then a transition $t \in T$ is enabled under M, if $pre(t) \leq M$. In this case, the transition can be *fired* and the follower marking M' is given by

$$M' = M \ominus pre(t) \oplus post(t)$$

△

Example 3.1.3 (Place/Transition Net)**.** An example of place/transition net is shown in Figure 3.1. The P/T net *LLPlatform* is a low-level is P/T net representation of our Apache Wave platform example in Section 2.2 (see Figure 2.2), modelling the resources and actions of the platform. The P/T net consists of places u, w and id, modelling users, wavelets and IDs, respectively. Moreover, for each action of the platform, there is a corresponding transition in the P/T net. The pre domain of each transition contains all the resources the corresponding action is using, e. g. the *new wavelet* transition uses an ID and a user, the creator of the new wavelet. The post domain contains all resources that are produced by the corresponding action. In the case of the transition *new wavelet* this is the next free ID, the newly created wavelet and also the creator of the wavelet (because the user is still present in the system).

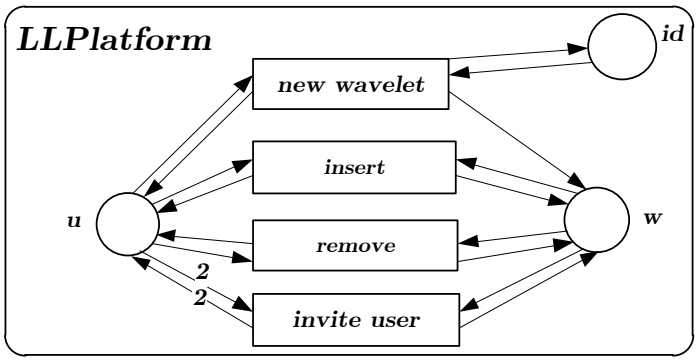


Figure 3.1: Place/transition net *LLPlatform*

Let us consider a marking

$$M = 2u \oplus id \oplus w$$

of the P/T net *LLPlatform*, meaning that there are two users, an ID and a wavelet in the platform. The transition *insert* is enabled, because we have

$$pre(insert) = u \oplus w \leq 2u \oplus id \oplus w = M$$

Thus, the transition *insert* can be fired and we obtain a new marking

$$M' = M \ominus pre(insert) \oplus post(insert) = 2u \oplus id \oplus w \ominus (u \oplus w) \oplus u \oplus w = M$$

♢

The example of a firing step above illustrates that there is something missing in the formalism of P/T nets in order to adequately model our communication platform. Although we fire the *insert* transition which models the *Insert Text* action from our platform in Figure 2.2, the resulting state of the platform (the follower marking) equals the original state before firing. This is due to the fact, that we do not have any modelling of the data in our P/T net platform model, and therefore, one wavelet is indistinguishable from another wavelet.

In the following we review the definition of algebraic high-level nets from [PER95, Ehr05, EHP+02], extending the above definition of place/transition nets by integration of algebraic data types.

Definition 3.1.4 (Algebraic High-Level Net)**.** An *algebraic high-level (AHL-) net*

$$AN = (\Sigma, P, T, pre, post, cond, type, A)$$

consists of

- a signature $\Sigma = (S, OP; X)$ with additional variables X;
- a set of places P and a set of transitions T;
- pre- and post domain functions $pre, post : T \to (T_\Sigma(X) \otimes P)^\oplus$;
- firing conditions $cond : T \to \mathcal{P}_{fin}(Eqns(\Sigma; X))$;
- a type of places $type : P \to S$ and
- a Σ-algebra A

where the signature $\Sigma = (S, OP)$ consists of sorts S and operation symbols OP, $T_\Sigma(X)$ is the set of terms with variables over X,

$$(T_\Sigma(X) \otimes P) = \{(term, p) | term \in T_\Sigma(X)_{type(p)}, p \in P\}$$

and $Eqns(\Sigma; X)$ are all equations over the signature Σ with variables X. △

The algebra as well as the variables of an AHL-net depend on the corresponding signature. So in order to allow mappings between AHL-nets with different data type parts, we need constructions that allow the mapping of algebras and sorted sets dependent on a corresponding mapping of signatures. For this purpose a suitable framework are Grothendieck categories based on the notion of indexed categories, because there are general categorical results for the construction of limits and colimits in Grothendieck categories [TBG91, Gog06]. For details on Grothendieck categories, we refer to Subsection A.1.2.

The following definition AHL-morphisms is very similar to the one in [PER95]. However, for the mapping of the variables in an AHL-net, we use a **SigSets**-morphisms as defined in Definition A.1.8.

Definition 3.1.5 (AHL-Morphism and Category **AHLNets**)**.** An *AHL-net morphism* $f : AN_1 \to AN_2$ is given by $f = (f_\Sigma, f_P, f_T, f_A)$, where

- $f_\Sigma = (f_\sigma, f_X)$ is a **SigSets**-morphism (see Definition A.1.8) with $f_\sigma = (f_S, f_{OP}) : \Sigma_1 \to \Sigma_2$ being a signature morphism[4] and $f_X = (f_{X,s})_{s \in S_1} : X_1 \to X_2$ a mapping of the variables, where we additionally require that for all $t \in T_1$ the restriction $f_X|_{Var(t)} : Var(t) \to Var(f_T(t))$ of f_X to variables of t is (componentwise) bijective,
- $f_P : P_1 \to P_2$ and $f_T : T_1 \to T_2$ are functions, and
- $(f_\sigma, f_A) : (\Sigma_1, A_1) \to (\Sigma_2, A_2)$ is a generalised algebra homomorphism (see Example A.1.7), i. e. $f_A : A_1 \to V_{f_\sigma}(A_2)$ is a Σ_1-homomorphism,

satisfying the following conditions[5]:

(1) $(f_\Sigma^\# \otimes f_P)^\oplus \circ pre_1 = pre_2 \circ f_T$ and
$(f_\Sigma^\# \otimes f_P)^\oplus \circ post_1 = post_2 \circ f_T$,
(2) $cond_2 \circ f_T = \mathcal{P}_{fin}(f_\Sigma^\#) \circ cond_1$ and
(3) $type_2 \circ f_P = f_\Sigma \circ type_1$.

[4] We will occasionally refer to f_Σ as signature morphism, despite the fact that it contains an additional part f_X.

[5] $f_\Sigma^\#$ is the free extension of f_Σ to terms and equations (see [EM85])

$$\begin{array}{ccccc} \mathcal{P}_{fin}(Eqns(\Sigma_1)) & \xleftarrow{cond_1} & T_1 & \overset{pre_1}{\underset{post_1}{\rightrightarrows}} & (T_{OP_1}(X_1) \otimes P_1)^{\oplus} \\ \downarrow{\scriptstyle \mathcal{P}_{fin}(f_\Sigma^\#)} & (\mathbf{2}) & \downarrow{\scriptstyle f_T} & (\mathbf{1}) & \downarrow{\scriptstyle (f_\Sigma^\# \otimes f_P)^{\oplus}} \\ \mathcal{P}_{fin}(Eqns(\Sigma_2)) & \xleftarrow[cond_2]{} & T_2 & \overset{pre_2}{\underset{post_2}{\rightrightarrows}} & (T_{OP_2}(X_2) \otimes P_2)^{\oplus} \end{array} \qquad \begin{array}{ccc} P_1 & \xrightarrow{type_1} & S_1 \\ \downarrow{\scriptstyle f_P} & (\mathbf{3}) & \downarrow{\scriptstyle f_S} \\ P_2 & \xrightarrow[type_2]{} & S_2 \end{array}$$

The category defined by AHL-nets and AHL-net morphisms is denoted by **AHLNets** where the composition and identities are defined componentwise. For a given data type part (Σ, A), we define the subcategory $\mathbf{AHLNets}(\boldsymbol{\Sigma}, \mathbf{A})$ of AHL-nets with data type part (Σ, A) and AHL-morphisms with identical data type part (id_Σ, id_A). △

Remark 3.1.6 (Algebraic High-Level Nets and Morphisms). There are different variants of high-level nets and morphisms in the literature. In the past, the data type of high-level nets consisted not only of a signature Σ but of an algebraic specification SP. However, every signature Σ can be considered as a special case of an algebraic specification $SP = (\Sigma, \emptyset)$ with an empty set of equations.

Moreover, for AHL-morphisms it was often required that the homomorphism $f_A : A_1 \to V_{f_\Sigma}(A_2)$ is an isomorphism (see Definition 4.6 in [PER95]), in order to allow rule-based transformation (see Section 3.2) to change not only the structure but also the specification part of an AHL-net. We drop this restriction and consider arbitrary generalised homomorphisms. On the one hand, with our definition of AHL-morphisms it is not possible to modify the data type part of an AHL-net using rule-based transformation, but on the other hand, it allows more general application of productions, since the data type part of the production does not have to be isomorphic to the data type part of an AHL-net to which the production is applied. △

The firing behaviour of AHL-nets is defined analogously to the firing behaviour of low-level nets. The difference is that in the high-level case all tokens are equipped with data values. Moreover, for the activation of a transition t, we additionally need an assignment v of the variables $Var(t)$ in the environment of the transition, such that the assigned pre domain is part of the given marking and the firing conditions of the transition are satisfied. This assignment is then used to compute the follower marking, obtained by firing of transition t with assignment v.

Definition 3.1.7 (Marking, Firing Behaviour)**.** A marking

$$M = \sum_{i=1}^{n} \lambda_i(a_i, p_i) \in (A \otimes P)^{\oplus}$$

of an AHL-net AN means that place p_i contains $\lambda_i \in \mathbb{N}$ data tokens $a_i \in A_{type(p_i)}$. Given an AHL-net AN with marking M. Then a transition $t \in T$ is enabled under M and an assignment $v : Var(t) \to A$, if all firing conditions $cond(t)$ are satisfied in A for v and we have enough tokens in the pre domain of t, i. e. $pre_A(t, v) \leq M$, where

$$pre_A(t, v) = \sum_{i=1}^{n} (v^*(term_i), p_i) \quad \text{for} \quad pre(t) = \sum_{i=1}^{n} (term_i, p_i)$$

with $term_i \in T_{OP}(X)$ and $v^*(term_i)$ is the evaluation of $term_i$ under v. In this case the follower marking M' is given by

$$M' = M \ominus pre_A(t, v) \oplus post_A(t, v).$$

△

Concept 3.1.8 (Modelling of Communication Platforms)**.** A communication platform can be modelled as an algebraic high-level net. The resources, including the users and communication-related data are structurally modelled as places while the concrete types and data values along with available data operations are given as an algebra.

The possible actions that are provided by the platform are modelled as transitions. Resources that are used or required by an action are contained in the pre conditions of the corresponding action, while the post conditions of transitions consist of those resources that are produced or retained by the corresponding action. The concrete data operations that are performed by each action are implemented in the firing conditions of the corresponding transition. △

Example 3.1.9 (Modelling of Apache Wave Platform as Algebraic High-Level Net)**.** The model of an Apache Wave platform depicted in Figure 3.2 is an AHL-net

$$Platform = (\Sigma\text{-}Wave, P, T, pre, post, cond, type, A)$$

where the signature Σ-$Wave$ is shown in Table 3.1 and the Σ-$Wave$-algebra A is shown in Table 3.2. This signature and algebra is also used for all the following examples.

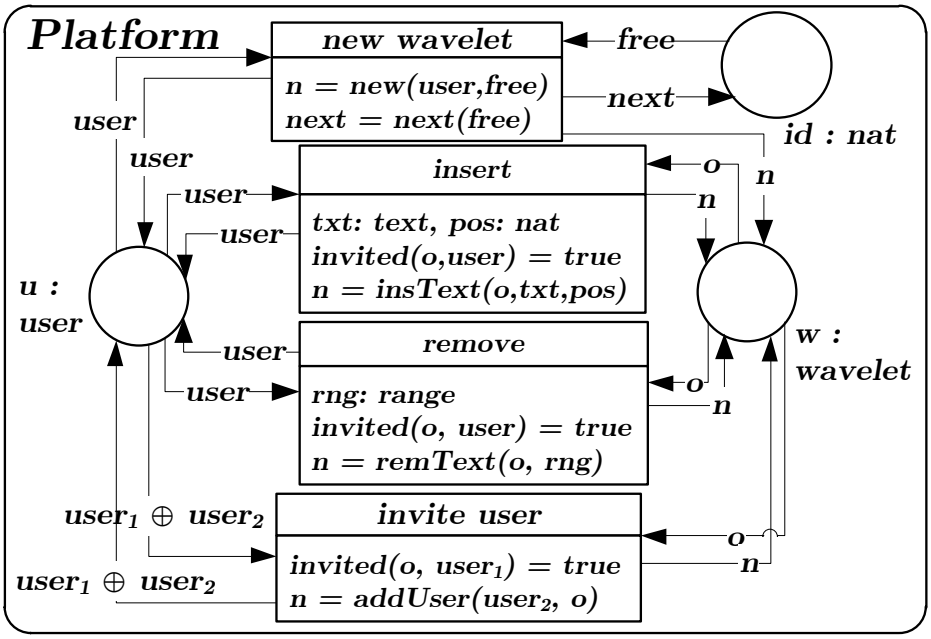


Figure 3.2: AHL-net *Platform*

The AHL-net *Platform* has the same structure as the low-level P/T net *LLPlatform* in Figure 3.1, but additionally it has a data type part, further specifying the type of resources and the semantics of the transitions. The places u, w and id have types $user$, $wavelet$ and nat, respectively. Moreover, each transition has a set of equations (firing conditions), and there are terms on the pre and post arcs, constraining the resources consumed and produced by firing of the transition.

Let us consider a marking

$$M = (Alice, u) \oplus (Bob, u) \oplus (1, id) \oplus ((0, \{Alice, Bob\}, \epsilon), w)$$

of the AHL-net *platform* in Figure 3.2 which means that we have two users *Alice* and *Bob* on the place u, a free ID 1 and an empty wavelet with ID 0 on place w where *Alice* and *Bob* are invited.

Considering all variables in the environment of the transition *insert*, we have a set of variables $Var(insert) = \{user, txt, pos, o, n\}$. An assignment $v : Var(insert) \to A$ with $v(user) = Alice$, $v(txt) = Hello\ Bob$, $v(pos) = 0$, $v(o) = (0, \{Alice, Bob\}, \epsilon)$ and $v(n) = (0, \{Alice, Bob\}, Hello\ Bob)$ satisfies the firing conditions of the transition *insert*, because

$$\begin{aligned} v^*(invited(o, user)) &= invited_A(v(o), v(user)) \\ &= invited_A((0, \{Alice, Bob\}, \epsilon), Alice) \\ &= true_A \\ &= v^*(true) \end{aligned}$$

and

$$\begin{aligned} v^*(n) &= (0, \{Alice, Bob\}, Hello\ Bob) \\ &= insText_A((0, \{Alice, Bob\}, \epsilon), Hello\ Bob, 0) \\ &= insText_A(v(o), v(txt), v(pos)) \\ &= v^*(insText(o, txt, pos)) \end{aligned}$$

means that v satisfies all equations in $cond(insert)$, and we have

$$\begin{aligned} pre_A(insert, v) &= (v(user), u) \oplus (v(o), w) \\ &= ((Alice, u) \oplus (0, \{Alice, Bob\}, \epsilon) \leq M \end{aligned}$$

By firing the transition *insert* with assignment v we obtain the follower marking

$$M' = (Alice, u) \oplus (Bob, u) \oplus, (1, id) \oplus ((0, \{Alice, Bob\}, Hello\ Bob), w)$$

where the assigned text *Hello Bob* has been inserted at position 0 into the assigned wavelet. ♢

Table 3.1: Signature Σ-$Wave$

sorts:	bool, nat, range, text, user, wavelet	
opns:	true, false : → bool	zero : → nat
	next : nat → nat	start: range → nat
	toRange : nat nat → range	new : user nat → wavelet
	addUser : user wavelet → wavelet	invited : wavelet user → bool
	len : text → nat	sub : text range → text
	insText : wavelet text nat → wavelet	remText : wavelet range → wavelet
	empty : → text	
vars:	free, next : nat	rng : range
	txt : text	user, $user_1$, $user_2$: user
	o, n, r : wavelet	

Fact 3.1.10 (AHL-Morphisms Preserve Firing Behaviour)**.** *Given an AHL-net morphism $f : AN_1 \to AN_2$, the firing behaviour is preserved by f. That is, for an assignment $v_1 : Var(t) \to A_1$ and a marking M_1 with firing step $M_1 \xrightarrow{t,v_1} M_1'$ in AN_1, there is corresponding firing step $(f_A \otimes f_P)^{\oplus}(M_1) \xrightarrow{f_T(t),v_2} (f_A \otimes f_P)^{\oplus}(M_2)$ in AN_2 with $v_2 = f_A \circ v_1 \circ (f_X|_{Var(t)})^{-1}$.*

Table 3.2: Σ-$Wave$-algebra A

$A_{bool} = \{T, F\}$	$A_{nat} = \mathbb{N}$
$A_{user} = \{a, \ldots, z, A, \ldots, Z\}^*$	$A_{text} = \{a, \ldots, z, A, \ldots, Z, \ldots\}^*$
$A_{wavelet} = A_{nat} \times \mathcal{P}(A_{user}) \times A_{text}$	$A_{range} = A_{nat} \times A_{nat}$

$true_A = T \in A_{bool}$

$false_A = F \in A_{bool}$

$start_A : A_{range} \to A_{nat}$

$\quad (s, l) \mapsto s$

$toRange_A : A_{nat} \times A_{nat} \to A_{range}$

$\quad (s, l) \mapsto (s, l)$

$zero_A = 0 \in A_{nat}$

$next_A : A_{nat} \to A_{nat}$

$\quad n \mapsto n + 1$

$new_A : A_{user} \times A_{nat} \to A_{wavelet}$

$\quad (u, id) \mapsto (id, \{u\}, \epsilon)$

$addUser_A : A_{user} \times A_{wavelet} \to A_{wavelet}$

$\quad (u, (id, uset, t) \mapsto (id, uset \cup \{u\}, t)$

$invited_A : A_{wavelet} \times A_{user} \to A_{bool}$

$$(u, (id, uset, t)) \mapsto \begin{cases} T & \text{, if } u \in uset; \\ F & \text{, else.} \end{cases}$$

$len_A : A_{text} \to A_{nat}$

$$t \mapsto \begin{cases} 0 & \text{, if } t = \epsilon; \\ 1 + len_A(t_1 \ldots t_n) & \text{, if } t = t_0 \ldots t_n. \end{cases}$$

$sub_A : A_{text} \times A_{range} \to A_{text}$

$$(t, (s, l)) \mapsto \begin{cases} \epsilon & \text{, if } l = 0 \text{ or } len_A(t) \leq s; \\ t_s \ldots t_n & \text{, if } l \neq 0, s < m, t = t_0 \ldots t_m, \text{ and } n = min(m, s + l). \end{cases}$$

$insText_A : A_{wavelet} \times A_{text} \times A_{nat} \to A_{wavelet}$

$\quad ((id, uset, t), nt, pos) \mapsto (id, uset, sub_A(t, (0, pos)).nt.sub_A(t, (pos, len_A(t))))$

$remText_A : A_{wavelet} \times A_{range} \to A_{wavelet}$

$\quad ((id, uset, t), (s, l)) \mapsto (id, uset, sub_A(t, (0, s)).sub_A(t, (s + l, len_A(t))))$

$empty_A = \epsilon \in A_{text}$

Proof. This is shown in the proof for Remark 2.10 in [Ehr05] for a slightly different definition of AHL-morphisms (see Remark 3.1.6), but the proof works also for our definition. □

Remark 3.1.11 (AHL-Nets with Individual Tokens). In contrast to the firing behaviour defined in Definition 3.1.7 it is also possible to define a marking over a set I of individuals and a marking function $m : I \to A \otimes P$ assigning each individual to a pair of a data element and a place. This makes it possible to distinguish the single tokens of a marking.

In order to fire a transition under a given marking it is then necessary to specify a token selection (M, m, N, n) where $M \subseteq I$ is the set of individuals which are consumed by the transition, N is a set of newly created individuals with $(I \setminus M) \cap N = \emptyset$ and $m : M \to A \otimes P$, $n : N \to A \otimes P$ are corresponding marking functions. If a selection together with a consistent transition assignment (t, asg) meets the *token selection condition*:

$$\sum_{i \in M} m(i) = pre_A(t, asg) \quad \text{and} \quad \sum_{i \in N} n(i) = post_A(t, asg)$$

then t is asg-enabled and the follower marking (I', m') can be computed by

$$I' = (I \setminus M) \cup N, \quad m' : I' \to A \otimes P \quad \text{with} \quad m'(x) = \begin{cases} m(x), \text{ if } x \in I \setminus M; \\ n(x), \text{ if } x \in N. \end{cases}$$

Although this *individual token approach* is more complicated than the *collective token approach* in Definition 3.1.7 it has some benefits like the possibility to formulate transformation rules which can not only change the net structure but also the marking of an AHL-net. For more details we refer to [MGE+10, Mod12]. However, in this thesis we concentrate more on the process semantics of AHL-nets rather than on single markings and the firing of single transitions. △

3.2 Structural Evolution of Platforms

Due to the possibility to evolve the Apache Wave platforms by adding, removing or changing features we need also techniques that make it possible to evolve the corresponding model of a platform. For this reason we introduce rule-based AHL-net transformations [PER95] in the sense of graph transformations [Roz97].

A production (or transformation rule) for AHL-nets specifies a local modification of an AHL-net. It consists of a left-hand side, an interface which is the part of the left-hand side which is not deleted and a right-hand side which additionally contains newly created net parts.

Definition 3.2.1 (Productions for AHL-Nets)**.** A *production for AHL-nets* is a span $\varrho : L \xleftarrow{l} I \xrightarrow{r} R \in \mathcal{M}_{AHL}$, where $\mathcal{M}_{AHL}$ is the class of all injective AHL-morphisms with isomorphic data type part. We call L the left-hand side, I the interface, and R the right-hand side of the production ϱ. In most examples l and r are inclusions. △

Remark 3.2.2 (Productions with $\mathcal{M}$-Morphisms). Note that the restriction of rule morphisms to $\mathcal{M}_{AHL}$-morphisms with isomorphic data type part in the definition above is not necessary. In [PER95] productions for AHL-nets are defined via morphisms in a class

$$M_{AHLNET} = \{f = (f_{SPEC}, f_P, f_T, f_A) \mid f_{SPEC} \text{ strict injective and } f_P, f_T \text{ injective}\}$$

however, with the overall restriction that all AHL-morphisms are required to have an isomorphic algebra part (see Remark 3.1.6). A specification morphism $f_{SPEC} : SP_1 \to SP_2$ is called strict, if all equations in SP_2 can be derived from equations in SP_1. With that definition of productions for AHL-nets, it is possible to change not only the structure but also the specification part of an AHL-net using rule-based transformation.

Although the adaptation of the signature part is not possible with our definition, the restriction of rule-morphisms to $\mathcal{M}_{AHL}$-morphisms has several advantages. The main advantage is the fact that with our definition we obtain an $\mathcal{M}$-adhesive transformation system (see Subsection A.1.3) with a variety of useful results. In [EGH10] it is shown that all results for weak adhesive HLR systems [EEPT06b] are also valid for the slightly more general framework of $\mathcal{M}$-adhesive categories. These results include the Local Church-Rosser and Parallelism Theorems which are concerned with independent transformations, the Concurrency theorem which is concerned with dependent transformations, the Completeness of Critical Pairs which describes that we can find a set of *critical pairs* that completely describe all conflicts that occur in a given transformation system, the Local Confluence Theorem, stating that a transformation system is locally confluent if all its critical pairs are strictly confluent, where the strictness condition enhances standard local confluence in a certain way, and the

Embedding, Extension and Restriction Theorems. The latter three theorems are concerned with the embedding of transformations in a greater context. They are described in more detail further below in Remark 3.2.15.

Moreover, in Definition 3.3.1 on page 45 at the end of this section, we present an alternative solution for the modification of the data type part of AHL-nets, called data-images of AHL-nets. △

Example 3.2.3 (Production for Platform Evolution)**.** In Section 2.2 we presented an example of a platform evolution, where in the platform in Figure 3.2 we replaced the actions *Insert Text* and *Remove Text* by a new action *Modify Copy*. Considering our AHL-net model of the first platform in Example 3.1.9, this means that we want to replace the corresponding transitions *insert* and *remove* with a new transition *modify copy*.

A production that describes this replacement is shown in Figure 3.3. The left-hand side L of the production ϱ contains the two transitions that are to be removed together with the places u and w in their environment. The interface I contains only the two places that are not changed by application of the production. The right-hand side R contains the transition *modify copy* that shall be inserted, connected to the retained environment of the original transitions in the left-hand side. The morphisms $l : I \to L$ and $r : I \to R$ are inclusions. ♢

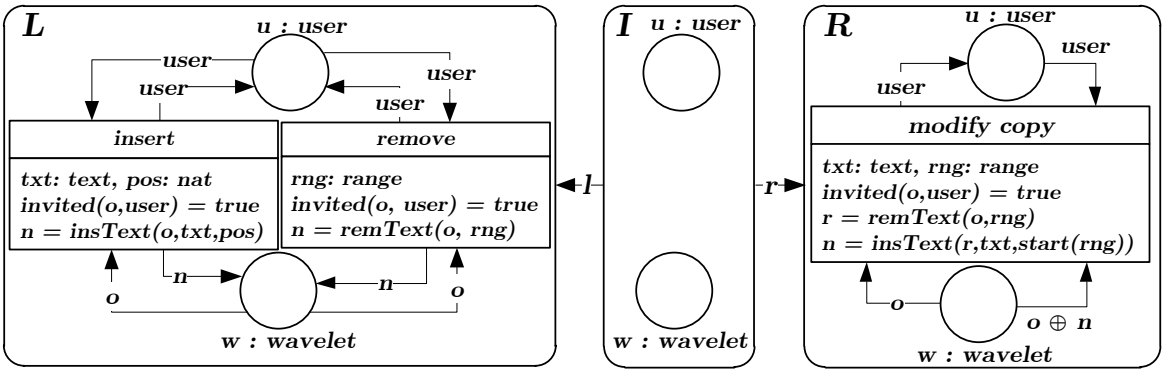


Figure 3.3: AHL-net production ϱ for the evolution of platforms

In order to add the new parts as specified in the right-hand side of a production to an AHL-net we define a gluing construction based on the gluing of its place and transition components in the category **Sets** of sets and functions in the following sense:

Definition 3.2.4 (Gluing of Sets)**.** Given sets A, B and C, and functions $f_1 : A \to B$, $f_2 : A \to C$. The gluing D of B and C along A (or more precisely along f_1 and f_2), written $D = B +_A C$, is defined as the quotient $D = (B \uplus C)/_{\equiv}$ where $\equiv$ is the smallest equivalence relation containing the relation

$$\sim = \{(f_1(a), f_2(a)) \mid a \in A\}.$$

This means that we transitively identify all those elements in $B \uplus C$ which are commonly mapped by the same interface element. Moreover, we obtain functions $g_1 : B \to D$ and $g_2 : C \to D$ with $g_1(b) = [b]_{\equiv}$ for all $b \in B$, and $g_2(c) = [c]_{\equiv}$ for all $c \in C$.

$$\begin{array}{ccc} A & \xrightarrow{f_1} & B \\ {\scriptstyle f_2}\downarrow & \textbf{(PO)} & \downarrow{\scriptstyle g_1} \\ C & \xrightarrow[g_2]{} & D \end{array}$$

△

Fact 3.2.5 (Pushout of Sets)**.** *The diagram (PO) in Definition 3.2.4 is a* pushout diagram *in the category* **Sets** *(see Definition A.1.2 for the definition of pushouts).*

Proof. See Fact 2.17 in [EEPT06b]. □

The gluing of AHL-nets over a given interface can be defined as the componentwise gluing in **Sets** and **Algs**[6]. Due to the fact that the gluing in **Sets** is also a pushout, we obtain also unique induced pre, post, condition and type functions, leading to a well-defined AHL-net.

Definition 3.2.6 (Gluing of AHL-Nets)**.** Given two AHL-net morphisms $f_1 : AN_0 \to AN_1$ and $f_2 : AN_0 \to AN_2$ with $f_1 \in \mathcal{M}_{AHL}$, the *gluing* AN_3 *of* AN_1 *and* AN_2 *along* f_1 *and* f_2, written $AN_3 = AN_1 +_{(AN_0,f_1,f_2)} AN_2$, with $AN_x = (\Sigma_x, P_x, T_x, pre_x, post_x, cond_x, type_x, A_x)$ for $x = 0, 1, 2, 3$ is constructed as follows:

- $T_3 = T_1 +_{T_0} T_2$ with $f'_{1,T}$ and $f'_{2,T}$ as pushout (2) in Figure 3.4 of $f_{1,T}$ and $f_{2,T}$ in **Sets**,
- $P_3 = P_1 +_{P_0} P_2$ with $f'_{1,P}$ and $f'_{2,P}$ as pushout (3) in Figure 3.4 of $f_{1,P}$ and $f_{2,P}$ in **Sets**,
- $(\Sigma_3, A_3) = (\Sigma_1, A_1) +_{(\Sigma_0,A_0)} (\Sigma_2, A_2)$ with $(f'_{1,\Sigma}, f'_{1,A})$ and $(f'_{2,T}, f'_{2,A})$ as the (trivial) pushout (4) in Figure 3.4 of isomorpism $(f_{1,\Sigma}, f_{1,A})$ and morphism $(f_{2,\Sigma}, f_{2,A})$ in **Algs**,
- $pre_3(t) = \begin{cases} (f'^{\#}_{1,\Sigma} \otimes f'_{1,P})^{\oplus} \circ pre_1(t_1) & \text{, if } f'_{1,T}(t_1) = t; \\ (f'^{\#}_{2,\Sigma} \otimes f'_{2,P})^{\oplus} \circ pre_2(t_2) & \text{, if } f'_{2,T}(t_2) = t. \end{cases}$
- $post_3(t) = \begin{cases} (f'^{\#}_{1,\Sigma} \otimes f'_{1,P})^{\oplus} \circ post_1(t_1) & \text{, if } f'_{1,T}(t_1) = t; \\ (f'^{\#}_{2,\Sigma} \otimes f'_{2,P})^{\oplus} \circ post_2(t_2) & \text{, if } f'_{2,T}(t_2) = t. \end{cases}$
- $cond_3(t) = \begin{cases} \mathcal{P}_{fin}(f'_{1,\Sigma}) \circ cond_1(t_1) & \text{, if } f'_{1,T}(t_1) = t; \\ \mathcal{P}_{fin}(f'_{2,\Sigma}) \circ cond_2(t_2) & \text{, if } f'_{2,T}(t_2) = t. \end{cases}$
- $type_3(p) = \begin{cases} f'_{1,S} \circ type_1(p_1) & \text{, if } f'_{1,P}(p_1) = p; \\ f'_{2,S} \circ type_2(p_2) & \text{, if } f'_{2,P}(p_2) = p. \end{cases}$
- $f'_1 = (f'_{1,\Sigma}, f'_{1,P}, f'_{1,T}, f'_{1,A})$ and $f'_2 = (f'_{2,\Sigma}, f'_{2,P}, f'_{2,T}, f'_{2,A})$.

△

Fact 3.2.7 (Pushout of AHL-Nets)**.** *The diagram (1) in Figure 3.4 constructed as in Definition 3.2.6 is a* pushout diagram *in the category* **AHLNets** *(see Definition A.1.2 for the definition of pushouts).*

Proof. In [Pra07] it is shown that the pushouts in **AHLNets** can be constructed componentwise by pushouts (2)-(4) in Figure 3.4 in **Sets** and **Algs**, respectively, using product and general comma category construction. The *pre*, *post*, *cond* and *type* functions are the unique functions induced by the corresponding pushouts in **Sets**. □

[6] for the gluing (amalgamation) of algebras see [EM85]

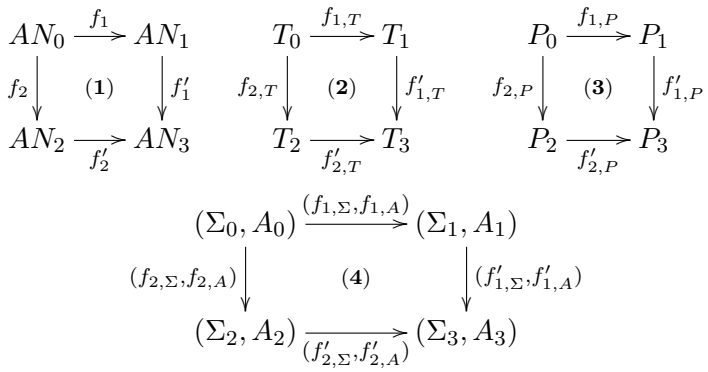


Figure 3.4: Construction of gluing in **AHLNets**

Concept 3.2.8 (Composition of Communication Platforms)**.** In some cases such as the different manifestations of Apache Wave, there may be different platforms with a different set of actions or data, but users on different platforms may still communicate with each other due to a common communication protocol. This means that the modelling of communication platforms should be *compositional*, allowing to view a combination of platforms as a platform.

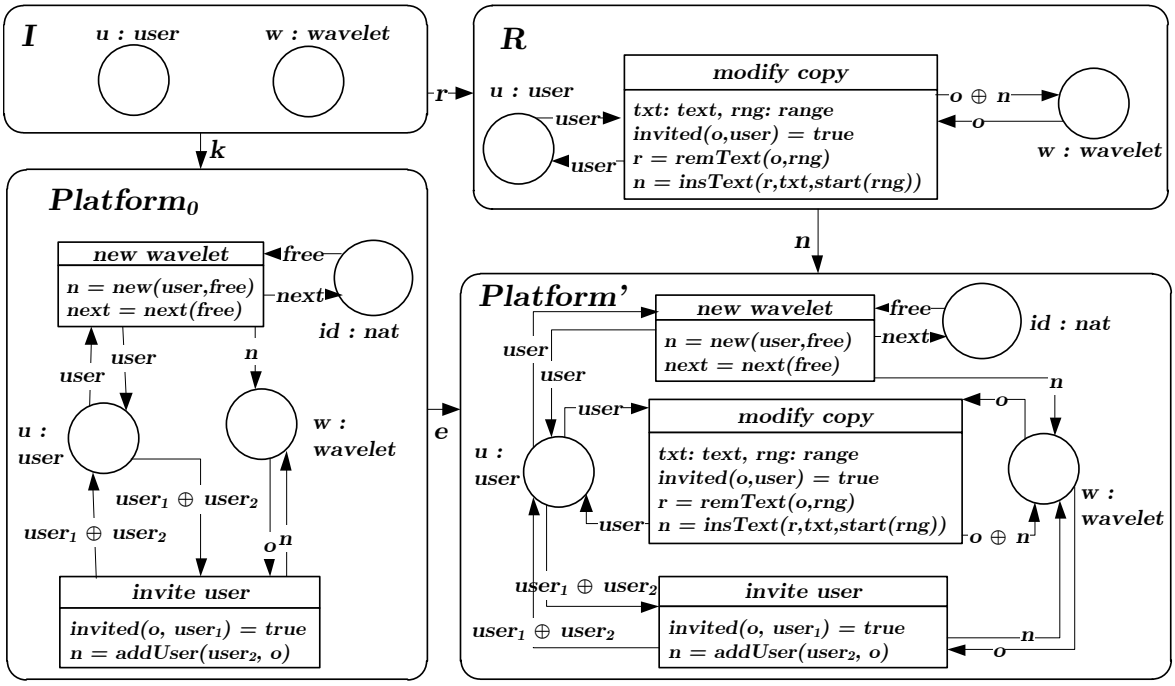
The composition of communication platforms can be modelled by gluing of the corresponding AHL-nets (see Concept 3.1.8). The resulting platform contains all resources and actions from both of the original platforms, where the interface of the gluing specifies the parts of the platforms that are identified in the composed platform. An example of the gluing of Apache Wave platforms is given in the following Example 3.2.9. △

Example 3.2.9 (Evolution of Apache Wave Platform)**.** Figure 3.5a shows the gluing of AHL-nets R and $Platform_0$ over an interface I, where all morphisms r, k, n and e are inclusions. The AHL-net R contains a transition *modify copy*, connected to places u and w that are also contained in the interface I. The AHL-net $Platform_0$ consists of transitions *new wavelet* and *invite user*, both of them connected to the common places u and w, and an additional place *id*, connected to *new wavelet*.

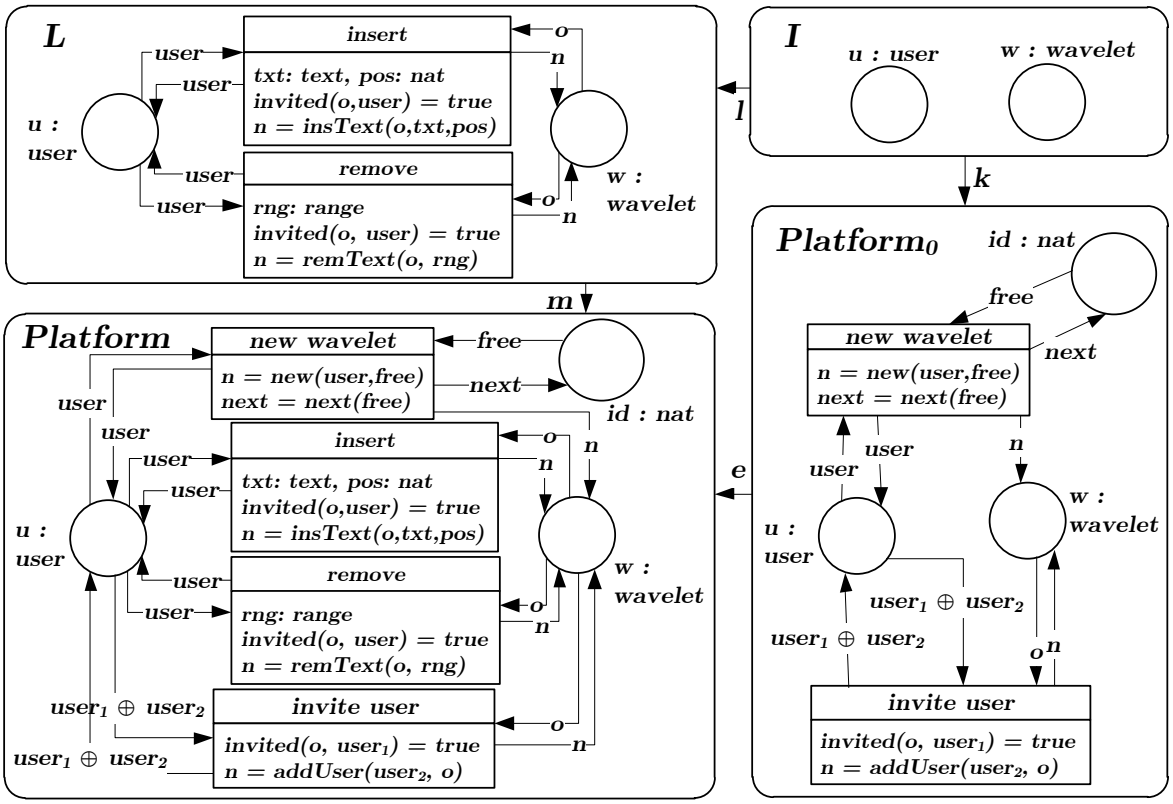
The result of the gluing of R and $Platform_0$ is the AHL-net $Platform'$ in the lower right of Figure 3.5a. The gluing contains all elements that are contained in each of the glued nets, where all the common elements from the interface are identified. Note that in this example the result of the gluing could also be achieved by a componentwise union of the sets of places and transitions. However, in our approach, the gluing does not depend from the naming of the elements in respective nets. This means that we would gain the same result even if for instance the place u in the net R would be called u'. Also, the result of the gluing would contain two copies of the place u, if the two u places would not be matched by a corresponding interface element.

Another example of a gluing of AHL-nets L and $Platform_0$ over the same interface I is depicted in Figure 3.5b. Analogously to the transition *modify copy* in the previous example, in this example the transitions *insert* and *remove* are glued into the same environment. The combination of these two gluings is a direct transformation of the AHL-net *Platform* using the production ϱ in Figure 3.3 as defined in the following, leading to the AHL-net $Platform'$. ♢

The application of a production, called *direct transformation* of AHL-nets, is defined by two gluings of AHL-nets: Given a production $\varrho : L \xleftarrow{l} I \xrightarrow{r} R$ and a match morphism $m : L \to AN$, first, the net AN is decomposed, leading to a context net C such that AN is the gluing of L and C over the interface I. Afterwards, the context net C and the right-hand side R are glued over the interface I, leading to the result AN' of the direct transformation.



(a) Gluing of AHL-nets R and $Platform_0$



(b) Gluing of AHL-nets L and $Platform_0$

Figure 3.5: Gluings of AHL-nets

Definition 3.2.10 (Direct Transformation of AHL-Nets)**.** Given a production $\varrho : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$ and a (match) morphism $m : L \rightarrow AN$ in **AHLNets**. Then a *direct transformation* $AN \overset{(\varrho,m)}{\Rightarrow} AN'$ in **AHLNets** is given by pushouts (1) and (2) in Figure 3.6. A transformation of AHL-nets is a sequence $AN_0 \overset{(\varrho_1,m_1)}{\Rightarrow} AN_1 \cdots \overset{(\varrho_n,m_n)}{\Rightarrow} AN_n$ of direct transformations, written $AN_0 \Rightarrow^* AN_n$. △

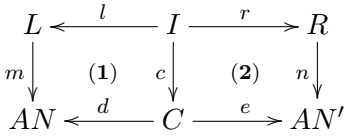


Figure 3.6: Direct transformation of an AHL-net

In order to delete parts of an AHL-net, we need pushout complements. That is, given morphisms $l : I \rightarrow L$ and $m : L \rightarrow G$, an object C together with morphisms g and c as shown in Figure 3.7 is a pushout complement of m and l, if the diagram (1) is a pushout. In the following, we review the construction of pushout complements in **Sets** which is used in the corresponding definition for AHL-nets.

Definition 3.2.11 (Gluing Condition in **Sets**)**.** Let $l : I \rightarrow L$ and $m : L \rightarrow G$ be morphisms in **Sets** with $l \in \mathcal{M}$. We define the set of identification points as $IP = \{x \in L \mid \exists x' \neq x : m(x) = m(x')\}$, and the set of gluing points as $GP = l(I)$. We say that l and f satisfy the gluing condition if $IP \subseteq GP$. △

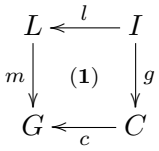


Figure 3.7: Pushout Complement in **Sets**

The following lemma provides a set-theoretical construction of pushout complements in the category **Sets**. The pushout complement C is obtained as a subset of G that consists only of those elements in the image of m that also have a preimage in I.

Lemma 3.2.12 (Pushout Complement in **Sets**)**.** *Let $l : I \rightarrow L$ and $m : L \rightarrow G$ be morphisms in* **Sets**. *There is a pushout complement C of l and m, if l and m satisfy the gluing condition. If a pushout complement exists it can be computed by*

$$C = (G \setminus m(L)) \cup m(l(I))$$

together with inclusion $c : C \hookrightarrow G$ and a morphism $g : I \rightarrow C$ with $g(x) = m(l(x))$ for every $x \in I$ (see Figure 3.7).

Proof. This is shown in Lemma A.6 in [MGE+10]. □

This construction above can be extended to the construction of pushout complements in the category **AHLNets**. The pushout complement C is then constructed by computing the pushout complements of the sets of places and transitions, while all other parts are derived from AHL-net G (see Figure 3.7). For the existence of the pushout complement in **AHLNets**, we need a gluing condition for AHL-nets that is presented in Definition 3.2.13 below.

The following gluing condition is a necessary and sufficient condition for the existence of a direct transformation of AHL-nets (see also [PER95]). In order to satisfy the gluing condition by a production ϱ under a match m some of the places and transitions in the AHL-net AN in the codomain of m must not be deleted by application of the production. The preimages of these elements in the left-hand side of the production are called identification points and dangling points.

The identification points are the preimages of places and transitions which are mapped non-injectively by the match m. The dangling points are the preimages of places which occur in the pre or post conditions of a transition which is matched, and therefore cannot be deleted by application of the production.

Definition 3.2.13 (Gluing Condition for AHL-Nets)**.** Given a production $\varrho : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$ for AHL-nets and an AHL-morphism $m : L \rightarrow AN$ (see Figure 3.8). We define the set of identification points[7]

$$IP = \{x \in P_L \mid \exists x' \neq x : m_P(x) = m_P(x')\} \cup \{x \in T_L \mid \exists x' \neq x : m_T(x) = m_T(x')\}$$

the set of dangling points[8]

$$DP = \{p \in P_L \mid \exists t \in T_{AN} \setminus m_T(T_L), term \in T_\Sigma(X)_{type(p)} : (term, m_P(p)) \leq pre_{AN}(t) \oplus post_{AN}(t)\}$$

and the set of gluing points[9]

$$GP = l_P(P_I) \cup l_T(T_I)$$

We say that ϱ and m satisfy the gluing condition, if $IP \cup DP \subseteq GP$. △

$$\begin{array}{ccccc} L & \xleftarrow{\ l\ } & I & \xrightarrow{\ r\ } & R \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle c} & (2) & \downarrow{\scriptstyle n} \\ AN & \xleftarrow[d]{} & AN_0 & \dashrightarrow & AN' \end{array}$$

Figure 3.8: Gluing condition and direct transformation of AHL-nets

Fact 3.2.14 (Direct Transformation of AHL-Nets)**.** *Given a production for AHL-nets* $\varrho = (L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R)$ *and a match* $m : L \rightarrow AN$ *(see Figure 3.8). The production* ϱ *is applicable on match* m*, i. e. there exists a context AHL-net* AN_0 *in the diagram below, such that (1) is pushout, iff* ϱ *and* m *satisfy the gluing condition in* **AHLNets**. *Then* AN_0 *is called* pushout complement *of* l *and* m. *Moreover, we obtain a unique* AN' *as pushout object of the pushout (2) in* **AHLNets**.

Proof. The fact that our gluing condition is sufficient and necessary for the existence of the pushouts follows from the proofs of Fact 4.13 and Lemma 4.14 in [PER95], where due to a slightly different definition of AHL-morphisms (see Remark 3.1.6) additionally a gluing condition for signatures as well as a dangling condition for the terms is considered. However, these conditions are satisfied since the signature and algebra part of the rule morphisms are isomorphisms. □

[7] i. e. all elements in L that are mapped non-injectively by m
[8] i. e. all places in L that would leave a dangling arc, if deleted
[9] i. e. all elements in L that have a preimage in I

Remark 3.2.15 (Embedding of Transformations). Consider again the modification $Platform \Rightarrow Platform'$ (see Figure 3.5). A realistic model of a communication platform should be much larger, consisting of more different resources and actions. Nonetheless, the models in Figure 2.2 and Figure 2.5 can be considered as a clipping of a larger platform model, containing all relevant parts for the presented modification, and one can think of an embedding $in : Platform \hookrightarrow Platform_{Large}$ of our platform into a larger and more complex platform model. Assuming that this embedding in is *consistent* with the transformation $Platform \Rightarrow Platform'$ (which basically means that the transformation does not modify anything that is needed in the greater context of $Platform_{Large}$), we can use the Embedding Theorem (Theorem 6.14 in [EEPT06b]) to obtain a corresponding modification $Platform_{Large} \Rightarrow Platform'_{Large}$, performing the same replacement of transitions *insert* and *remove* by *modify copy* in the larger model.

Vice versa, given the modification in the larger context, using the Restriction Theorem (Theorem 6.18 in [EEPT06b]), we obtain the corresponding transformation in the smaller context. Consequently, the restriction to relevant parts of the model and the embedding of transformations in the original context can be used to minimise the effort for the rule-based modelling and analysis of AHL-nets. △

In Section A.2 we additionally present a categorical characterization for the existence of direct transformations of AHL-nets, and we show that the categorical gluing condition is in fact equivalent to the set-theoretical gluing condition in Definition 3.2.13.

Concept 3.2.16 (Structural Evolution of Communication Platforms)**.** Changing the structure of a communication platform includes adding, removing or changing resources or actions, as well as changing the correlation between actions and resources. The structural evolution of communication platforms can be modelled as transformation of the corresponding AHL-nets (see Concept 3.1.8). The gluing condition for AHL-nets (see Definition 3.2.13) ensures that the result of the evolution is always well-formed, for instance, by guaranteeing that we do not remove resources that are needed by retained actions. An example of the structural evolution of a communication platform is shown in Example 3.2.17. △

Example 3.2.17 (Evolution of Apache Wave Platform (revisited))**.** The gluings of AHL-nets depicted in Figure 3.5 describe a direct transformation of AHL-nets $Platform \Rightarrow Platform'$ using production ϱ from Example 3.2.3 at inclusion match m. The diagram in Figure 3.5b corresponds to the pushout (1) and the diagram in Figure 3.5a corresponds to pushout (2) in Fact 3.2.14. ♢

3.3 Data Evolution and Abstraction of Platforms

As already pointed out in Remark 3.1.6, with our definition of AHL-morphisms we can use rule-based transformation in the sense of graph transformation to modify the structure of an AHL-net, but not for the data type part. A way to modify the data of a net with respect to a given generalised algebra homomorphism, is presented in the following definition. The data-image of an AHL-net along a generalised homomorphism yields another AHL-net with the same structure, but the algebraic data part is replaced by the codomain of the homomorphism. Moreover, we obtain an AHL-morphism from the original net to the new one.

Definition 3.3.1 (Data-Image of Algebraic High-Level Net)**.** Given an AHL-net $AN_1 = (\Sigma_1, P_1, T_1, pre_1, post_1, cond_1, type_1, A_1)$ and a generalised algebra homomorphism $f = (f_\Sigma, f_A) : (\Sigma_1, A_1) \to (\Sigma_2, A_2)$. We define the *data-image* $\bar{f} : AN_1 \to AN_2$ of AN_1 along f in the following way:

- $AN_2 = (\Sigma_2, P_2, T_2, pre_2, post_2, cond_2, type_2, A_2)$ is an AHL-net with
  - $P_1 = P_2$ and $T_1 = T_2$,
  - $pre_2 = (f_\Sigma^\# \otimes id)^\oplus \circ pre_1$,
  - $post_2 = (f_\Sigma^\# \otimes id)^\oplus \circ post_1$,
  - $cond_2 = \mathcal{P}_{fin}(f_\Sigma^\#) \circ cond_1$, and
  - $type_2 = f_\Sigma \circ type_1$;
- $\bar{f} : AN_1 \to AN_2 = (\bar{f}_\Sigma, \bar{f}_P, \bar{f}_T, \bar{f}_A)$ is an AHL-morphism with $\bar{f}_\Sigma = f_\Sigma$, $\bar{f}_A = f_A$, and $\bar{f}_P$ and $\bar{f}_T$ are identities.

We call $\bar{f} : AN_1 \to AN_2$ a *basic data-image*, if f_Σ is an identity, i. e. if $\bar{f}$ is a data-image along a (non-generalised) Σ_1-homomorphism $f_A : A_1 \to A_2$. △

Well-definedness. The fact that $\bar{f}$ is a well-defined AHL-morphism follows directly from the definitions of AN_2 and $\bar{f}$. □

Concept 3.3.2 (Data Evolution of Communication Platforms)**.** The data type part of a communication platform is modelled by the signature and algebra of the corresponding AHL-net (see Concept 3.1.8). So, since the relation between different data type parts can be expressed by a generalised homomorphism, the evolution of the data type part of a communication platform can be modelled using the data-image construction with the corresponding AHL-net (see Concept 3.1.8) along a generalised algebra homomorphism. The construction basically replaces the previous data type part with the target of the homomorphism. An example of the data evolution of a communication platform is shown in Example 3.3.3. △

Example 3.3.3 (Data Evolution of Communication Platforms)**.** In Example 3.2.9 we changed some actions of our example platform, but the newly introduced action used the same data structures and operations that were already present in the data type part of the original platform. With the construction above, it is also possible to introduce new data structures and corresponding operations, or to change existing ones.

For instance, consider a signature $\Sigma\text{-}Wave'$ that extends the signature $\Sigma\text{-}Wave$ in Table 3.1 by a new operation $removeUser : user\ wavelet \to wavelet$. Moreover, let A' be a $\Sigma\text{-}Wave'$-algebra that coincides with A in all carrier sets and functions, and additionally:

$$exclude_{A'} : A'_{user} \times A'_{wavelet} \to A'_{wavelet}$$
$$(u, (id, uset, t)) \mapsto (id, uset \setminus \{u\}, t)$$

There is an inclusion $in : (\Sigma\text{-}Wave, A) \hookrightarrow (\Sigma\text{-}Wave', A')$, and we can compute the data-image of the AHL-net *Platform* in Figure 3.2 along in, leading to a new AHL-net $Platform'$ with $(\Sigma\text{-}Wave', A')$ as data type part. Furthermore, we obtain a relationship between *Platform* and $Platform'$ in form of an AHL-morphism $\overline{in} : Platform \to Platform'$. Based on this data-evolution, it is then possible to introduce a new *exclude* transition into the platform net, allowing to exclude previously invited users from participation in a wavelet. ♢

Fact 3.3.4 (Compatibility of Structural and Data Evolution)**.** *Given a direct transformation of AHL-nets $AN_1 \overset{o,m}{\Longrightarrow} AN'_1$ and a data-image $f : AN_1 \to AN_2$. Then there exists an AHL-net AN'_2 with direct transformation $AN_2 \overset{o,f\circ m}{\Longrightarrow} AN'_2$ and data-image $f' : AN'_1 \to AN'_2$.*

$$\begin{array}{ccc} AN_1 & \overset{o,m}{\Longrightarrow} & AN'_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f'} \\ AN_2 & \underset{o,f\circ m}{\Longrightarrow} & AN'_2 \end{array}$$

Proof. Given the direct transformation consisting of pushouts (1) and (2) below, and a data-image $f : AN_1 \to AN_2$. By Fact A.6.10 we know that data-images are *Data*-cocreations, and by Fact A.1.35 we know that there is a *Data*-shifting of the span $AN_1 \overset{d_1}{\leftarrow} AN_0 \overset{e_1}{\to} AN_1'$ along f. According to Definition A.1.34, the *Data*-shifting is given by pushouts (3) and (4) below, where k' and f' are *Data*-cocreations and hence data-images. Thus, by composition of pushouts (1)+(3) and (2)+(4), we have a direct transformation $AN_1' \overset{o,f\circ m}{\Longrightarrow} AN_2'$, and data-image $f' : AN_1' \to AN_2'$.

$$\begin{array}{ccccc}
L & \xleftarrow{l} & I & \xrightarrow{r} & R \\
{\scriptstyle m}\downarrow & (1) & {\scriptstyle k}\downarrow & (2) & \downarrow{\scriptstyle n} \\
AN_1 & \xleftarrow{d_1} & AN_0 & \xrightarrow{e_1} & AN_1' \\
{\scriptstyle f}\downarrow & (3) & {\scriptstyle k'}\downarrow & (4) & \downarrow{\scriptstyle f'} \\
AN_2 & \xleftarrow{d_2} & AN_0' & \xrightarrow{e_2} & AN_2'
\end{array}$$

□

Definition 3.3.5 (Basic Data-Preimages of Algebraic High-Level Nets)**.** Given an AHL-net $AN_2 = (\Sigma_2, P_2, T_2, pre_2, post_2, cond_2, type_2, A_2)$ and a Σ_2-homomorphism $f_A : A_1 \to A_2$. We define the *basic data-preimage* $f : AN_1 \to AN_2$ of AN_2 along f_A by $(\Sigma_2, P_2, T_2, pre_2, post_2, cond_2, type_2, A_1)$, and $f = (id_{\Sigma_2}, id_{P_2}, id_{T_2}, f_A)$. △

Corollary 3.3.6 (Basic Data-Preimages and Images)**.** *Every basic data-preimage $f : AN_1 \to AN_2$ of AN_2 along $f_A : A_1 \to A_2$ is also a basic data-image of AN_1 along f_A, and vice versa.*

Proof. This follows directly from the definitions of basic data-images and -preimages, as both constructions do nothing else than replacing the algebra of the AHL-net with the domain respectively codomain of the homomorphism f_A, and the AHL-morphism f consists in both cases of f_A as its algebra part and identities for the other components. □

Remark 3.3.7 (Data-Preimages of Algebraic High-Level Nets). In the case of generalised homomorphisms, the construction of (non-basic) data-preimages requires to check the existence of a well-defined result, similar to the dangling and identification points of the data type part in the gluing condition in [PER95]. This is necessary because the preimage construction may remove parts of the signature that are used as a type, or inside arc inscriptions or firing conditions in the AHL-net. However, in this thesis we do not offer a detailed definition of a non-basic preimage construction. △

Changing the signature part of an AHL-net as in Example 3.3.3 using data-images, we can change the data types and operations that can be used in the AHL-net. Changing the algebra part on the other hand, changes the concrete data values that can occur in the firing behaviour of the net. Considering the fact that the firing behaviour of AHL-nets relies on the evaluation of terms and equations, it is interesting to note that homomorphisms preserve terms and equations (see Theorems 3.3.2 and 4.12.2 in [EM85]). This means that given assignments $v_A : X \to A$ and $v_B : X \to B$ and a homomorphism $f : A \to B$ such that $f \circ v_A = v_B$, by Theorem 3.3.2 in [EM85] we also have $f \circ v_A^* = v_B^*$ (which especially means that $(f_A \circ v_A)^* = f_A \circ v_A^*$), and if (A, v_A) satisfies an equation e, then by Theorem 4.12.2 in [EM85] also $(B, v_B) = (B, f \circ v_A)$ satisfies that equation. Consequently, we obtain that basic data-images preserve firing steps as shown in Corollary A.7.2 on page 227, giving rise to the following conceptual definition of abstraction:

Concept 3.3.8 (Levels of Abstraction of Communication Platforms)**.** In order to identify and investigate specific patterns of interactions, we need a way to model different *levels of*

abstraction. Since we model communication platforms using AHL-nets (see Concept 3.1.8), the actions in a platform are modelled by transitions, and the exact behaviour of an action is determined by the firing behaviour of the corresponding transition (see Definition 3.1.7). Note that both the marking and the consistent transition assignment used in a firing step are highly dependent from the concrete data type part of the AHL-net. So, replacing the data type part with a more abstract one, like for instance a term algebra, results in an abstraction of the firing behaviour.

The data-images of AHL-nets along homomorphisms induce a quasi-order $\triangleleft$ of abstraction respectively concretisation of the data in communication platforms. Given a basic data-image $\bar{f} : AN_1 \to AN_2$ along a Σ-homomorphism $f : A_1 \to A_2$, we can write $AN_1 \triangleleft AN_2$, meaning that AN_1 is an abstraction of AN_2, and AN_2 is a concretisation of AN_1.

The level of abstraction of a communication platform has an impact on the existence of possible scenarios in that platform. On a more abstract level, it can be expected that we may have a lower number of different scenarios than on a more concrete level. This means that abstraction can possibly be used to lower the effort of testing and analysis of a platform, however, with the risk of missing cases that only occur on the more concrete level. This is also discussed in the following example. △

Example 3.3.9 (Levels of Abstraction of Communication Platforms)**.** Consider again our platform model *Platform* in Figure 3.2. Let *Platform*″ be a modification where we replace algebra A with a term algebra $T_{\Sigma\text{-}Wave}(Y)$[10] with $Y_{user} = \{a, b\}$ and $Y_{wavelet} = \{w_0\}$. Since two terms are equal if and only if they are syntactically equal, it is not possible to find a marking such that any of the transitions the AHL-net is enabled, because it is not possible to satisfy the corresponding firing conditions. However, we can formulate *firing schemes* like for instance

$$a \oplus b \oplus w_0 \overset{invite\ user,v}{\longrightarrow} a \oplus b \oplus addUser(b, w_0)$$

where $v : X \to Y$ is a valuation with $v(user_1) = a$, $v(user_2) = b$, $v(o) = w_0$ and $v(n) = addUser(b, w_0)$. This firing scheme can be concretised to different firing steps in the AHL-net *Platform*, considering *Platform* as a concretisation of *Platform*″ along a homomorphism $h_1 : T_{\Sigma\text{-}Wave}(Y) \to A$ that can be given by a valuation of variables $asg : Y \to A$. For example, we can map a to Alice, b to Bob, and w_0 to the empty wavelet with id 0 where only Alice is invited. Then we obtain a concrete firing step

$$\text{Alice} \oplus \text{Bob} \oplus (0, \{\text{Alice}\}, \epsilon) \overset{invite\ user,asg \circ v}{\longrightarrow} \text{Alice} \oplus \text{Bob} \oplus (0, \{\text{Alice}, \text{Bob}\}, \epsilon)$$

where Alice invites Bob. Analogously it is possible with another valuation to concretise the firing scheme to a firing step where Bob invites Alice. This idea can be extended to the abstractions of whole scenarios as demonstrated in Example 4.4.6.

Another rather simple example of an abstraction is given by an AHL-net *Platform*′ with algebra A' which is identical to A, except for the carrier set $A'_{user} = \{Alice, Bob\}$. There is an inclusion $h : A' \hookrightarrow A$ such that *Platform*′ can be considered as an abstraction of *Platform*. This abstraction can be helpful for testing because we restrict the infinite set of users to a quite small set with only two elements, nevertheless with the drawback that we cannot consider cases in this abstraction that occur only with more than two users.

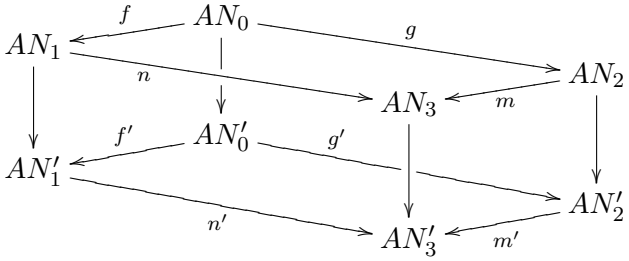
Note that there is also a homomorphism $h_2 : T_{\Sigma\text{-}Wave}(Y) \to A'$ given by valuation $asg' : Y \to A'$ which maps everything exactly as asg. This means that *Platform*″ is also an

[10] Note that the variable family Y must not be confused with the family X of additional variables in $\Sigma = (S, OP; X)$. The term algebra $T_{\Sigma\text{-}Wave}(X)$ is used for the syntax of the AHL-net, whereas the term algebra $T_{\Sigma\text{-}Wave}(Y)$ now specifies the semantics of the net.

abstraction of $Platform'$, and we have a chain $Platform'' \triangleleft Platform' \triangleleft Platform$ of different levels of abstraction. ♢

The following fact states that it is not important on which abstraction level we perform structuring constructions using pushouts and pullbacks, because all results can be shifted or lifted along corresponding data-images. Note that this also includes compatibility of abstraction with direct transformation, since direct transformations are the result of pushout constructions.

Theorem 3.3.10 (Compatibility of Abstraction and Structuring)**.** *Given the commutative cube of AHL-nets below with $f, m \in \mathcal{M}_{AHL}$, where all vertical morphisms are the result of a basic data-image. Then the top of the cube is a pushout (pullback) if and only if the bottom is a pushout (pullback).*



Proof. From Fact A.6.10 we know that all vertical morphisms are *Data*-cocreations which by Corollary A.1.33 implies that all side faces of the cube are pushouts. Assuming that the top of the cube is a pushout, by pushout composition we have that the top and right front face together form a pushout, and thus, the left back and bottom face form a pushout. So by pushout decomposition we obtain that the bottom is a pushout.

Now, let us assume that the bottom is a pushout. Due to componentwise construction of pushouts in **AHLNets**, we have that the P- and T-components of the bottom form pushouts in **Sets**. According to the definition of basic data-images we have identical P- and T-components in the top. Moreover, due to f and m being $\mathcal{M}_{AHL}$-morphisms, they have an isomorphic data type part, and thus also the data type part of the top face form a trivial pushout in **Algs**. Hence, by componentwise construction of pushouts in **AHLNets** we can conclude that the top face is also a pushout.

The proof for pullbacks works dually, using the fact that pushouts along $\mathcal{M}$-morphisms are also pullbacks, and $\mathcal{M}$-morphisms are closed under pushouts. □

3.4 Structure and Semantics of Platforms by Skeleton and Flattening

There are different ways to obtain a low-level place/transition (P/T) net from an algebraic high-level net. The skeleton of an AHL-net "forgets" the data part of the net, leading to a low-level P/T net with the same structure as the AHL-net but, however, not with the same firing behaviour. The flattening construction for AHL-nets also leads to a corresponding P/T net, but instead of forgetting the data part, all high-level information is encoded in the places and transitions. The new contribution of this thesis is that we consider skeletons and flattenings of AHL-nets where we do not fix the data type part to a specific signature or algebra in general.

Definition 3.4.1 (Skeleton)**.** Given an AHL-net $AN = (\Sigma, P, T, pre, post, cond, type, A)$ then the skeleton $Skel(AN)$ is defined by $Skel(AN) = (P, T, pre_S, post_S)$ with

$$pre_S(t) = \sum_{i=1}^{n} p_i \text{ for } pre(t) = \sum_{i=1}^{n} (term_i, p_i)$$

and similar for $post_S : T \to P^{\oplus}$. Given an AHL-morphism $f : AN_1 \to AN_2$ with $f = (f_P, f_T)$ then the skeleton of f is defined by $Skel(f) = (f_P : P_1 \to P_2, f_T : T_1 \to T_2)$.

For the special case that we have a given fixed data type part (Σ, A), we denote the skeleton as $Skel_{(\Sigma,A)} : \mathbf{AHLNets}(\mathbf{\Sigma}, \mathbf{A}) \to \mathbf{PTNets}$. △

Concept 3.4.2 (Structure of Platforms)**.** The skeleton of an AHL-net modelling a communication platform describes the structure of the platform. Accordingly, in a first step of the modelling process of a platform, a low-level P/T net N can abstractly describe the resources and possible actions in the platform. In a second step the P/T net can then be enriched with high-level data types, firing conditions and arc inscriptions, leading to a corresponding algebraic high-level net AN such that $Skel(AN) = N$. △

Example 3.4.3 (Skeleton)**.** The P/T net *LLPlatform* presented in Example 3.1.3 on page 32 is the skeleton of the AHL-net *Platform* in Figure 3.2 on page 35. ♢

The skeleton constructions for AHL-nets and AHL-morphisms form a functor from the category of AHL-nets to the category of P/T nets.

Fact 3.4.4 (Skeleton Functor)**.** *The skeleton construction of AHL-nets and AHL-morphisms defined in Definition 3.4.1 is a functor Skel* : **AHLNets** $\to$ **PTNets** *that maps* $\mathcal{M}_{AHL}$*-morphisms to* $\mathcal{M}_{PT}$*-morphisms and preserves coproducts and also pushouts and pullbacks along* $\mathcal{M}_{AHL}$*.*

Proof-Idea. The fact that the flattening $Skel(f)$ of an AHL-morphism f is a well-defined P/T-morphism follows from the preservation of pre and post domains by morphisms in both categories. Further, since an $\mathcal{M}_{AHL}$-morphism f consists of injective functions f_P and f_T, implying that the skeleton $Skel(f) = (f_P, f_T)$ is an injective $\mathcal{M}_{PT}$-morphism. Finally, the preservation of coproducts, and of pushouts and pullbacks along $\mathcal{M}_{AHL}$ follows from the fact that these constructions can be performed componentwise in **AHLNets** as well as in **PTNets**. For the detailed proof see Section B.1. □

Definition 3.4.5 (Flattening and Weak Flattening of AHL-Nets)**.** Given an AHL-net $AN = (\Sigma, P, T, pre, post, cond, type, A)$, then the *weak flattening* $wFlat(AN)$ is defined by a P/T net $wFlat(AN) = (CP, TA, pre_A, post_A)$ with

- $CP = A \otimes P = \{(a, p) \mid a \in A_{type(p)}, p \in P\}$,
- $TA = \{(t, v) \mid t \in T, v : Var(t) \to A\}$, and
- $pre_A, post_A : TA \to CP^{\oplus}$ are the assigned pre and post domains (see Definition 3.1.7).

Given an AHL-morphism $f : AN_1 \to AN_2$ then the weak flattening of f is given by

$$wFlat(f) = (f_A \otimes f_P : CP_1 \to CP_2, f_C : TA_1 \to TA_2)$$

where

- $f_A \otimes f_P(a, p) = (f_A(a), f_P(p))$, and
- $f_C(t, v) = (f_T(t), f_A \circ v \circ (f_X|_{Var(t)})^{-1})$.

Moreover, the *flattening* $Flat(AN)$ of AN is defined by a P/T net

$$Flat(AN) = (CP, CT, pre_A, post_A)$$

with CP, pre_A and $post_A$ as given above, and

$$CT = \{(t, v) \mid t \in T, v : Var(t) \to A \text{ such that } cond(t) \text{ is valid in } A \text{ under } v\}.$$

Given an AHL-morphism $f : AN_1 \to AN_2$, the flattening of f is defined by the restriction

$$Flat(f) = wFlat(f)|_{Flat(AN_1)} : Flat(AN_1) \to Flat(AN_2).$$

For the special case that we have a given fixed data type part (Σ, A), we denote the flattenings as $Flat_{(\Sigma,A)}, wFlat_{(\Sigma,A)} : \mathbf{AHLNets}(\mathbf{\Sigma}, \mathbf{A}) \to \mathbf{PTNets}$. △

Concept 3.4.6 (Type Conformity and Flat Semantics of Platforms)**.** The weak flattening of an AHL-net modelling a communication platform (see Concept 3.1.8) is a P/T-net, modelling a type conform "black box" semantics of the platform. This means that for every type conform assignment of the variables in an action, however, without regarding the concrete firing conditions, we have a corresponding occurrence of that action in the weak flattening, representing the case that the action is executed using the specified assignment. In contrast, the flattening is a P/T-net that models the actual flat semantics of the platform, taking also into account the firing conditions of each action. This means that the flattening consists only of *realistic* occurrences of action executions that could occur in an actual scenario of interactions in that platform. Note that apart from its flat semantics, an AHL-net consists also of a process semantics as presented in Section 4.1. △

Example 3.4.7 ((Weak) Flattening)**.** In the case of infinitely large carrier sets like in our example algebra in Table 3.2, also the flattening construction yields an infinitely large P/T net. So for our example, we consider the net AN_1 in Figure 3.9, where we restrict the carrier set A_{nat} to the natural numbers between 0 and 3. Also, we assume that the signature consists of an operation $lt : nat\ nat \to bool$ with algebraic semantics $lt_A(x, y) = true$ if x is lower than y, and $false$ otherwise.

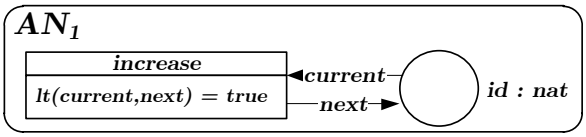


Figure 3.9: AHL-net AN_1

Then with this finite carrier set we also obtain a finite P/T net $Flat(AN_1)$, depicted in Figure 3.10, where we have four places – one for each of the possible natural numbers that can be put as a token on the place id in the AHL-net AN_1. Moreover, the flattened net consists of six transitions, as there are six different consistent assignments for the transition *increase*, given by possible assignments such that the assigned value for *next* is greater than the assigned value for *current*. The pre and post domains of the transitions in $Flat(AN_1)$ are determined by applying the specified assignments to the pre and post domain of the transition *increase*. For instance, due to the pre domain $pre(increase) = (current, id)$ and the assignment $v_1(current) = 0$, the pre domain of transition $(increase, v_1)$ is given by $pre_A(increase, v_1) = (v_1(current), id) = (0, id)$.

Considering the weak flattening, we are not only interested in consistent assignments but in all possible assignments. This means that the weak flattening $wFlat(AN_1)$ additionally contains "reflexive" transitions, where *current* and *next* are assigned to the same value, and correspondingly, there is the same place (x, id) for $x = 0, ..., 3$ in the pre and post domain of each of these transitions. Further, for every transition in $Flat(AN_1)$ there is a corresponding "inverse" transition in $wFlat(AN_1)$ with swapped pre and post domain and assignments. ♢

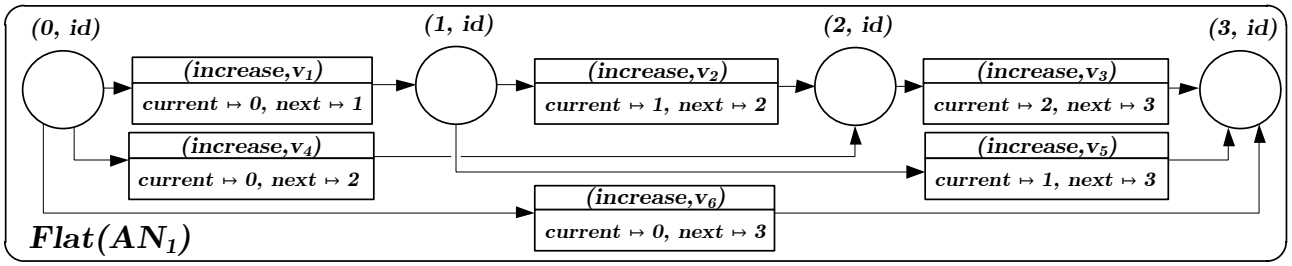


Figure 3.10: Flattening of AHL-net AN_1

Fact 3.4.8 (Natural Inclusion)**.** *There is a natural inclusion $w : Flat \Rightarrow wFlat$, i. e. for every AHL-net AN there is an inclusion $w(AN) : Flat(AN) \hookrightarrow wFlat(AN)$ such that $w : Flat \Rightarrow wFlat$ is a natural transformation.*

Moreover, for a given fixed data type part (Σ, A), the transformation $w_{(\Sigma,A)} : Flat_{(\Sigma,A)} \Rightarrow wFlat_{(\Sigma,A)}$ is cartesian, i. e. for every AHL-morphism $f : AN_1 \to AN_2$ in $\mathbf{AHLNets}(\mathbf{\Sigma}, \mathbf{A})$ the naturality square (1) in Figure 3.11 is a pullback. Since for $f \in \mathcal{M}_{AHL}$ the data type (f_Σ, f_A) is isomorphic, this especially implies that $w : Flat \Rightarrow wFlat$ is $\mathcal{M}$-cartesian, i. e. the diagram (1) in Figure 3.11 is also a pullback for $w : Flat \Rightarrow wFlat$ and $f \in \mathcal{M}_{AHL}$.

$$\begin{array}{ccc} Flat(AN_1) & \xrightarrow{Flat(f)} & Flat(AN_2) \\ {\scriptstyle w(AN_1)}\downarrow & (1) & \downarrow{\scriptstyle w(AN_2)} \\ wFlat(AN_1) & \xrightarrow[wFlat(f)]{} & wFlat(AN_2) \end{array}$$

Figure 3.11: Naturality square of w

Proof Idea. The existence of the inclusion follows directly from the definitions of $Flat$ and $wFlat$. For the fact that diagram (1) is a pullback it basically suffices to show that the T-component of the square is a pullback, since due to similar P-components in $Flat$ and $wFlat$, the P-components trivially form a pullback. For the complete proof, we refer to Section B.2. □

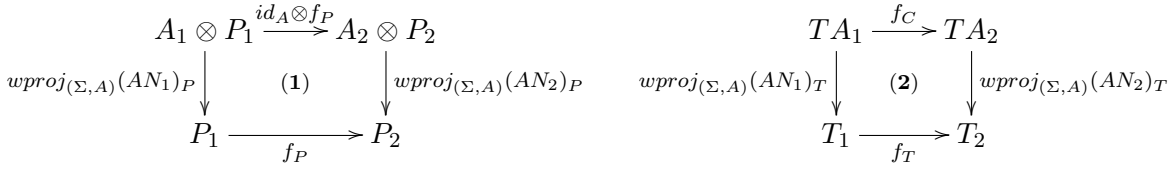
Fact 3.4.9 (Natural Projection)**.** *There is a natural projection $wproj : wFlat \Rightarrow Skel$, i. e. for every AHL-net AN there is a projection $wproj(AN) : wFlat(AN) \to Skel(AN)$ with*

- $wproj(AN)_P(a, p) = p$, *and*
- $wproj(AN)_T(t, v) = t$,

such that $wproj : wFlat \Rightarrow Skel$ is a natural transformation. Moreover, for a given fixed data type part (Σ, A) and projections for components $P, T : \mathbf{PNets} \to \mathbf{Sets}$, the transformations $P(wproj_{(\Sigma,A)}) : P \circ wFlat_{(\Sigma,A)} \Rightarrow P \circ Skel_{(\Sigma,A)}$ and $T(wproj_{(\Sigma,A)}) : T \circ wFlat_{(\Sigma,A)} \Rightarrow T \circ Skel_{(\Sigma,A)}$ are cartesian, i. e. for every AHL-morphism $f : AN_1 \to AN_2$ with fixed data type part (id_Σ, id_A) the diagrams (1) and (2) in Figure 3.12 are pullbacks.

Since for $f \in \mathcal{M}_{AHL}$ the data type (f_Σ, f_A) is isomorphic and $Skel(f) \in \mathcal{M}_{PT}$, this especially implies that $wproj : wFlat \Rightarrow Skel$ is $\mathcal{M}$-cartesian, i. e. for $f \in \mathcal{M}_{AHL}$ the diagram (3) in Figure 3.13 is a pullback in **PTNets**.

Proof Idea. This can be shown by explicitly by verification of the universal property of pullbacks in **Sets** for the diagrams (1) and (2). For the complete proof see Section B.3. □



(a) Pullback of places (b) Pullback of transitions

Figure 3.12: Naturality squares of $P(wproj_{(\Sigma,A)})$ and $T(wproj_{(\Sigma,A)})$

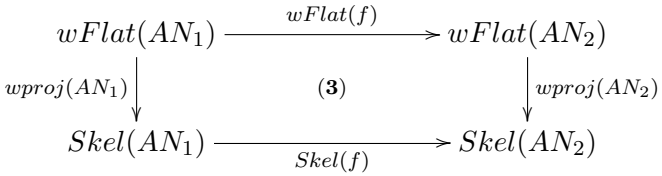
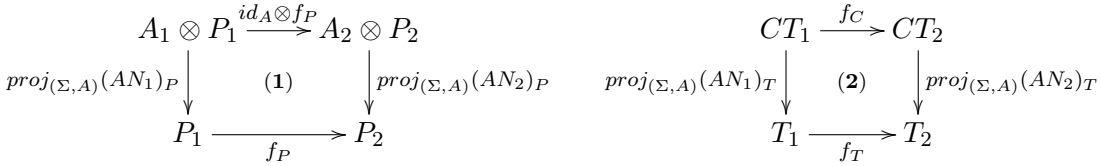


Figure 3.13: Naturality square of $wproj$ with $f \in \mathcal{M}_{AHL}$

Corollary 3.4.10 (Natural Projection)**.** *There is a natural projection $proj : Flat \Rightarrow Skel$ given by $proj = wproj \circ w$ such that for a given data type part (Σ, A) and projections for components P, T : **PNets** $\to$ **Sets**, the transformations $P(proj_{(\Sigma,A)}) : P \circ Flat_{(\Sigma,A)} \Rightarrow P \circ Skel_{(\Sigma,A)}$ and $T(proj_{(\Sigma,A)}) : T \circ Flat_{(\Sigma,A)} \Rightarrow T \circ Skel_{(\Sigma,A)}$ are cartesian, i. e. diagrams (1) and (2) in Figure 3.14 are pullbacks in* **Sets**.



(a) Pullback of places (b) Pullback of transitions

Figure 3.14: Naturality squares of $P(proj_{(\Sigma,A)})$ and $T(proj_{(\Sigma,A)})$

Since for $f \in \mathcal{M}_{AHL}$ the data type (f_Σ, f_A) is isomorphic and $Skel(f) \in \mathcal{M}_{PT}$, this especially implies that $proj : Flat \Rightarrow Skel$ is $\mathcal{M}$-cartesian, i. e. for $f \in \mathcal{M}_{AHL}$ the diagram (3) in Figure 3.15 is a pullback in **PTNets**.

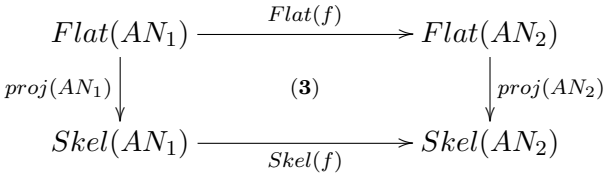


Figure 3.15: Naturality square of $proj$ with $f \in \mathcal{M}_{AHL}$

Proof. The projection $proj$ is obtained as composition of natural transformations $proj = wproj \circ w$ where $wproj$ and w are the natural transformations from Fact 3.4.8 and Fact 3.4.9 such that $P(wproj_{(\Sigma,A)})$, $T(wproj_{(\Sigma,A)})$ and $w_{(\Sigma,A)}$ are cartesian. Since the natural inclusion $w_{(\Sigma,A)}$ consists of injective morphisms as components, and pullbacks along injective morphisms can be constructed componentwise in **PTNets**, by composition of pullbacks we obtain

that also $P(proj_{(\Sigma,A)}) = P(wproj_{(\Sigma,A)} \circ w_{(\Sigma,A)})$ and $T(proj_{(\Sigma,A)}) = T(wproj_{(\Sigma,A)} \circ w_{(\Sigma,A)})$ are cartesian. □

Fact 3.4.11 (Flattening Functors)**.** *The flattening constructions of AHL-nets and AHL-morphisms defined in Definition 3.4.5 are functors* $wFlat, Flat$: **AHLNets** → **PTNets** *that map* $\mathcal{M}_{AHL}$*-morphisms to* $\mathcal{M}_{PT}$*-morphisms and preserve pullbacks along* $\mathcal{M}_{AHL}$*. Moreover, the restricted functors* $wFlat_{(\Sigma,A)}, Flat_{(\Sigma,A)}$: **AHLNets**$(\mathbf{\Sigma}, \mathbf{A})$ → **PTNets** *preserve pushouts along* $\mathcal{M}_{AHL}$*.*

$$\begin{array}{ccc} AN_0 & \xrightarrow{f} & AN_1 \\ {\scriptstyle g}\downarrow & (\mathbf{1}) & \downarrow{\scriptstyle g'} \\ AN_2 & \xrightarrow[f']{} & AN_3 \end{array}$$

(a) Pullback or pushout in **AHLNets**

$$\begin{array}{ccc} Flat(AN_0) & \xrightarrow{Flat(f)} & Flat(AN_1) \\ {\scriptstyle Flat(g)}\downarrow & (\mathbf{2}) & \downarrow{\scriptstyle Flat(g')} \\ Flat(AN_2) & \xrightarrow[Flat(f')]{} & Flat(AN_3) \end{array} \qquad \begin{array}{ccc} wFlat(AN_0) & \xrightarrow{wFlat(f)} & wFlat(AN_1) \\ {\scriptstyle wFlat(g)}\downarrow & (\mathbf{3}) & \downarrow{\scriptstyle wFlat(g')} \\ wFlat(AN_2) & \xrightarrow[wFlat(f')]{} & wFlat(AN_3) \end{array}$$

(b) Pullbacks or pushouts in **PTNets**

Figure 3.16: Preservation of pullbacks and pushouts along $\mathcal{M}_{AHL}$

Proof-Idea. For the well-definedness of the flattening there is a proof in [Ehr05] which is still valid for the more general notion of AHL-morphisms used in this thesis. The preservation of $\mathcal{M}$-morphisms, pushouts and pullbacks can be shown explicitly, considering the exact definitions of $Flat$ and $wFlat$. For the complete proof, we refer to Section B.4. □

Remark 3.4.12 (Counterexample for General Preservation of Pushouts). Note that the functors $Flat$ and $wFlat$ do not preserve pushouts along $\mathcal{M}_{AHL}$ in general, if the data type part of the respective other morphism is not isomorphic. For a counterexample for the functor $Flat$ consider the pushout of AHL-nets in Figure 3.17 where the AHL-nets AN_0 and AN_1 have the data type part (Σ, T_Σ), and the AHL-nets AN_2 and AN_3 have the data type part (Σ, A) where $A_{nat} = \mathbb{N}$, and $lt_A(x, y)$ is *true* if x is lower than y and *false* otherwise. Now, since AN_0 and AN_1 contain no transitions, there are also no consistent transition assignments, i. e. $CT_0 = CT_1 = \emptyset$. Furthermore, we also have that $CT_2 = \emptyset$, because there is no assignment such that the left- and right-hand side of the equation *lt(current,next) = true* become (syntactically) equal. But, in contrast, the set of consistent transition assignments $CT_3 = \{(increase, v) \mid v(current) < v(next)\}$ is not empty. Hence, we do not have a pushout $CT_3 = CT_1 +_{CT_0} CT_2$ in **Sets**, which by componentwise construction of pushouts in **PTNets** implies that we do not have a pushout $Flat(AN_3) = Flat(AN_1) +_{Flat(AN_0)} Flat(AN_2)$ in **PTNets**.

For a counterexample for the functor $wFlat$ we consider the case that AN_0 and AN_1 have a data type part (Σ, B) with $B_{nat} = \{0\}$ and the AHL-nets AN_2 and AN_3 have the data type part (Σ, C) with $C_{nat} = \{0, 1\}$. Also in this case the sets TA_0 and TA_1 of transition assignments of AN_0 and AN_1, respectively, are empty, because they have no transitions. For

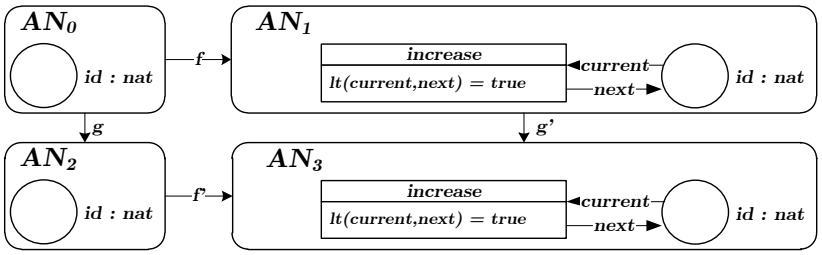


Figure 3.17: Pushout of AHL-nets

the net AN_1 there is one transition assignment mapping *current* and *next* to 0, but for the net AN_3 there are four transition assignments for all combinations of mappings to 0 and 1. Thus, we do not have a pushout $TA_3 = TA_1 +_{TA_0} TA_2$ in **Sets** and therefore no pushout of the corresponding weak flattenings. △

4 Modelling and Evolution of Scenarios

This chapter concerns techniques for the modelling and evolution of interaction scenarios. First, in Section 4.1 we present the modelling technique of AHL-processes based on AHL-process nets, and we show how this technique can be used for the modelling of abstract scenarios.

Then, in Section 4.2 we extend the rule-based transformation of AHL-nets, presented in Section 3.2, to support also the rule-based transformation of AHL-process nets and -processes, allowing the structural evolution of abstract scenarios.

Moreover, the concept of instantiations and instantiated AHL-processes in Section 4.3 is used to model also concrete and semi-concrete scenarios. In the subsequent Section 4.4, we extend the mechanisms for the data evolution and abstraction of AHL-nets in Section 3.3 to corresponding mechanisms for AHL-process nets, -processes and instantiations, yielding also a concept of different levels of abstraction for scenarios.

The restriction and amalgamation techniques for (instantiated) AHL-processes, presented in Section 4.5, support the restriction and union of views on scenarios based on underlying embeddings and composition, respectively, of communication platforms.

Further, in Section 4.6 we present the rule-based transformation of instantiations, and, in combination with the rule-based transformation of AHL-processes, an integrated approach of the rule-based transformation of instantiated AHL-processes. This can be used to model also the structural evolution of concrete and semi-concrete scenarios.

The modelling techniques for histories and user behaviour, presented in Section 4.7, are special cases of the techniques for the modelling respectively evolution of concrete scenarios. Finally, in Section 4.8 we introduce the evolution of scenarios based on communication platform evolutions. This includes the extension of scenarios as well as techniques to evolve abstract and concrete scenarios according to evolutions of the underlying communication platform.

4.1 Modelling of Abstract Scenarios Using Algebraic High-Level Processes

Now, we introduce AHL-process nets based on low-level occurrence nets (see [GR83]) and AHL-processes according to [Ehr05, EHP+02]. The net structure of a high-level occurrence net has similar properties like a low-level occurrence net, but it captures a set of different concurrent computations due to different initial markings. In fact, high-level occurrence nets can be considered to have a set of initial markings for the input places, whereas there is only one implicit initial marking of the input places for low-level occurrence nets.

Moreover, in a low-level occurrence net with an initial marking there is for any complete order of transitions compatible with the causal relation a corresponding firing sequence once there is a token on all input places. This is a consequence of the fact that in an occurrence net the causal relation is finitary. In the case of high-level occurrence nets an initial marking additionally contains data values and in general some of the firing conditions in a complete

order of transitions are not satisfied. Hence, even in the case that the causal relation is finitary, we cannot expect to have complete firing sequences.

In order to ensure a complete firing sequence in a high-level occurrence net there has to be an instantiation of the occurrence net (see [Ehr05] and Section 4.3) In the following definition of AHL-process nets, in contrast to occurrence nets, we omit the requirement that the causal relation has to be finitary, because this is not a meaningful requirement for our application domain.

Definition 4.1.1 (Algebraic High-Level Process Net)**.** An algebraic high-level (AHL-) process net K is an AHL -net

$$K = (\Sigma, P, T, pre, post, cond, type, A)$$

such that for all $t \in T$ with $pre(t) = \sum_{i=1}^{n}(term_i, p_i)$ and notation $\bullet t = \{p_1, \ldots, p_n\}$ and similarly $t\bullet$ we have

1. (*Unarity*): $\bullet t, t\bullet$ are sets rather than multisets for all $t \in T$, i.e. for $\bullet t$ the places $p_1 \ldots p_n$ are pairwise distinct. Hence $|\bullet t| = n$ and the arc from p_i to t has a unary arc-inscription $term_i$.
2. (*No Forward Conflicts*): $\bullet t \cap \bullet t' = \emptyset$ for all $t, t' \in T, t \neq t'$
3. (*No Backward Conflicts*): $t\bullet \cap t'\bullet = \emptyset$ for all $t, t' \in T, t \neq t'$
4. (*Partial Order*): the causal relation $<_K \subseteq (P \times T) \cup (T \times P)$ defined by the transitive closure of $\{(p,t) \in P \times T \mid p \in \bullet t\} \cup \{(t,p) \in T \times P \mid p \in t\bullet\}$ is a strict partial order, i. e. the partial order is irreflexive.

AHL-process nets together with AHL-net morphisms between AHL-process nets form the full subcategory **AHLPNets** $\subseteq$ **AHLNets**. △

Remark 4.1.2 (AHL-occurrence Nets). Note that an AHL-process net with a finitary causal relation is an AHL-occurrence net as defined in [Ehr05], and consequently, also a finite AHL-process net is an AHL-occurrence net. △

We define the sets of input and output places of an AHL-process nets as the sets of places which are not in the post respectively pre domain of a transition:

Definition 4.1.3 (Input and Output Places of AHL-Process Net)**.** Given an AHL-process net K. We define the set $IN(K)$ of input places of K as $IN(K) = \{p \in P_K \mid \nexists t \in T_K : p \in t\bullet\}$, and similar the set $OUT(K)$ of output places of K as $OUT(K) = \{p \in P_K \mid \nexists t \in T_K : p \in \bullet t\}$. △

Corollary 4.1.4 (Data-Image of AHL-Process Net)**.** *For the data-image $f : K_1 \to K_2$ of an AHL-process net K_1 along homomorphism (f_Σ, f_A), we have that K_2 is also an AHL-process net.*

Proof. This follows from the fact that the requirements of AHL-process nets only depend on the structure of the net, and the structure remains unchanged by the data-image construction. □

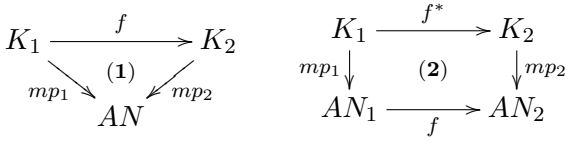
An AHL-process of an AHL-net AN is defined as an AHL-morphism from an AHL-process net K into the net AN. Note that from the preservation of firing behaviour by AHL-morphisms (see Fact 3.1.10) it follows that a firing sequence in K corresponds to a firing sequence in AN. Thus, due to the conflict-free and acyclic structure of AHL-process

nets, an AHL-process mp of an AHL-net AN models a part of the semantics of AN which – up to concurrency and possibly different data values – does not contain any branches or iterations.

Definition 4.1.5 (AHL-Process)**.** An *algebraic high-level (AHL-) process* of an AHL-net AN is an AHL-net morphism $mp : K \to AN$ where K is an AHL-process net. An AHL-process mp is called *strict*, if it has an isomorphic data type part (mp_Σ, mp_A).

The category $\mathbf{Proc}(\mathbf{AN})$ of AHL-processes of an AHL-net AN is defined as the full subcategory of the slice category $\mathbf{AHLNets} \setminus AN$ such that the objects are AHL-processes. This means that the objects of $\mathbf{Proc}(\mathbf{AN})$ are AHL-processes $mp : K \to AN$ and the morphisms of the category are AHL-net morphisms $f : K_1 \to K_2$ such that diagram (1) below commutes.

The category $\mathbf{AHLProcs}$ of all AHL-processes is defined as full subcategory of the arrow category $\mathbf{AHLNets}^{\to}$ such that the objects are AHL-processes, and the morphisms are pairs (f^*, f) of AHL-process net morphisms and AHL-morphisms such that diagram (2) below commutes.



△

Remark 4.1.6 (Strict AHL-Processes and Equivalent Representations). In previous works on AHL-processes, only AHL-nets with fixed data type part (in the category $\mathbf{AHLNets}(\mathbf{\Sigma}, \mathbf{A})$) were considered, instead of the more general category definition of AHL-nets and morphisms in this work. This means that all results regarding the behaviour of AHL-processes [Ehr05] only apply to the notion of strict AHL-processes.

However, applying the data-image construction of AHL-nets (see Definition 3.3.1) to AHL-process nets, every AHL-process $mp : K \to AN$ can be factorized into AHL-processes $k : K \to \bar{K}$ and $\bar{mp} : \bar{K} \to AN$, where $\bar{mp}$ is a strict AHL-process with the same structure as mp. The morphism k is obtained from the data-image, and $\bar{mp}$ is induced by the fact that data-images are *Data*-cocreations (see Fact A.6.10).

Moreover, we regard AHL-processes as equivalent if the factorization above leads to the same result, i. e. if the AHL-processes have the same structure and therefore share the same "normal form" of a strict AHL-process. In this case we have data-images $k : K \to \bar{K}$ and $k' : K' \to \bar{K}$. According to Theorem 3.3.10, we know that any structuring using pushouts or pullbacks can be transferred to a corresponding structuring of the respective other AHL-process.

The only difference between the AHL-processes lies in the different data type part, but since an AHL-process (without additional information) does not specify any concrete data values that are used during the process, the data type part is rather irrelevant at this point of view. However, the concrete data type part of an AHL-process becomes important when we consider instantiated AHL-processes in Section 4.3, where we enrich the AHL-processes with information about concrete data values. △

Concept 4.1.7 (Modelling of Abstract Scenarios)**.** Since we want to investigate the interactions of human-centric communication platforms, we need a way to model these interactions. As the interactions usually are performed by humans, we have to consider different scenarios of interactions on a platform. A scenario contains a number of actions performed by users,

and these actions have to be in compliance with the actions and data that can be used on the corresponding platform. The model of a scenario reflects the causal relation of subsequent actions, but it is also possible to model the *concurrent* occurrence of actions, since usually it is possible that different users perform actions independently at the same time (see also the discussion on page 23 about a tolerant interpretation of simultaneity in near-real time communication).

An abstract scenario of interactions in a communication platform can be modelled as an algebraic high-level process $mp : K \to AN$ of the AHL-net AN corresponding to the communication platform (see Concept 3.1.8). The required morphism mp ensures that the scenario actually conforms to the resources and actions provided by the platform.

The scenario modelled by an AHL-process is an abstract scenario in the sense that we do not specify any concrete data values. Note that in Concept 4.4.5 we introduce a concept of levels of abstraction of scenarios. In that context, the abstract scenarios modelled by AHL-processes can be seen as *top-level* abstract scenarios. An example of an abstract scenario is given in the following Example 4.1.8. △

Example 4.1.8 (Modelling of Abstract Scenario as Algebraic High-Level Process)**.** Consider again the abstract scenario of interactions in the Apache Wave platform presented in Section 2.2, depicted in Figure 2.4. An AHL-process $wave : Wave \to Platform$ modelling this abstract scenario is shown in Figure 4.1. The mappings of the AHL-morphism $wave$ are illustrated by the notation $element : mapping$, e. g. $id_1 : id$ means that place id_1 in AHL-process net $Wave$ is mapped to place id in the AHL-net $Platform$ (see Figure 3.2), and $new_1 : new\ wavelet$ means that the transition new_1 is mapped to the platform transition $new\ wavelet$. We also say that id_1 is an occurrence of id and new_1 is an occurrence of $new\ wavelet$ in the AHL-process $wave$.

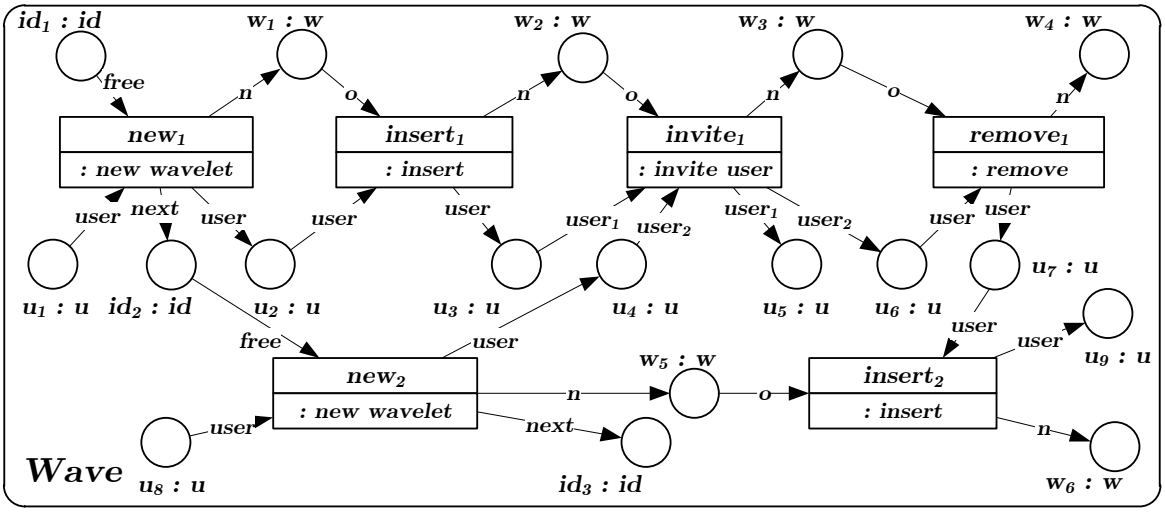


Figure 4.1: AHL-process model of an abstract scenario

Note that, although not visible in the figure, each place of the AHL-process has a type, and the transitions have firing conditions, but these are not explicitly depicted because they can be derived from the corresponding mapping. For instance, the place id_1 has the same type nat as the place id it is mapped to, and the transition new_1 has the same firing conditions as the transition $new\ wavelet$ in the AHL-net $Platform$ in Figure 3.2.

In Example 3.1.9 we introduced the modelling of the Apache Wave platform as AHL-net $Platform$, where resources of the platform are modelled as places, and actions are modelled as

transitions. Accordingly, in our AHL-process model of an abstract scenario, the occurrences of resources and actions in the scenario are represented by corresponding occurrences of places and transitions in the AHL-process. As a result, the causal relation of events is not given as a time line, but instead, it results from the (transitive) connections of transitions and places.

The initial state of the abstract scenario can be seen at the input places $IN(Wave) = \{id_1, u_1, u_8\}$ of the AHL-process net, i. e. initially there is a (free) id and two users. If we assume that there is a suitable marking on each of these three input places, the only transition that could be fired first, is the transition new_1, because all other transitions have at least one place in their pre domain which is not an input place. Further, assuming that the transition new_1 can fire, we obtain a new marking where we have tokens on the places w_1, u_2, id_2 and u_8. In this situation we have two candidates for firing next: the transition $insert_1$ with w_1 and u_2 in its pre domain, and the transition new_2 with id_2 and u_8 in its pre domain. When we proceed with this practice, we obtain that there are two possible sequences of firing transitions (provided that there are suitable data values):

$$\text{Sequence 1} : new_1 \ ; \ insert_1 \ ; \ new_2 \ ; \ invite_1 \ ; \ remove_1 \ ; \ insert_2$$

$$\text{Sequence 2} : new_1 \ ; \ new_2 \ ; \ insert_1 \ ; \ invite_1 \ ; \ remove_1 \ ; \ insert_2$$

It is not necessary to assume the existence of suitable data values, if we consider the low-level *skeleton* of the AHL-process net (see Definition 3.4.1), i. e. if we omit all high-level components of the net. However, the high-level net *Wave* is more expressive than its low-level skeleton. For instance, the transition $invite_1$ has two different user places in its pre domain. Observing the firing conditions of transition *invite user*, we know that the user bound by variable $user_1$ is inviting the user bound by variable $user_2$ and not the other way around. Taking into account only the low-level structure of the net, we do not know if a user on place u_3 is inviting a user on place u_4 or the other way around, while the high-level arc inscriptions tell us that the inviter has to come from place u_3 and the invitee from place u_4.

As one can see, the actions $insert_1$ and new_2 can happen in different order. Considering the truly concurrent behaviour of Petri nets, this means that these transitions could happen at the same time[11], and thus, the possible behaviour modelled by the AHL-process coincides with the abstract scenario in Figure 2.4. ♢

Since every AHL-process conforms to a specific AHL-net, the question arises, what happens to the AHL-process, if we change the data of the corresponding AHL-net, e. g. by abstraction or concretisation. In the following fact we show that AHL-processes are in fact compatible with data evolution and abstraction in the sense that we can always get an AHL-process in correspondence to the changes of the data type. Note that the first and second part of the fact are also applicable to concretisation, since concretisation is a special case of data evolution.

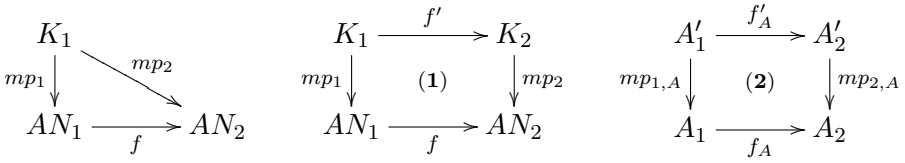
Fact 4.1.9 (Compatibility of AHL-Processes with Data Evolution and Abstraction)**.**

(Data Evolution) Given an AHL-process $mp_1 : K_1 \to AN_1$ *and a data-image* $f : AN_1 \to AN_2$ *of* AN_1 *along* $(f_\Sigma, f_A) : (\Sigma_1, A_1) \to (\Sigma_2, A_2)$*, then there exists an AHL-process* $mp_2 : K_1 \to AN_2$*, called* extension *of* mp_1 *along* f*, with* $mp_2 = f \circ mp_1$*.*

(Strict Data Evolution) Given a strict AHL-process $mp_1 : K_1 \to AN_1$ *and a data-image* $f : AN_1 \to AN_2$ *of* AN_1 *along* $(f_\Sigma, f_A) : (\Sigma_1, A_1) \to (\Sigma_2, A_2)$*, then there exists a strict AHL-process* $mp_2 : K_2 \to AN_2$ *with* $f' : K_1 \to K_2$ *being the data-image of* K_1 *along* (f_Σ, f_A) *and diagram (1) in Figure 4.2b is a pushout.*

[11] Technically this can be achieved by firing of a parallel transition $insert_2 + new_2$.

(Abstraction) Given an AHL-process $mp_2 : K_2 \to AN_2$ and a basic data-preimage $f : AN_1 \to AN_2$ of AN_2 along a Σ-homomorphism $f_A : A_1 \to A_2$, then there exists an AHL-process $mp_1 : K_1 \to AN_1$ such that diagram (1) in Figure 4.2b commutes, where $f' : K_1 \to K_2$ is a basic data-preimage of K_2 along $f'_A : A'_1 \to A'_2$, obtained by pullback (2) in Figure 4.2c in $\mathbf{Alg}(\mathbf{\Sigma})$.



(a) Compatibility with data evolution

(b) Compatibility with abstraction

(c) Pullback of Σ-algebras

Figure 4.2: Compatibility of AHL-processes with data evolution and abstraction

Proof.

1. This follows directly from the fact that also the composition $f \circ mp_1 : K_1 \to AN_2$ is an AHL-process, because K_1 is an AHL-process net.

2. Given strict AHL-process $mp_1 : K_1 \to AN_1$, we can w. l. o. g. assume that K_1 and AN_1 have the same data type part, allowing us to construct the data-image $f' : K_1 \to AN_1$ along (f_Σ, f_A). By Corollary 4.1.4 we have that K_2 is an AHL-process net. From Fact A.6.10 we know that data-image f' is a data-cocreation, and therefore by Corollary A.1.33 the trivial pushout along (f_Σ, f_A) and identities in **Algs** induces a unique $mp_2 : K_2 \to AN_2$ such that diagram (1) in Figure 4.2b becomes a pushout.

3. Let $mp_2 : K_2 \to AN_2$ be an AHL-process and $f : AN_1 \to AN_2$ of AN_2 be a basic data-preimage along Σ-homomorphism $f_A : A_1 \to A_2$. Since $\mathbf{Alg}(\mathbf{\Sigma})$ is complete, we can construct the pullback (2) in Figure 4.2c. Using Definition 3.3.5, we construct the data-preimage $f' : K_1 \to K_2$ of K_2 along $f' : A'_1 \to A'_2$. Note that the P-, T- and Σ-components of K_1 and AN_1 are identical with the P-, T- and Σ-components in K_2 and AN_2, respectively. Therefore, we obtain a well-defined AHL-morphism $mp_1 : K_1 \to AN_1$ as $mp_1 = (mp_{2,\Sigma}, mp_{2,P}, mp_{2,T}, mp_{1,A})$ such that diagram (1) in Figure 4.2b commutes, due to pullbacks in the underlying categories.

□

The question of the compatibility of AHL-processes with structural changes of the corresponding AHL-nets is more complicated and is therefore discussed separately in Section 4.5 and Section 4.8.

4.2 Evolution of Abstract Scenarios

Now, we extend our framework by the gluing and transformation of AHL-process nets and AHL-processes. For this purpose we define productions for AHL-process nets where the left and right hand side and the interface of the production are AHL-process nets.

Definition 4.2.1 (Production for AHL-Process Nets)**.** A *production for AHL-process nets* $\varrho : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$ is a span of **AHLPNets**-morphisms $l : I \to L, r : I \to R \in \mathcal{M}_{AHL}$. △

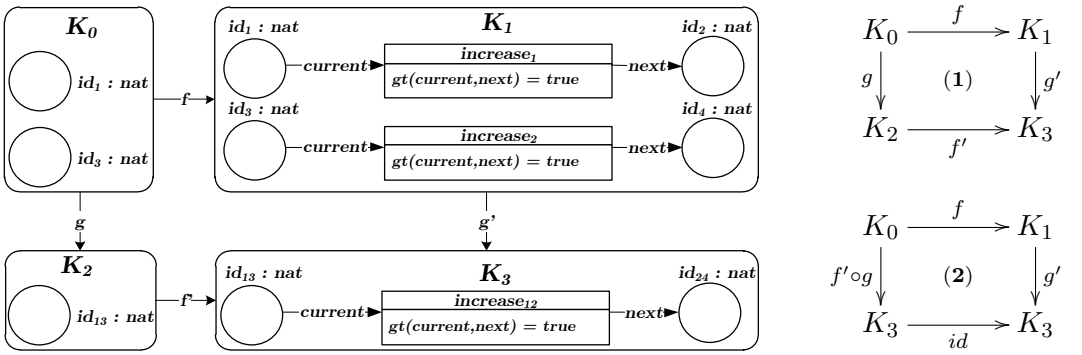
Definition 4.2.2 (Gluing and Transformation of AHL-Process Nets)**.** Given AHL-process nets K_0, K_1 and K_2 together with morphisms $f : K_0 \to K_1$ and $g : K_0 \to K_2$. Then the gluing of AHL-nets $K_3 = K_1 +_{K_0,f,g} K_2$ is called *gluing of AHL-process nets* if K_3 is an AHL-process net.

Further, given an AHL-net K and a production for AHL-process nets $\varrho : L \xleftarrow{l} I \xrightarrow{r} R$ together with a match $m : L \to K$. Then, a direct transformation of AHL-nets $K \overset{\varrho,m}{\Longrightarrow} K'$ with pushouts (1) and (2) below is called *transformation of AHL-process nets* if (1) and (2) are gluings of AHL-process nets.

$$\begin{array}{ccccc} L & \xleftarrow{\ l\ } & I & \xrightarrow{\ r\ } & R \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle c} & (2) & \downarrow{\scriptstyle n} \\ K & \xleftarrow[\ d\ ]{} & C & \xrightarrow[\ e\ ]{} & K' \end{array}$$

△

Remark 4.2.3 (Gluing and Transformation of AHL-Process Nets). Note that every pushout in the category **AHLNets** is a gluing of AHL-nets, whereas we consider a pushout in the category **AHLPNets** only as a gluing of AHL-process nets if it is also a gluing of AHL-nets. To understand the reason for this decision, consider the diagram in Figure 4.3a, where the mappings are indicated by indices of the elements in the depicted AHL-process nets.



(a) Pushout (1)

(b) Pushouts (1) and (2)

Figure 4.3: Pushouts in **AHLPNets** but not in **AHLNets**

The diagram is not a gluing of AHL-process nets, because it is not a pushout in **AHLNets**, since according to the interface only the places p_1 and p_2 have to be identified. Instead, also the transitions and the places p_2 and p_4 are identified. Nonetheless, the diagram is a pushout in the category **AHLPNets**. The diagram commutes, and for every AHL-process net K_3' with morphisms $h_1 : K_1 \to K_3'$ and $h_2 : K_2 \to K_3'$ such that $h_1 \circ f = h_2 \circ g$ there is a single place $p = h_1(id_1) = h_2(id_{13}) = h_1(id_2)$ in K_3', and since K_3' does not have any forward conflicts, there has to be a single transition $t = h_1(increase_1) = h_1(increase_2)$ with p in its pre domain, and with a single place $h_1(id_2) = h_1(id_4)$ in its post domain since K_3' also does not have backward conflicts. Therefore, there exists a unique morphism $h : K_3 \to K_3'$ satisfying the universal pushout property.

So considering all pushouts of AHL-process nets would be quite unintuitive since for AHL-process nets being special kinds of AHL-nets, one would expect that the gluing of AHL-process nets is the same as gluing the respective AHL-nets.

Regarding the transformation of AHL-process nets there is another technical reason for our decision to consider only gluings of AHL-nets. From the pushout in Figure 4.3a it follows that $K_0 \xrightarrow{g} K_2 \xrightarrow{f'} K_3$ is a pushout complement of $K_0 \xrightarrow{f} K_1 \xrightarrow{g'} K_3$. Using a similar

argumentation as above, we obtain that also diagram (2) in Figure 4.3b is a pushout in **AHLPNets**.

The diagram commutes and for every AHL-process net K_3'' with morphisms $k_1 : K_1 \to K_3''$ and $k_2 : K_3 \to K_3''$ such that $k_1 \circ f = k_2 \circ f' \circ g$ there is a single place $p = k_1(id_1) = k_2(id_{13}) = k_1(id_2)$ in K_3''. So, assuming that there are two different transitions $k_1(increase_1)$ and $k_2(increase_2)$ would mean that there is a forward conflict in p which contradicts the fact that K_3'' is an AHL-process net. Thus, we have $k_1(increase_1) = k_2(increase_2)$. Then by the fact that AHL-morphisms preserve pre and post conditions, we obtain that there is a single place $p' \leq post_{K_3''}(increase_1) = post_{K_3''}(increase_2)$ such that $k_1(id_2) = p'$ and $k_1(id_4) = p'$. Hence, any comparison object K_3'' contains a substructure isomorphic to K_3, allowing to define a unique morphism $k : K_3 \to K_3''$, embedding K_3 into that structure, and satisfying the universal pushout property.

This implies that pushout complements in **AHLPNets** along $\mathcal{M}_{AHL}$ are not unique since we have $f \in \mathcal{M}_{AHL}$. In contrast, from Fact A.1.11 we know that $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ is an $\mathcal{M}$-adhesive category and therefore has unique pushout complements (see [EEPT06b]). So with our decision to consider only gluings of AHL-nets for the transformation of AHL-process nets, we ensure that we also have unique transformation results for AHL-process nets. △

The gluing of AHL-nets may produce forward or backward conflicts as well as cycles in the causal relation. So for the gluing of two AHL-process nets via pushout construction the AHL-process nets have to be *composable* in order to obtain again an AHL-process net as a result of the gluing. Composability of AHL-process nets with respect to an interface means that the result of the gluing does not violate the process net properties in Definition 4.1.1.

A span of **AHLPNets**-morphisms $i_1 : I \to K_1$ and $i_2 : I \to K_2$ induces a causal relation between the elements of the interface I. This relation consists of the causal relation between elements in K_1 and K_2 and additionally between those elements in both of the AHL-process nets which is obtained by gluing over the interface.

Definition 4.2.4 (Induced Causal Relation)**.** Given three AHL-process nets I, K_1 and K_2, and two AHL-net morphisms $i_1 : I \to K_1$ and $i_2 : I \to K_2$. The *induced causal relation* $<_{(i_1,i_2)}$ is defined as the transitive closure of the relation $\prec_{(i_1,i_2)}$ defined by

$$\prec_{(i_1,i_2)} = \{(x,y) \in (P_I \uplus T_I) \times (P_I \uplus T_I) \mid i_1(x) <_{K_1} i_1(y) \text{ or } i_2(x) <_{K_2} i_2(y)\}.$$

△

Definition 4.2.5 (Composability of AHL-Process Nets)**.** Given three AHL-process nets I, K_1 and K_2, and two AHL-net morphisms $i_1 : I \to K_1$ and $i_2 : I \to K_2$ with $i_1 \in \mathcal{M}_{AHL}$. Then (K_1, K_2) *are composable w. r. t.* (I, i_1, i_2) if

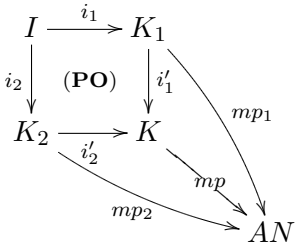
1. *(No Cycles)* the induced causal relation $<_{(i_1,i_2)}$ is a strict partial order,
2. *(Non-Injective Gluing)*
   - for all $p_1 \neq p_2 \in IN(I)$ with $i_2(p_1) = i_2(p_2)$ there is $i_1(p_1) \in IN(K_1)$ or $i_1(p_2) \in IN(K_1)$,
   - for all $p_1 \neq p_2 \in OUT(I)$ with $i_2(p_1) = i_2(p_2)$: there is $i_1(p_1) \in OUT(K_1)$ or $i_1(p_2) \in OUT(K_1)$, and
3. *(No Conflicts)*
   - for all $p \in IN(I) : i_1(p) \notin IN(K_1) \Rightarrow i_2(p) \in IN(K_2)$,
   - for all $p \in OUT(I) : i_1(p) \notin OUT(K_1) \Rightarrow i_2(p) \in OUT(K_2)$.

△

The composability of AHL-process nets is a sufficient and necessary condition for the existence of the gluing of AHL-process nets along $\mathcal{M}_{AHL}$-morphisms.

Fact 4.2.6 (Gluing of AHL-Process Nets)**.** *Given AHL-process nets I, K_1, K_2 and AHL-net morphisms $i_1 : I \to K_1$ and $i_2 : I \to K_2$ where $i_1 \in \mathcal{M}_{AHL}$. Then there exists the gluing K of K_1 and K_2 along i_1 and i_2, written $K = K_1 +_{(I,i_1,i_2)} K_2$, if and only if (K_1, K_2) are composable w. r. t. (I, i_1, i_2).*

Extension to Processes. *In order to extend this gluing construction for AHL-processes in the category* **Proc**(**AN**) *(see Definition 4.1.5) one additionally requires AHL-morphisms $mp_1 : K_1 \to AN$ and $mp_2 : K_2 \to AN$ with $mp_1 \circ i_1 = mp_2 \circ i_2$. The pushout (PO) in* **AHLNets** *then provides a unique morphism $mp : K \to AN$ such that (PO) is also a pushout in* **Proc**(**AN**)*.*



Proof-Idea. In order to show that the diagram (PO) constructed as pushout in **AHLNets** is also a pushout in the full subcategory **AHLPNets** $\subseteq$ **AHLNets** it suffices to show that the pushout object K is an AHL-process net. The fact that the gluing does not produce conflicts or cycles is ensured by the corresponding items 1 and 3 of the required composability of K_1 and K_2. Furthermore, item 2 ensures that there are no conflicts or violations of the unarity condition created by non-injective gluing.

The other way around it can be shown that the conditions of the composability can be derived from the fact that K satisfies the requirements of an AHL-process net. For a detailed proof see Section B.5. □

Concept 4.2.7 (Composition of Abstract Scenarios)**.** For different scenarios it may be desirable to combine them, in order to obtain a new integrated scenario, where the single subcomponents are run parallel, sequentially, or that they are synchronised at some points and run otherwise in parallel. The composition of abstract scenarios can be modelled as gluing of the corresponding ahl-processes, where the interface defines all synchronisation points of the composition. If the interface is empty, the gluing leads to a new scenario, where the original scenarios run completely parallel. An example of the composition of two abstract scenarios is given in the following Example 4.2.8 △

Example 4.2.8 (Composition of Abstract Scenarios)**.** The composition of two scenarios $wave_1 : Wave_1 \to Platform$ and $wave_2 : Wave_2 \to Platform$ is shown in Figure 4.4. The interface of the gluing contains three places id_2, u_4 and u_7 and the AHL-morphisms $i_1 : I \to Wave_1$ and $i_2 : I \to Wave_2$ are inclusions.

In the result $wave : Wave \to Platform$ of the gluing, the two scenarios are "glued" together at the specified interface places. Consequently, the new scenario consists of both of the scenarios which are synchronised at the interface places. ♢

We define a gluing relation for the transformation of AHL-process nets which is induced by a production ϱ for AHL-process nets and a match m. The gluing relation is a relation between the interface elements of ϱ which consists of the causal relation between elements

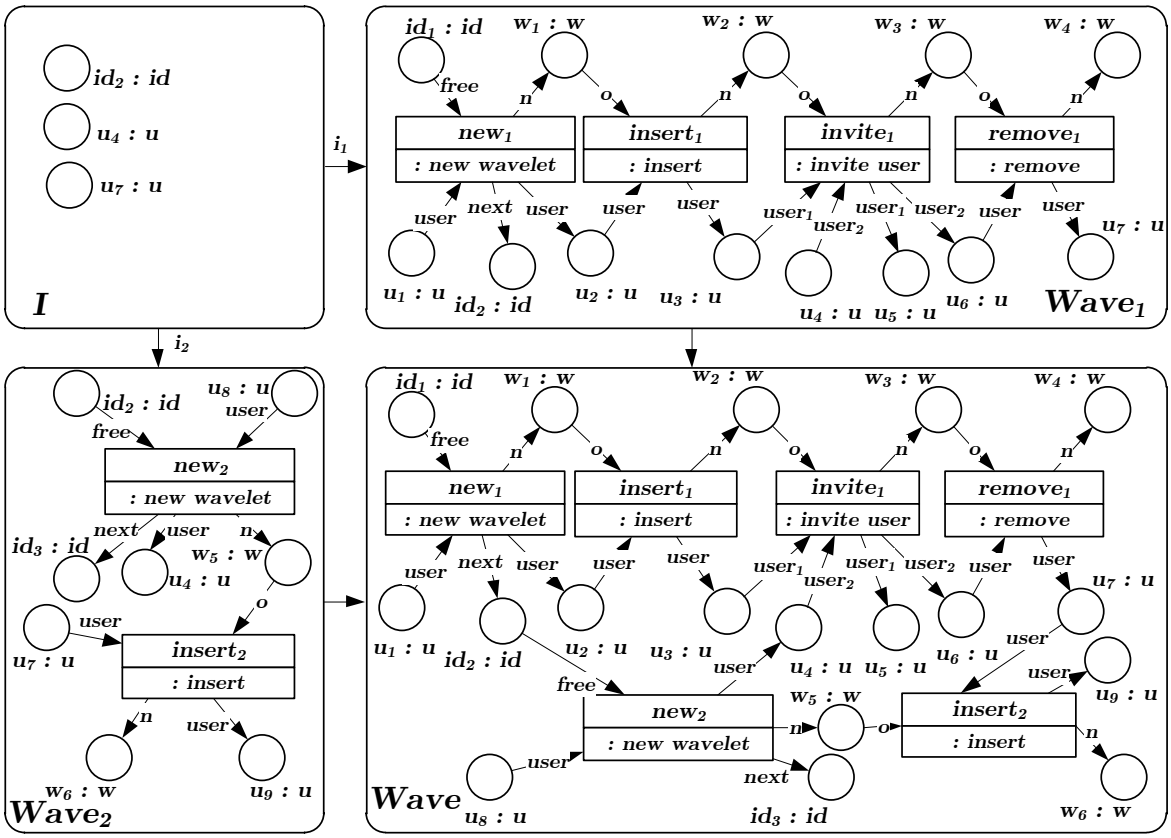


Figure 4.4: Gluing of AHL-processes

in the codomain of m that are preserved by application of ϱ and the causal relation of the right hand side of the production, and additionally it consists of the causal relations that are obtained by gluing over the interface.

Definition 4.2.9 (Gluing Relation for Transformations)**.** Given a production for AHL-process nets $\varrho : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$ and a match $m : L \to K$ we define the relations

$$\prec_{(K,m)} = \{(x,y) \in (P_K \times (T_K \setminus m_T(T_L))) \uplus ((T_K \setminus m_T(T_L)) \times P_K) \mid x \in {\bullet y}\}$$

and $<_{(K,m)}$ as the transitive closure of $\prec_{(K,m)}$. Furthermore we define

$$\prec_{(\varrho,m)} = \{(x,y) \in (P_I \times T_I) \uplus (T_I \times P_I) \mid m \circ l(x) <_{(K,m)} m \circ l(y) \vee r(x) <_R r(y)\}$$

The transitive closure $<_{(\varrho,m)}$ of $\prec_{(\varrho,m)}$ is called *gluing relation* of production ϱ under match m. △

For the transformation of AHL-process nets we define a *transformation condition* which is a necessary and sufficient condition that the direct transformation of an AHL-process nets exists. The satisfaction of the transformation condition by a production ϱ and a match m requires that the gluing condition for AHL-nets (see Definition 3.2.13) is satisfied. Moreover, it requires that the gluing condition is irreflexive and that the application of the production does neither produce any conflicts nor violates the unarity condition of AHL-process nets.

Definition 4.2.10 (Transformation Condition for AHL-Process Nets)**.** Given a production for AHL-process nets $\varrho : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$ and an AHL-process net K. Then ϱ satisfies the transformation condition under a (match) morphism $m : L \to K$ if

1. *(Gluing Condition)* the gluing condition is satisfied (see Definition 3.2.13),
2. *(No Cycles)* the gluing relation $<_{(\varrho,m)}$ of ϱ under m is a strict partial order,
3. *(Non-Injective Gluing)*
   - for all $p_1 \neq p_2 \in IN(I)$ with $m \circ l(p_1) = m \circ l(p_2)$ we have $r(p_1) \in IN(R)$ or $r(p_2) \in IN(R)$,
   - for all $p_1 \neq p_2 \in OUT(I)$ with $m \circ l(p_1) = m \circ l(p_2)$ we have $r(p_1) \in OUT(R)$ or $r(p_2) \in OUT(R)$,
4. *(No Conflicts)* for the sets of in and out places of the match

$$InP = \{x \in IN(I) \mid l(x) \in IN(L) \text{ and } m \circ l(x) \notin IN(K)\}, \text{ and}$$

$$OutP = \{x \in OUT(I) \mid l(x) \in OUT(L) \text{ and } m \circ l(x) \notin OUT(K)\}$$

there is

$$r(InP) \subseteq IN(R) \text{ and } r(OutP) \subseteq OUT(R).$$

△

Theorem 4.2.11 (Direct Transformation of AHL-Process Nets)**.** *Given a production for AHL-process nets $\varrho : L \xleftarrow{l} I \xrightarrow{r} R$ and an AHL-process net K together with a morphism $m : L \to K$. Then the direct transformation of AHL-process nets with pushouts (1) and (2) in* **AHLPNets** *exists if and only if ϱ satisfies the transformation condition for AHL-process nets under m.*

Extension to processes. *In order to extend this construction for AHL-processes in the category* **Proc**(**AN**) *one additionally requires AHL-morphisms $mp : K \to AN$ and $rp : R \to AN$ with $mp \circ m \circ l = rp \circ r$. Then by composition of AHL-morphisms we obtain an AHL-process $cp = mp \circ d : C \to AN$ and the pushout (1) in* **AHLPNets** *is also a pushout of $mp \circ m$ and cp in* **Proc**(**AN**)*. Moreover, the pushout (2) in* **AHLNets** *provides a unique morphism $mp' : K' \to AN$ such that mp' is pushout of cp and rp in* **Proc**(**AN**) *according to Fact 4.2.6.*

$$\begin{array}{ccccc} L & \xleftarrow{l} & I & \xrightarrow{r} & R \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle c} & (2) & \downarrow{\scriptstyle n} \\ K & \xleftarrow[d]{} & C & \xrightarrow[e]{} & K' \end{array}$$

Proof-Idea. Satisfaction of the transformation condition for AHL-process nets means that the gluing condition for AHL-nets is satsfied which by Fact 3.2.14 implies that pushouts (1) and (2) can be constructed in **AHLNets**. Due to Lemma A.7.1 the process net properties in Definition 4.1.5 are reflected by injective AHL-morphisms, implying that C is an AHL-process net and (1) is also a pushout in **AHLPNets**. Finally, it can be shown that the satisfaction of the transformation condition implies that C and R are composable w. r. t. (I, c, r), i. e. the pushout (2) in **AHLNets** is also a pushout in **AHLPNets**.

Vice versa, given pushouts (1) and (2) in **AHLNets**, we have that the gluing condition for AHL-nets is satisfied because it is sufficient for transformation of AHL-nets. The satisfaction of the rest of the transformation condition can be obtained by composability of C and R w. r. t. (I, c, r) by pushout (2) in **AHLPNets**, and the construction of pushout complement C. For a detailed proof see Section B.6. □

Remark 4.2.12 (Direct Transformation of AHL-Processes). Note that the extension of the direct transformation defined in Theorem 4.2.11 to AHL-processes requiring an AHL-process $rp : R \to AN$ is only the required minimum. Alternatively it is possible to equip also the other parts of the production with corresponding morphisms $lp : L \to AN$ and $ip : I \to AN$ such that $lp \circ l = ip$ and $rp \circ r = ip$. The production is then called a *production for AHL-processes of AHL-net AN*. For a match $m : L \to K$ and AHL-process $mp : K \to AN$, there is then a corresponding direct transformation $mp \Rightarrow mp'$ in $\mathbf{Proc}(\mathbf{AN})$ if the transformation condition is satisfied and we have $mp \circ m = lp$. △

Concept 4.2.13 (Structural Evolution of Abstract Scenarios)**.** In Concept 3.2.16 we discussed the structural evolution of communication platforms that can be modelled using rule-based transformation of the corresponding AHL-nets. As the structural evolution of communication platforms involves adding, removing or changing resources and actions, accordingly it is possible that a scenario evolves by adding, removing or changing occurrences of the resources and actions in the scenario. The structural evolution of abstract scenarios of interactions in a communication platform can be modelled using rule-based transformation of the corresponding AHL-process (see Concept 4.1.7). An example of the structural evolution of an abstract scenario is given in Example 4.2.14, where we change the order of two action occurrences in our abstract scenario presented in Example 4.1.8. △

Example 4.2.14 (Evolution of Abstract Scenario as Transformation of AHL-Process)**.** A production ϱ_1 for AHL-processes of the AHL-net *Platform* in Figure 3.2 is shown in Figure 4.5. The production takes two occurrences of actions *insert* and *invite* and swaps their positions in the way that the user who originally inserted some text now invites another user and vice versa. There is only one place that is removed, namely w_2 in L_1, and one place that is newly inserted, namely w_2'' in R_1.

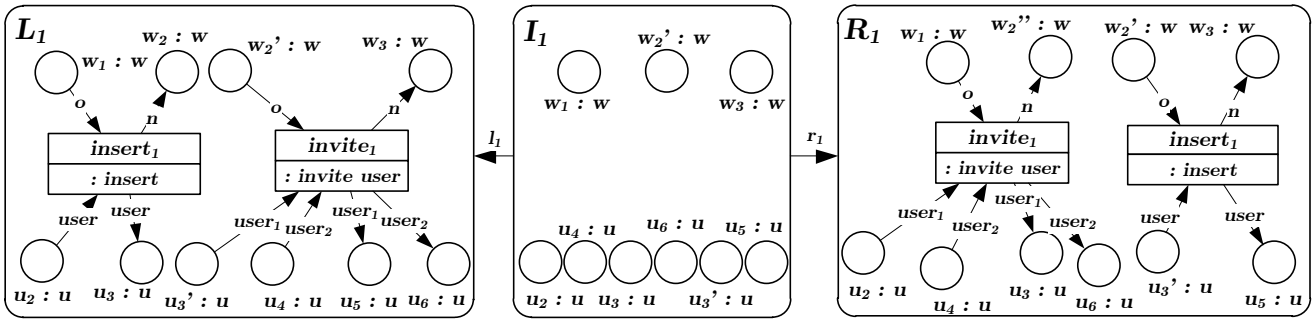


Figure 4.5: Production ϱ_1 for AHL-processes of the *Platform* net

Considering our abstract scenario *Wave* in example Example 4.1.8, depicted in Figure 4.1 on page 60, we can find two different matches $m_1, m_2 : L_1 \to Wave$ that are compatible with the AHL-process $wave : Wave \to Platform$ and the AHL-processes defined in the production.

The first match m_1 maps $insert_1$ to $insert_1$ in *Wave*, and the second match m_2 maps $insert_1$ to $insert_2$, while both of them map $invite_1$ to $invite_1$. The places are matched accordingly to the environments of the connected transitions. As for match m_1 we have identification points w_2, w_2', u_3 and u_3' (see Definition 3.2.13), but place w_2 is not a gluing point since it is deleted by the rule. Hence, the production is not applicable with match m_1, because the gluing condition for AHL-nets is not satisfied. Another violation of the gluing condition would be that w_2 is also a dangling point, and therefore, removing w_2 would cause the o-inscribed arc in the pre domain of $invite_1$ in *Wave* to be a dangling arc.

Now, in the case of match m_2 we do not have any identification points, and matching place w_2 to w_6, we have that w_2 is no dangling point. So the gluing condition for AHL-nets is satisfied and by Fact 3.2.14 we know that we can apply ϱ_1 with match m_2 to obtain a direct transformation of AHL-nets. But for the direct transformation of AHL-process net *Wave*, we additionally need that the transformation condition in Definition 4.2.10 is satisfied, requiring that the gluing relation $<_{(\varrho_1,m_2)}$ is a strict partial order (see Definition 4.2.9).

In the relation $\prec_{(Wave,m_2)}$ we have that u_6 is a predecessor of $remove_1$ which in turn is a predecessor of u_7, and thus in the transitive closure $<_{(Wave,m_2)}$ the place u_6 is a predecessor of u_7. This means that place u_6 is a predecessor of u_2 in I_1 in $\prec_{(\varrho_1,m_2)}$, since $m_2 \circ l_1(u_2) = u_7$. Further, considering the causal relation of R_1, we have that $u_2 <_{R_1} u_6$, and therefore also $u_2 \prec_{(\varrho_1,m_2)} u_6$. Thus, in the gluing relation $<_{(\varrho_1,m_2)}$ which is the transitive closure of $\prec_{(\varrho_1,m_2)}$, we have $u_6 <_{(\varrho_1,m_2)} u_6$. Hence, the gluing relation is not irreflexive which means that it is not a strict partial order. So the transformation condition is not satisfied, and ϱ_1 can not be used for a direct transformation at match m_2.

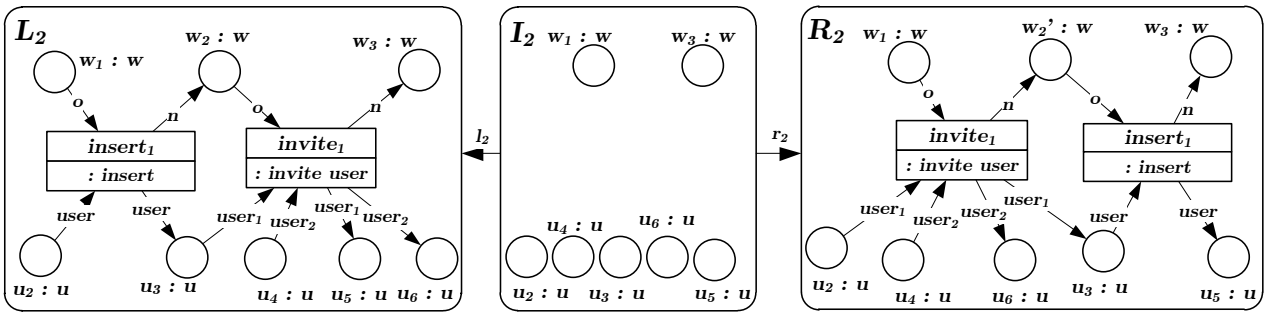


Figure 4.6: Production ϱ_2 for AHL-processes of the *Platform* net

Another production ϱ_2 for AHL-processes of AHL-net *Platform* is depicted in Figure 4.6. The production describes rather the same modification as production ϱ_1, with the exception that the two transitions $insert_1$ and $insert_2$ are connected via places w_2 and u_3. Consequently, there is only one possible match $m_3 : L_2 \to Wave$ into the our *Wave* model with m_3 being an inclusion. The gluing condition is satisfied, since there are no identification points and all dangling points are preserved by the production.

Observing the result $Wave_2$ in Figure 4.7 of the direct transformation $Wave \Rightarrow Wave_2$ using production ϱ_2 at match m_3, one can see that the resulting net is indeed an AHL-process net. By Theorem 4.2.11 we know that the transformation condition for AHL-process nets is not only sufficient but also necessary. Thus, we know that the transformation condition is satisfied by ϱ_2 and m_3 without the need for explicit verification of the condition. Moreover, the AHL-process $wave : Wave \to Platform$ and the corresponding processes specified in the production are consistent as required for the extension to AHL-processes in Theorem 4.2.11 (see also Remark 4.2.12), implying that there is also an AHL-process $wave_2 : Wave_2 \to Platform$ as depicted in Figure 4.7, and we have a direct transformation of AHL-processes $wave \overset{\varrho_2,m_2}{\Longrightarrow} wave$. ♢

4.3 Modelling of Concrete Scenarios Using Instantiations

Based on the skeleton and flattening constructions presented in Section 3.4, instantiations of AHL-processes were defined in [Ehr05]. In [Ehr05], an instantiation of an AHL-occurrence net K was defined as a subnet L of the flattening $Flat(K)$, i. e. there is an inclusion $in : L \hookrightarrow Flat(K)$, such that the composition $proj(K) \circ in : L \to Skel(K)$ is an isomorphism. Intuitively, this means that L is a section of the over-all behaviour of K with the same

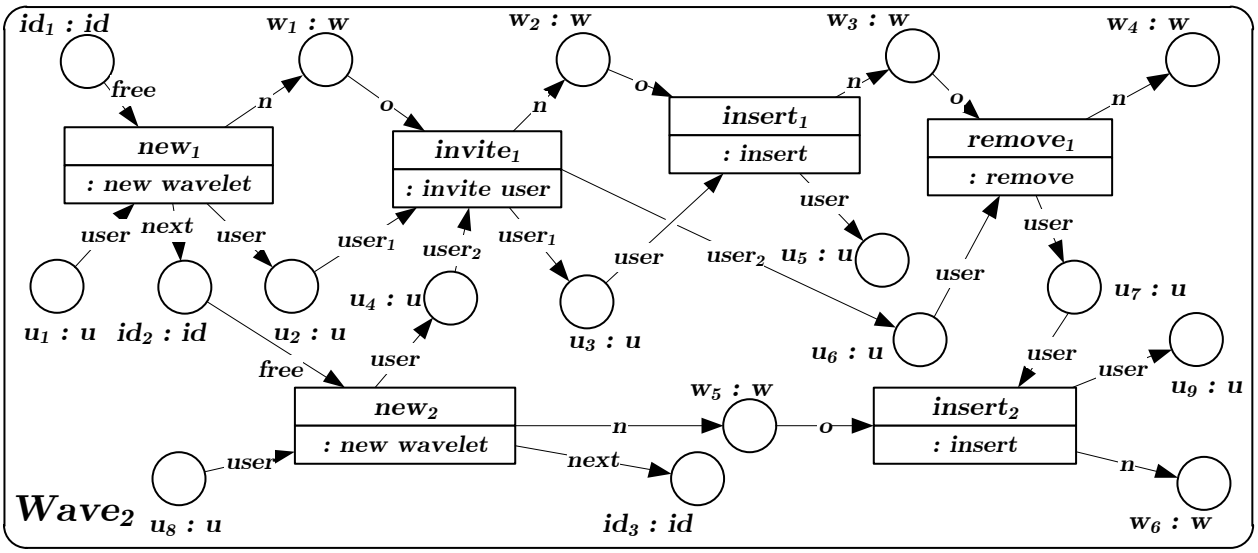


Figure 4.7: Modified abstract scenario $Wave_2$

structure as K itself. As shown in [Ehr05] an instantiation of an AHL-occurrence net K bijectively corresponds, up to concurrency, to exactly one complete firing sequence in K that is compatible with the causal relation.

In this section, we introduce a slightly different definition of instantiations, taking also into account our new definition of weak flattenings. Moreover, we do not restrict our definition to AHL-process nets, but we give a general definition of instantiations for AHL-nets. Note that apart from the generalisation, our definition of instantiations is equivalent to the one in [Ehr05].

Definition 4.3.1 ((Weak) Instantiations of Algebraic High-Level Nets)**.** Given an AHL-net AN, a *concrete instantiation*, or short *instantiation*, $Inst$ of AN is a pair

$$Inst = (inst : Skel(AN) \to Flat(AN), AN)$$

such that $proj(AN) \circ inst = id_{Skel(AN)}$. Moreover, given two instantiations $Inst_i = (inst_i, AN_i)$ for $i = 1, 2$, an *instantiation morphism* $f : Inst_1 \to Inst_2$ is an AHL-morphism $f : AN_1 \to AN_2$ such that $Flat(f) \circ inst_1 = inst_2 \circ Skel(f)$, i. e. diagram (1) below commutes. We define the category **Inst** with instantiations and instantiation morphisms, where composition and identities are defined as composition and identities in **AHLNets**.

$$\begin{array}{ccc} Skel(AN_1) & \xrightarrow{inst_1} & Flat(AN_1) \\ {\scriptstyle Skel(f)}\downarrow & (1) & \downarrow{\scriptstyle Flat(f)} \\ Skel(AN_2) & \xrightarrow[inst_2]{} & Flat(AN_2) \end{array} \qquad \begin{array}{ccc} Skel(AN_1) & \xrightarrow{winst_1} & wFlat(AN_1) \\ {\scriptstyle Skel(f)}\downarrow & (2) & \downarrow{\scriptstyle wFlat(f)} \\ Skel(AN_2) & \xrightarrow[winst_2]{} & wFlat(AN_2) \end{array}$$

Further, the category **wInst** of *weak instantations* is defined analogously using the functor $wFlat$ instead of $Flat$ as illustrated in diagram (2) above. A weak instantiation $(inst, AN)$ is called *abstract instantiation*, if the net AN has a term algebra $(\Sigma, T_\Sigma(X))$ as data type part.[12] △

[12] Note that in general the variable family X is different from the additional variable family that is used for arc inscriptions and firing conditions inside the AHL-net.

The definition of $inst$ as section of $proj$ corresponds exactly to the intuitive meaning of instantiations, described above. Moreover, we shall sometimes refer to the P/T-morphism $inst : Skel(AN) \to Flat(AN)$ as instantiation of AN, if the AHL-net AN in question is clear.

Remark 4.3.2 (Generalisation of Instantiations). Note that the generalisation of the definition of instantiations from AHL-occurrence nets to AHL-nets is only relevant for technical aspects. On the conceptual layer, we are only considering instantiations of AHL-process nets and AHL-processes as defined in the following Definition 4.3.3. The benefits of loosening the definition should become clear in the next Section 4.6, when we consider the evolution of instantiations. In the previous Section 4.2, we defined the evolution of AHL-process nets and AHL-processes in dependence of the corresponding evolution of AHL-nets, which allows us to benefit from the fact that there is an adequate $\mathcal{M}$-adhesive category of AHL-nets, although this is not the case for AHL-process nets and AHL-processes. Similarly, there is no suitable $\mathcal{M}$-adhesive category of instantiations of AHL-process nets or processes, but as shown in Fact A.5.9, we obtain an $\mathcal{M}$-adhesive category, considering instantiations of AHL-nets. This is a great advantage for the theory of the evolution of instantiations, regarding that each AHL-process net is also an AHL-net. △

Definition 4.3.3 ((Weakly) Instantiated AHL-Processes)**.** The categories of (weak) instantiations of AHL-process nets are defined as full subcategories **PInst** $\subseteq$ **Inst** and **wPInst** $\subseteq$ **wInst** where the high-level net parts of the objects are AHL-process nets.

Moreover, we define categories **ProcInst** and **wProcInst** of (weak) instantiations of AHL-processes, also called *(weakly) instantiated AHL-processes*. The objects of these categories are pairs $(inst, mp)$, where $mp : K \to AN$ is an AHL-process and $inst : Skel(K) \to Flat(K)$ is a (weak) instantiation of K. The morphisms of these categories are AHL-process morphisms (see Definition 4.1.5). △

Remark 4.3.4 (Short Notation). We sometimes use a short notation for instantiated AHL-processes: For an instantiated AHL-process $(inst, mp : K \to AN)$ we write $mp : Inst_K \to AN$ where $Inst_K = (inst, K)$ is the corresponding instantiated AHL-process net. △

Remark 4.3.5 (Equivalence of Definitions). In Fact A.5.1 we show that our definition of instantiated AHL-process nets is in fact equivalent to the definition in [Ehr05], where an instantiation of an AHL-occurrence net K was defined as a subnet L of the flattening $Flat(K)$ with inclusion $in : L \hookrightarrow Flat(K)$ such that the composition $proj(K) \circ in : L \to Skel(K)$ is an isomorphism. The explicit representation of L was useful in order to bijectively relate instantiations with initial markings in order to ensure that there is exactly one "run" specified for each initial state. However, in the context of communication platforms as considered in this thesis, it can be expected that outgoing from one initial state of a platform there can happen a variety of different interactions represented by different instantiations with the same initial marking. Therefore we omit the explicit representation of L since it is already implicitly given by the skeleton to which it is isomorphic. △

Concept 4.3.6 (Modelling of Concrete Scenarios)**.** In Concept 4.1.7 we already introduced the concept of abstract scenarios which abstractly specifies a possibly concurrent sequence of actions in a communication platform, however, without consideration of specific data values. In contrast to an abstract scenario, a concrete scenario specifies concrete data values that are used and present throughout the runtime of the scenario.

Concrete scenarios of interactions in a communication platform can be modelled using instantiated strict AHL-processes. The causal and logical structure of resources and actions in the scenario is captured by the structure of the corresponding AHL-process. The concrete

data-values of the resources that are used or produced by the actions of the scenario are specified in the corresponding instantiation of the strict AHL-process net which has the same data type part as the underlying model of the corresponding communication platform. An example of the modelling of a concrete scenario is given in Example 4.3.7, where we model the concrete scenario presented in Figure 2.3 on page 23.

It is interesting to note that we always obtain an abstract scenario model (see Concept 4.1.7) whenever we are creating a concrete scenario model using our modelling framework of instantiated AHL-processes.

Further, note that abstract instantiations can be used to model *semi-concrete* scenarios, i. e. scenarios which are partially abstract. In that case, instead of concrete values, the instantiation part of the model specifies terms for the resources that serve as place holders for a set of different possible values. Concrete realisations of the abstractly defined values (i. e. the terms) can be determined by assignments of the variables, occurring in the terms, into the data type part of the corresponding AHL-net representing the communication platform. The concept of semi-concrete scenarios is discussed further in Concept 4.4.5 and illustrated in Example 4.4.6. △

Example 4.3.7 (Modelling of Concrete Scenario as Instantiated AHL-Process)**.** Consider again the model $Wave$ of an abstract scenario, presented in Figure 4.1. An instantiation $Inst = (inst, Wave)$ of the AHL-process net $Wave$ is shown in Figure 4.8. Instead of explicitly depicting the AHL-morphism $inst : Skel(Wave) \to Flat(Wave)$ which is rather impossible due to the fact that the flattening $Flat(Wave)$ is infinitely large, we use an integrated visualisation based on the fact that the domain $Skel(Wave)$ has the same structure as the AHL-process net $Wave$.

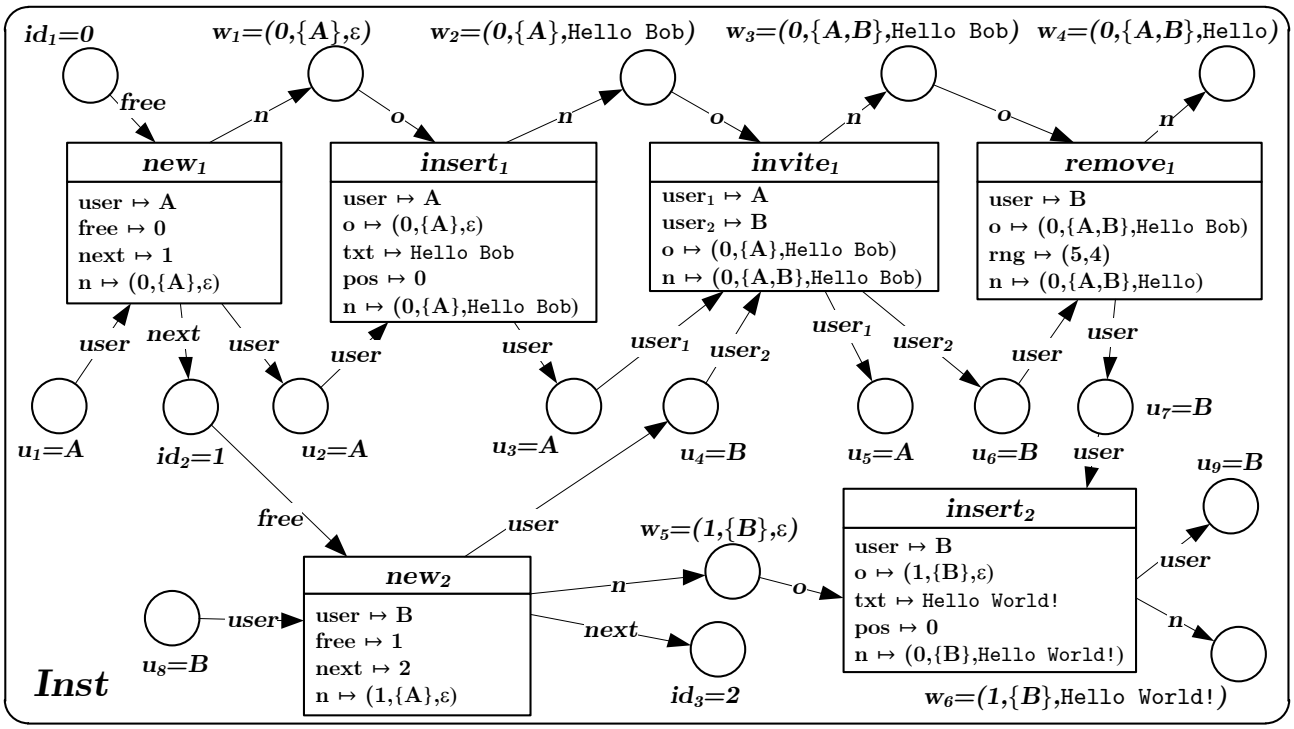


Figure 4.8: Instantiation $Inst$ of AHL-process net $Wave$

The mappings of places are illustrated by a notation $place = value$, indicating that the place in the skeleton is mapped to the corresponding pair $(value, place)$ in the flattening.

For instance, $id_1 = 0$ in the upper left corner of Figure 4.8 indicates that the place id_1 in $Skel(Wave)$ is mapped to the place $(0, id_1)$ in $Flat(Wave)$, where 0 is an element of type $type(id_1) = nat$ in the algebra A (see Table 3.2).

Moreover for the mapping $inst_T(t) = (t, v)$ of a transition, the variable assignment v is denoted element-wise inside the transition, e. g. the transition new_1 is mapped to an assignment v_1 with $v_1(user) = A$, $v_1(free) = 0$, and so on, where A and B are abbreviations for Alice and Bob, respectively.

Note that the skeleton is a low-level P/T net which does not consist of arc inscriptions, but in our visualisation we include the arc inscriptions of the corresponding AHL-process net for a better comprehensibility of the data flow inside the instantiation.

As pointed out in Example 4.1.8, the AHL-process $wave : Wave \to Platform$ models exactly the abstract scenario presented in Figure 2.4 on page 24. Combining the AHL-process $wave$ and the instantiation $Inst$ of the AHL-process net $Wave$, we obtain an instantiated AHL-process $(inst, wave)$.

Now, taking into account the assigned values on the places and the consistent transition assignments, the scenario becomes concrete. For instance, the abstract action new_1 which is an occurrence of the action *new wavelet* in our platform is now bound to be used by user Alice with id 0. The output of the action is a new empty wavelet, where Alice is invited, and the next free id is the natural number 1. In fact, regarding all values and assignments, one can see that the instantiated AHL-process is a model of the concrete scenario depicted in Figure 2.3 on page 23. ♢

Remark 4.3.8 (Unique Representation of Instantiations as Weak Instantiations). According to Fact A.5.3 and Lemma A.5.4, for every instantiation $(inst, AN)$ there are interchangeable representations $(inst, AN)$ in **Inst** and $W(inst, AN)$ in **wInst** which can be uniquely obtained from one another. Due to this fact, we shall often refer to instantiations in their representation as weak instantiations, i. e. as objects of the category **wInst**, where $inst$ is a P/T-morphism with codomain $wFlat(AN)$, despite the fact that instantiations by definition are P/T-morphisms with codomain $Flat(AN)$. △

4.4 Data Evolution and Abstraction of Concrete Scenarios

Based on the data-image of AHL-nets presented in Definition 3.3.1, we define also an data-image of corresponding instantiations. While the data-image construction of AHL-nets does nothing else than replacing the data type part in the AHL-net with another one, the data-image construction for instantiations replaces also the values occurring in the instantiation with values from the target data types. The existence of the data-images is shown in Lemma A.5.15.

Definition 4.4.1 (Data-Image of Instantiations)**.** Given an instantiation $(inst_1, AN_1)$ of an AHL-net $AN_1 = (\Sigma_1, P_1, T_1, pre_1, post_1, cond_1, type_1, A_1)$, and a generalised algebra homomorphism $f = (f_\Sigma, f_A) : (\Sigma_1, A_1) \to (\Sigma_2, A_2)$. We define the *data-image* $\bar{f} : (inst_1, AN_1) \to (inst_2, AN_2)$ of $(inst_1, AN_1)$ along f in the following way:

- $\bar{f} : AN_1 \to AN_2$ is the data-image of AN_1 along f as defined in Definition 3.3.1, and
- $inst_2 = Flat(\bar{f}) \circ inst_1$.

We call $\bar{f} : (inst_1, AN_1) \to (inst_2, AN_2)$ a *basic data-image*, if f_Σ is an identity, i. e. if $\bar{f}$ is a data-image along a non-generalised Σ_1-homomorphism $f_A : A_1 \to A_2$.

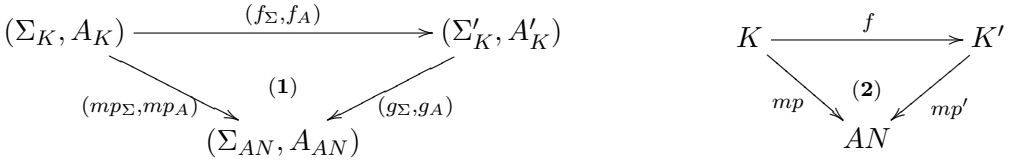
Analogously to the definition of data-images of concrete instantiations above, the data-image of a weak instantiation $(winst_1, AN_1)$ along f is defined as $(winst_2, AN_2)$ where $winst_2 = wFlat(\bar{f}) \circ winst_1$. △

Remark 4.4.2 (Data-Preimages of Instantiations). As pointed out in Remark 3.3.7, data-preimages of AHL-nets only exist if certain conditions are satisfied. In the case of instantiations this applies even to the existence of basic data-preimages. For instance, consider the instantiation $Inst$ in Figure 4.8 and a homomorphism $h : T_{\Sigma\text{-}Wave}(Y) \to A$ with $Y_{user} = \emptyset$. In order to obtain a basic data-preimage $h' : Inst' \to Inst$ of $Inst$ along h, it would b necessary to replace all data values of algebra A occurring in $Inst$ with corresponding data values of term algebra $T_{\Sigma\text{-}Wave}(Y)$. But since there are no generating ground terms for the sort $user$, because there are no constructors providing users in $\Sigma\text{-}Wave$, and there are also no variables of that sort, we have that $T_{\Sigma\text{-}Wave}(Y)_{user} = \emptyset$. Thus, there is, for instance, no suitable data value for the place u_1, and therefore we cannot find even a weak instantiation $Inst'$ such that $Inst$ is the basic data-image of $Inst'$ along h.

Furthermore, let us consider the case that we have $Y_{nat} = \{n\}$ and $h_{nat}(n) = 0$. Then the data value 0 on place id_1 in $Inst$ could be replaced by variable n as well as by the constructor $zero$, because we also have $h_{nat}(zero) = zero_A = 0$. This means that even if the data-preimage would exist, it may be the case that we have several different candidates to be the result of a data-preimage construction. △

As morphisms between AHL-processes of one AHL-net AN are required to be in the slice category $\mathbf{AHLNets} \setminus AN$ (see Definition 4.1.5), for the data evolution of instantiated AHL-processes it is required that the corresponding homomorphism is in the slice category $\mathbf{Algs} \setminus (\Sigma_{AN}, A_{AN})$ over the data type part of the respective AHL-net.

Fact 4.4.3 (Data-Image of Instantiated AHL-Process). *Given an instantiated AHL-process $(inst, mp : K \to AN)$, and generalised algebra homomorphisms $(f_\Sigma, f_A) : (\Sigma_K, A_K) \to (\Sigma'_K, A'_K)$ and $(g_\Sigma, g_A) : (\Sigma'_K, A'_K) \to (\Sigma_{AN}, A_{AN})$ such that diagram (1) below commutes. Then the data-image $f : (inst, K) \to (inst', K')$ of $(inst, K)$ along (f_Σ, f_A) induces a unique AHL-process $mp : K' \to AN$ such that diagram (2) below commutes and $Data(mp') = (g_\Sigma, g_A)$.*



Proof. Given the commuting diagram (1) of generalised algebra homomorphisms and the data-image $f : (inst, K) \to (inst', K')$ of $(inst, K)$ along (f_Σ, f_A). By Definition 4.4.1 we have that AHL-morphism $f : K \to K'$ is the data-image of K along (f_Σ, f_A) which according to Fact A.6.10 means that f is $Data$-cocreation of (f_Σ, f_A) via K. Thus, due to the universal property of cocreations, the homomorphism (g_Σ, g_A) implies a unique AHL-morphism $mp' : K' \to AN$ such that diagram (2) commutes and $Data(mp') = (g_\Sigma, g_A)$. □

Concept 4.4.4 (Data Evolution of Scenarios). In Concept 3.3.2 we already discussed the possibility to evolve the data type part of a communication platform. Since it is possible that we already specified concrete scenarios of the original communication platform, it would be desirable to evolve also the concrete scenario accordingly. This can be achieved using the data-image construction for AHL-process nets in Definition 4.4.1.

It is easy to see that using the data-image construction for instantiations, the first two items of Fact 4.1.9 can also be transferred to instantiated (strict) AHL-processes. This especially means that if we have a concrete scenario of a communication platform, and we evolve the data type part of that communication platform, the data evolution can be applied accordingly to the concrete scenario, and we obtain a corresponding concrete scenario by

replacing all occurrences of concrete values in the instantiation with the corresponding values of the new data type part.

Moreover, since concretisation of communication platforms are a special case of data evolution, it makes sense to consider also the data evolution of semi-concrete scenarios of communication platforms with a more abstract data type part (like a term algebra). Using data evolution to replace the abstract data type part of a platform with a more concrete one, the corresponding data evolution of a semi-concrete scenario of the original platform may result in a concrete scenario of the resulting platform.

The example of a data-image construction from abstract instantiation $AInst$ to instantiation $Inst$ in Example 4.4.6 can be considered as a data-evolution of a semi-concrete scenario of a platform with term algebra as data type part to a concrete scenario of the platform presented in Example 3.1.9. △

As the data-image construction for instantiations is an extension of the data-image construction for AHL-nets (Definition 3.3.1) to instantiations, also the concept of levels of abstractions (Concept 3.3.8) can be extended to scenarios.

Concept 4.4.5 (Levels of Abstraction of Scenarios)**.** In Concept 4.1.7 we introduced the modelling of abstract scenarios using AHL-processes, and in Concept 4.3.6 we introduced the modelling of concrete scenarios using instantiated AHL-processes. We already pointed out, that using abstract instantiations, we can also model semi-concrete scenarios, where we determine only a part of the concrete values in a scenario, while the rest remains abstract. Based on these concepts, we obtain a mechanism of different levels of abstractions for scenarios analogously to the level of abstractions for communication platforms in Concept 3.3.8.

Due to the possibility to extend processes on more concrete communication platforms (see Fact 4.1.9), we have the following cases of concretisation:

(Realisation) A (semi-) concrete scenario $(inst, mp : K \to AN)$ is a concretisation of the abstract scenario $mp : K \to AN$.

(Extension) A (semi-)concrete scenario $(inst, mp : K \to AN)$ is a concretisation of $(inst, mp' : K \to AN')$ if mp is an extension of mp' along a basic data-image $f : AN \to AN'$ (see Fact 4.1.9).

(Data-Image) A (semi-)concrete scenario $(inst, mp : K \to AN)$ is a concretisation of $(inst, mp' : K' \to AN')$, if there is a basic data-image $f : (inst, K) \to (inst, K')$, and there is a corresponding basic data-image $f' : AN \to AN'$ such that $mp' \circ f = f' \circ mp$. Note that each identity $id_{AN} : AN \to AN$ is a data-image along the identity of AN's algebra, implying that also data-images of instantiated AHL-processes as described in Fact 4.4.3 are concretisations of instantiations.

(Transitive Closure) If scenario A is a concretisation of scenario B, and B is a concretisation of scenario C, then A is also a concretisation of C.

We say that scenario A is an abstraction of scenario B, if B is a concretisation of A. The possible different levels of abstraction that can be generated by a single homomorphism are illustrated in Figure 4.9.

We consider a homomorphism $h : T_\Sigma(Y) \to A$, mapping an abstract term algebra into another algebra. If we consider an (abstract) communication platform AN' with algebra part $T_\Sigma(Y)$, according to Concept 3.3.8 we can obtain a concretisation AN of the communication platform by a basic data-image $f : AN' \to AN$ along homomorphism h.

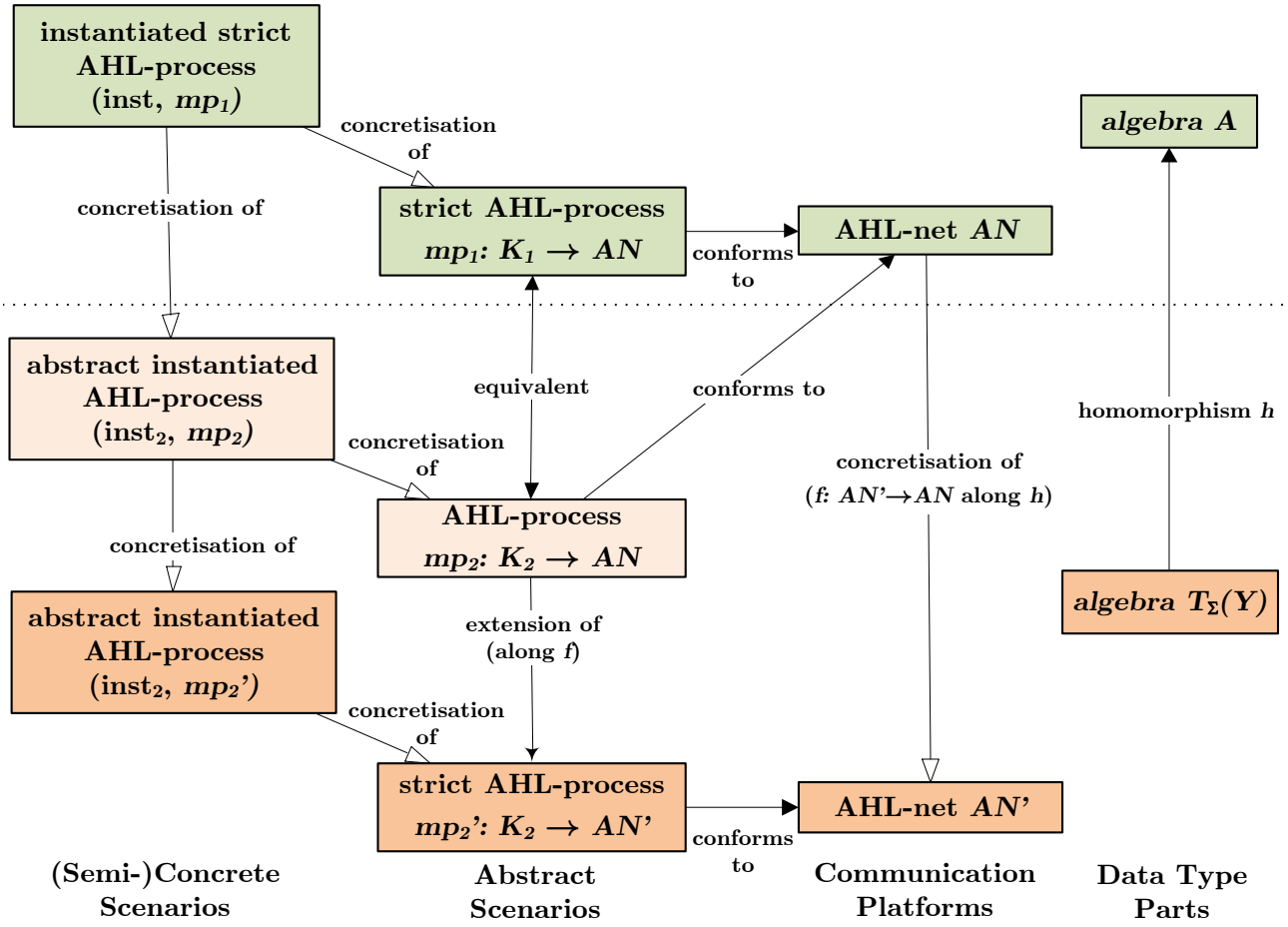


Figure 4.9: Levels of abstraction induced by single homomorphism $h : T_\Sigma(Y) \to A$

Moreover, if we consider a strict AHL-process $mp'_2 : K_2 \to AN'$ of the platform AN', this scenario can be extended along f to a new AHL-process $mp_2 : K_2 \to AN$ that is obtained as composition $mp_2 = f \circ mp'_2$. Note that these two processes describe the same abstract scenario, and we merely changed the context of the scenario from the more abstract version of the communication platform AN' to the more concrete version AN. Changing also the algebra part of the AHL-process net K_2 to the algebra A, we obtain an equivalent AHL-process $mp_1 : K_1 \to AN$ (see Remark 4.1.6). We consider none of the changes on the abstract scenario layer as concretisations, because the algebra part that is contained in an AHL-process net is not used without having any additional context. This changes when we consider also instantiations of the AHL-processes.

Assuming an instantiation $(inst_2, mp'_2)$ of the AHL-process $mp'_2 : K_2 \to AN'$, this instantiation is definitely abstract, because we K_2 has the term algebra $T_\Sigma(Y)$ as algebra part, and thus, the instantiated AHL-process $(inst_2, mp'_2)$ specifies a semi-concrete scenario. This is quite expectable since the instantiated AHL-process specifies a scenario of the abstract communication platform AN'. Due to the extension mp_2 of mp'_2, we automatically have a corresponding instantiated AHL-process $(inst_2, mp_2)$ which is a concretisation of $(inst_2, mp'_2)$. Just like on the abstract scenario layer, we changed only the context of the scenario, but the difference is that this time we possibly have some (more or less) concrete values, and changing the context also changes the interpretation of these values. For instance, let us assume that the signature Σ is the signature Σ-$Wave$ in Table 3.1 on page 36, and A is the algebra in

Table 3.2 on page 37. Then a term value $zero$ occurring in the instantiation $inst_2$, in the context of the AHL-process mp_2 is always evaluated in the algebra A, and therefore this value always stands for the concrete value 0. In contrast, the same term in the context of the AHL-process mp_2' does not yet have a definite interpretation, because the algebra part is the term algebra which can also be mapped to a different algebra B, where the constant $zero$ has a completely different meaning as in A.

So the instantiated AHL-process $(inst_2, mp_2)$ is more concrete than $(inst_2, mp_2')$, and it can be even further concretised by computing the basic data-image $f' : (inst_2, mp_2') \to (inst_1, mp_1)$ along homomorphism h, replacing all abstract data values from term algebra $T_\Sigma(Y)$ by concrete ones from algebra A. Note that this does not necessarily lead to a concrete instantiation, since it is possible that the concrete values for variables in the environment of some transition occurring in the process do not satisfy some of the firing conditions of that transition. So all of the firing conditions have to be checked in order to ensure that the resulting scenario is indeed a concrete realisation of the abstract scenario. If all transition assignments are consistent (i. e. if they satisfy the respective firing conditions), then we have a concrete scenario of interactions in our concrete communication platform AN.

Furthermore, all of the abstract scenarios in Figure 4.9 are abstractions of the respective (semi-)concrete scenarios that are build upon the corresponding AHL-processes. Moreover, note that all these levels of abstractions are based on this single homomorphism h, and it may be possible that there are other homomorphisms $h' : T_\Sigma(Y') \to T_\Sigma$ leading again to a hierarchy of abstractions as shown in Figure 4.9. This means that we can obtain step-wise more abstract or concrete scenarios along this chain of homomorphisms. For instance, a data value can be specified as a variable in $T_\Sigma(Y')$, then as a ground term in $T_\Sigma(Y)$, and finally as a concrete value in A. An example for the abstraction of scenarios is given in Example 4.4.6. △

Example 4.4.6 (Levels of Abstraction Using Abstract Instantiations)**.** We consider a slight alteration $Wave^*$ of the AHL-process net $Wave$, where instead of the algebra A, the net $Wave^*$ consists of a term algebra $T_{\Sigma\text{-}Wave}(Y)$, where Y is a variable family with $Y_{user} = \{a, b\}$, $Y_{nat} = \{p_1, p_2\}$, $Y_{range} = \{r\}$, $Y_{text} = \{t_1, t_2\}$, and $Y_{wavelet} = \{w_1, ..., w_6\}$. Since two terms are only equal if they are syntactically equal, the only equations that hold in a term algebra are equations of the form $x = x$. This means that there are no consistent transition assignments for any of the transitions in $Wave^*$, and thus the AHL-process net $Wave^*$ does not have any instantiations.

However, since for weak instantiations it is not required that the transition assignments are consistent, there are possible abstract instantiations.[13] One example of an abstract instantiation $AInst$ is illustrated in Figure 4.10.

The abstract instantiation $AInst$ is an abstraction of the instantiation $Inst$: The places representing wavelets are assigned with meaningless variables that are named after the corresponding places, and the user places are assigned with variables a and b instead of the concrete users Alice and Bob. Also, the transition assignments do not map the variables to concrete values, but to other variables, consistently with the pre and post domains where the variables occur. The exception are the natural numbers, representing ids, which are not assigned to variables, but id_1 is mapped to the constant $zero$, and id_2 and id_3 are mapped to the ground terms $next(zero)$ and $next(next(zero))$, respectively. According to the interpretation of the operation symbols $zero$ and $next$ in the algebra A, this mapping has the meaning of fixing the ids to the values 0, 1 and 2.

[13] According to Definition 4.3.1, weak instantiations of $Wave^*$ can be called abstract instantiations, because the net has a term algebra as data type part.

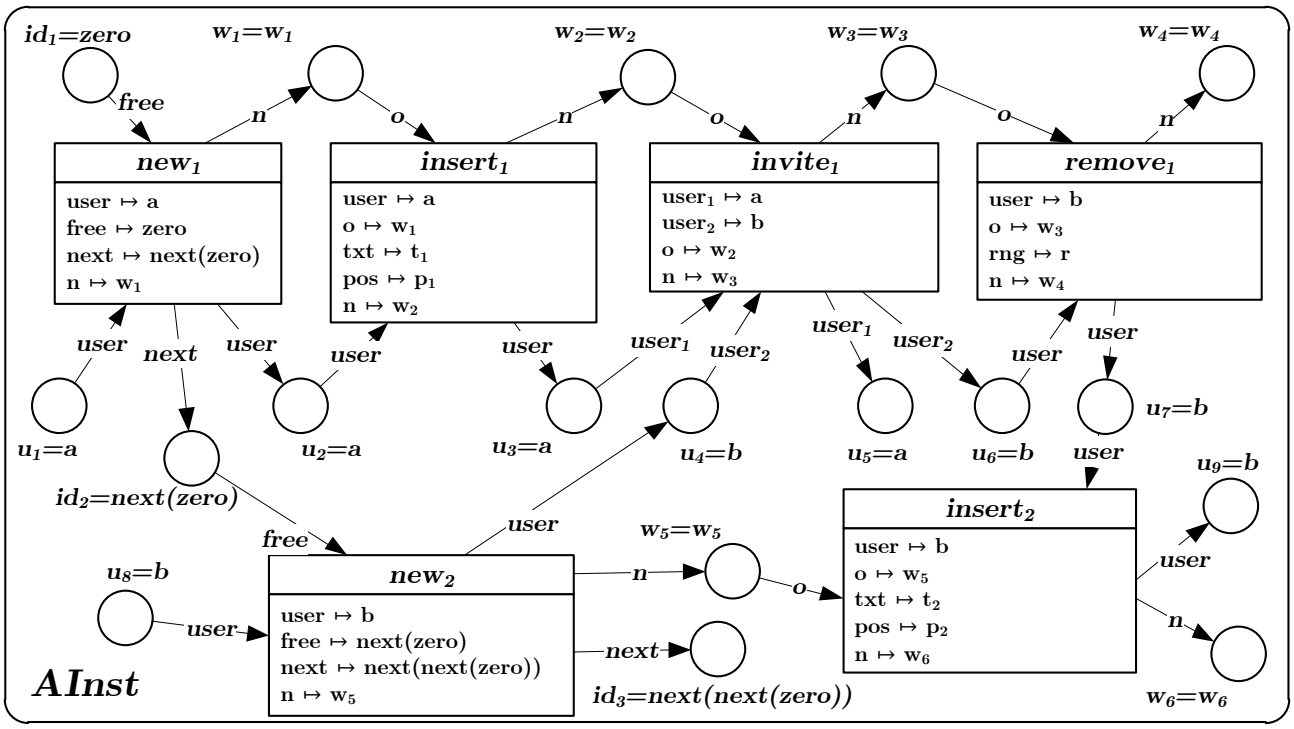


Figure 4.10: Abstract instantiation *AInst* of AHL-process net $Wave^*$

The instantiation *Inst* is a concretisation of *AInst*, in the sense that there is a homomorphism $h : T_{\Sigma\text{-}Wave}(Y) \to A$ such that *Inst* is the basic data-image of *AInst* along h as defined in Definition 4.4.1. Intuitively, this means that we can obtain *Inst* from *AInst* by applying the homomorphism h to all data elements of algebra $T_{\Sigma\text{-}Wave}(Y)$, occurring in *AInst*.

Table 4.1: Variable assignment $asg : Y \to A$

$a \mapsto Alice$	$b \mapsto Bob$
$p_1 \mapsto 0$	$p_2 \mapsto 0$
$t_1 \mapsto$ `Hello Bob`	$t_2 \mapsto$ `Hello World!`
$r \mapsto (5, 4)$	$w_1 \mapsto (0, \{Alice\}, \epsilon)$
$w_2 \mapsto (0, \{Alice\},$ `Hello Bob`$)$	$w_3 \mapsto (0, \{Alice, Bob\},$ `Hello Bob`$)$
$w_4 \mapsto (0, \{Alice, Bob\},$ `Hello`$)$	$w_5 \mapsto (1, \{Bob\}, \epsilon)$
$w_6 \mapsto (1, \{Bob\},$ `Hello World!`$)$	

Since $T_{\Sigma\text{-}Wave}(Y)$ is a term algebra, the homomorphism h can be completely described by a variable assignment $asg : Y \to A$ as shown in Table 4.1.

We can obtain a further abstraction by defining a variable family Y' with $Y' = Y \cup \{z\}$ that is otherwise identical to Y. We can define an assignment $asg' : Y' \to Y$ with $asg(z) = zero$ and all other values are matched identical. Then this assignment induces a unique homomorphism $h' : Y' \to Y$, where all occurrences of z in a term are replaced by the constant *zero*. In fact, we can also define an instantiation *AInst*′ that is obtained from *AInst* by applying the reverse term replacement to the terms occurring in *AInst*, i. e. by replacing occurrences of *zero* with z. Then *AInst*′ is a concretisation of *AInst*, where the ids of the wavelet are not fixed to the values 0, 1 and 2, but instead, they are fixed to three consecutive but otherwise variable values.

Finally, we obtain an abstraction where the values have no restriction (apart from their type[14]), by omitting the instantiation part and considering only the AHL-process $Wave^*$, or equivalently $Wave$, where no concrete data values are specified. ♢

4.5 Restriction and Amalgamation of Scenarios

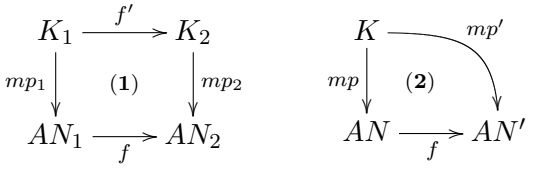
In [EG11] we already defined the restriction and extension of AHL-processes with a fixed data type part. The restriction of an AHL-process $mp_2 : K_2 \to AN_2$ to a subnet $AN_1 \subseteq AN_2$ (or, more generally, along an $\mathcal{M}_{AHL}$-morphism $f : AN_1 \to AN_2$) yields a new AHL-process $mp_1 : K_1 \to AN_1$ that contains only those parts of mp_2 that are also part of the subnet AN_1. The extension of an AHL-process $mp : K \to AN$ to a larger net AN' (or, more generally, along an AHL-morphism $f : AN \to AN'$) is an AHL-process $mp' : K \to AN'$ that has exactly the same behaviour as mp, however, with respect to the given larger net. In the following we define the restriction and extension also for our more general notion of AHL-process, and we show the existence of these constructions for arbitrary AHL-processes.

Definition 4.5.1 (Restriction and Extension of AHL-Processes)**.**

(Restriction) Given an AHL-process $mp_2 : K_2 \to AN_2$ and an $\mathcal{M}_{AHL}$-morphism $f : AN_1 \to AN_2$. An AHL-process $mp_1 : K_1 \to AN_1$ is called restriction of mp_2 along f, if there exists an AHL-morphism $f' : K_1 \to K_2$ such that diagram (1) in Figure 4.11a is a pullback in **AHLNets**.

(Extension) Given an AHL-process $mp : K \to AN$ and an AHL-morphism $f : AN \to AN'$. An AHL-process $mp' : K \to AN'$ is called extension of mp along f, if the diagram (2) in Figure 4.11b commutes.

△



(a) Restriction of AHL-process

(b) Extension of AHL-process

Figure 4.11: Restriction and extension of AHL-processes

Fact 4.5.2 (Restriction and Extension of AHL-Processes)**.**

(Restriction) *Given an AHL-process $mp_2 : K_2 \to AN_2$ and an $\mathcal{M}_{AHL}$-morphism $f : AN_1 \to AN_2$. Then there exists (up to isomorphism) a unique AHL-process $mp_1 : K_1 \to AN_1$ such that mp_1 is restriction of mp_2 along f.*

(Extension) *Given an AHL-process $mp : K \to AN$ and an AHL-morphism $f : AN \to AN'$. Then there exists a unique AHL-process $mp' : K \to AN'$ such that mp' is the extension of mp along f.*

Proof.

[14]Every possible data value in a scenario corresponds either to a place or to a term occurring in the firing conditions of a transitions. Places and terms always have a type, and each value for one of these elements has to be of the corresponding type.

(Restriction) Given an AHL-process $mp_2 : K_2 \to AN_2$ and an $\mathcal{M}_{AHL}$-morphism $f : AN_1 \to AN_2$. Since **AHLNets** is $\mathcal{M}$-adhesive, there exists a pullback (1) along $f \in \mathcal{M}_{AHL}$ as shown in Figure 4.11a in **AHLNets**, and it follows that $f' \in \mathcal{M}_{AHL}$, because $\mathcal{M}_{AHL}$-morphisms are closed under pullback. Then using Lemma A.7.1, we obtain that K_1 is an AHL-process net, because K_2 is an AHL-process net and $f' \in \mathcal{M}_{AHL}$ is transition-injective. Hence, mp_1 is an AHL-process that is a restriction of mp_2 along f. The uniqueness follows from uniqueness of pullbacks.

(Extension) Given an AHL-process $mp : K \to AN$ and an AHL-morphism $f : AN \to AN'$. We obtain an extension $mp' : K \to AN'$ of mp along f by composition $mp' := f \circ mp$. Clearly, mp' is unique, because for every extension $\overline{mp} : K \to AN'$ of mp along f, we have $\overline{mp} = f \circ mp = mp'$.

□

Analogously to the restriction of AHL-processes, we can also define the restriction of instantiations. Similar to the restriction of AHL-processes we define the restriction of instantiations along $\mathcal{M}_{AHL}$-morphisms. Note that the definition can be extended to non-injective AHL-morphisms with an isomorphic data type part, but since we use the restriction of (weak) instantiations in combination with the restriction of AHL-processes, we consider it only in the context of $\mathcal{M}_{AHL}$-morphisms.

Definition 4.5.3 (Restriction of Instantiations and Weak Instantiations)**.** Given a (weak) instantiation $(inst_2, AN_2)$ and an $\mathcal{M}_{AHL}$-morphism $f : AN_1 \to AN_2$. A (weak) instantiation $(inst_1, AN_1)$ is called restriction of $(inst_2, AN_2)$ along f, if the diagram (1) respectively (2) below is a pullback in **PTNets**. △

$$\begin{array}{ccc} Skel(AN_1) & \xrightarrow{inst_1} & Flat(AN_1) \\ {\scriptstyle Skel(f)}\downarrow & (1) & \downarrow{\scriptstyle Flat(f)} \\ Skel(AN_2) & \xrightarrow[inst_2]{} & Flat(AN_2) \end{array} \qquad \begin{array}{ccc} Skel(AN_1) & \xrightarrow{inst_1} & wFlat(AN_1) \\ {\scriptstyle Skel(f)}\downarrow & (2) & \downarrow{\scriptstyle wFlat(f)} \\ Skel(AN_2) & \xrightarrow[inst_2]{} & wFlat(AN_2) \end{array}$$

Fact 4.5.4 (Restriction of (Weak) Instantiations)**.**

1. *Given a (weak) instantiation morphism $f : Inst_1 \to Inst_2$ with $f \in \mathcal{M}_{AHL}$, then $Inst_1$ is a restriction of $Inst_2$ along f.*
2. *Given a (weak) instantiation $(inst_2, AN_2)$, and an $\mathcal{M}_{AHL}$-morphism $f : AN_1 \to AN_2$, then there exists a unique restriction $(inst_1, AN_1)$ of $(inst_2, AN_2)$ along f.*

Proof. We show the fact w. l. o. g. only for instantiations and not for weak instantiations, since $Flat$ and $wFlat$ as well as $proj$ and $wproj$ share the same properties that are required for the proof.

1. Given an instantiation morphism $f : (inst_1, AN_1) \to (inst_2, AN_2)$ with $f \in \mathcal{M}_{AHL}$, we have that diagram (1) and the outer triangles in Figure 4.12 commute. Moreover, due to Fact 3.4.9, diagram (2) is a pullback, and together with the trivial pullback in the outer square, we obtain by pullback decomposition that also (1) is a pullback. Hence, $(inst_1, AN_1)$ is a restriction of $(inst_2, AN_2)$ along f.
2. Given an instantiation $(inst_2, AN_2)$, and an $\mathcal{M}_{AHL}$-morphism $f : AN_1 \to AN_2$, again we have the pullback (2) and commuting outer square in Figure 4.12. Thus, the universal pullback property implies a unique P/T-morphism $inst_1 : Skel(AN_1) \to Flat(AN_1)$ such that diagram (1) and the upper triangle in Figure 4.12 commute. These commutativities mean that $(inst_1, AN_1)$ is an instantiation and that $f : (inst_1, AN_1) \to$

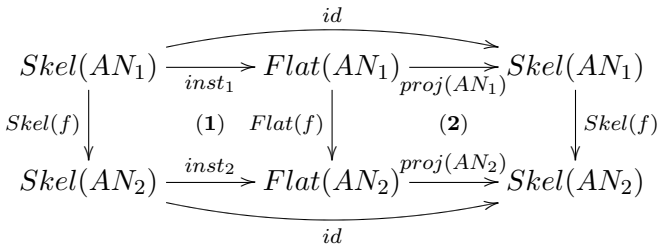


Figure 4.12: Restriction of instantiation

$(inst_2, AN_2)$ is an instantiation morphism, and according to item 1, we have that $(inst_1, AN_1)$ is a restriction of $(inst_2, AN_2)$ along f.

□

Based on the restriction and extension of AHL-processes (Definition 4.5.1) without instantiations, and the restriction of instantiations in Definition 4.5.3, we can also define the restriction and extension of instantiated AHL-processes.

Definition 4.5.5 (Restriction and Extension of (Weakly) Instantiated AHL-Processes)**.**

(Restriction) Given a (weakly) instantiated AHL-process $(inst_2, mp_2 : K_2 \to AN_2)$ and an AHL-morphism $f : AN_1 \to AN_2 \in \mathcal{M}_{AHL}$. A (weakly) instantiated AHL-process $(inst_1, mp_1 : K_1 \to AN_1)$ is called restriction of $(inst_2, mp_2)$ along f, if mp_1 is a restriction of mp_2 along f with corresponding $f' : K_1 \to K_2$ (see Definition 4.5.1), and $(inst_1, K_1)$ is a restriction of $(inst_2, K_2)$ along f' (see Definition 4.5.3).

(Extension) Given a (weakly) instantiated AHL-process $(inst, mp : K \to AN)$ and an AHL-morphism $f : AN \to AN'$. A (weakly) instantiated AHL-process $(inst, mp' : K \to AN')$ is called extension of $(inst, mp)$ along f, if mp' is an extension of mp (see Definition 4.5.1).

△

Fact 4.5.6 (Restriction and Extension of (Weakly) Instantiated AHL-Processes)**.**

(Restriction) *Given a (weakly) instantiated AHL-process $(inst_2, mp_2)$ and an AHL-morphism $f : AN_1 \to AN_2 \in \mathcal{M}_{AHL}$. Then there exists (up to isomorphism) a unique restriction $(inst_1, mp_1)$ of $(inst_2, mp_2)$ along f.*

(Extension) *Given a (weakly) instantiated AHL-process $(inst, mp)$ and an AHL-morphism $f : AN \to AN'$. Then there exists a unique extension $(inst, mp')$ of $(inst, mp)$ along f.*

Proof.

(Restriction) Given a (weakly) instantiated AHL-process $(inst_2, mp_2 : K_2 \to AN_2)$ and an AHL-morphism $f : AN_1 \to AN_2 \in \mathcal{M}_{AHL}$, due to Fact 4.5.2 there exists up to isomorphism a unique restriction mp_1 of mp_2 along f with corresponding $f' : K_1 \to K_2$. Then due to Fact 4.5.4, we obtain a unique restriction $(inst_1, K_1)$ of $(inst_2, K_2)$ along f' which means that $(inst_1, mp_1)$ is a restriction of $(inst_2, mp_2)$ along f.

(Extension) This follows directly from the existence of unique extensions of AHL-processes (Fact 4.5.2).

□

In the following we consider the amalgamation of instantiated AHL-processes, that is, we want to glue instantiated AHL-processes according to a gluing of the respective platforms to

which the processes conform. The gluing of AHL-processes was already introduced in Section 4.2. It remains to consider the gluing of instantiated AHL-processes which is introduced in the following definition.

Definition 4.5.7 (Gluing and Composability of Instantiated AHL-Process Nets)**.** Given instantiations $Inst_0$, $Inst_1$ and $Inst_2$ and instantiation morphisms $f : Inst_0 \to Inst_1$, $g : Inst_0 \to Inst_2$ with $f \in \mathcal{M}_{AHL}$. An instantiation $Inst_3$ with morphisms $g' : Inst_1 \to Inst_3$ and $f' : Inst_2 \to Inst_3$ is called *gluing* of $Inst_1$ and $Inst_2$ over the interface $Inst_0$, written $Inst_3 = Inst_1 +_{Inst_0} Inst_2$, if diagram (1) below is a pushout in **Inst** (see Fact A.5.8 on page 209 for the construction).

$$\begin{array}{ccc} Inst_0 & \xrightarrow{f} & Inst_1 \\ {\scriptstyle g}\downarrow & (\mathbf{1}) & \downarrow{\scriptstyle g'} \\ Inst_2 & \xrightarrow[f']{} & Inst_3 \end{array}$$

Moreover, let $Inst_0 = (inst_0, K_0)$, $Inst_1 = (inst_1, K_1)$ and $Inst_2 = (inst_2, K_2)$ be instantiated AHL-process nets. The gluing $Inst_3 = Inst_1 +_{Inst_0} Inst_2$ is a *gluing of instantiated AHL-process nets* if $Inst_3$ is an instantiated AHL-process net. We say that $Inst_1$ and $Inst_2$ are composable w. r. t. $(Inst_0, f, g)$ if K_1 and K_2 are composable w. r. t. (K_0, f, g) (see Definition 4.2.5). △

Fact 4.5.8 (Gluing of Instantiated AHL-Process Nets)**.** *Given instantiations $Inst_0$, $Inst_1$ and $Inst_2$ and instantiation morphisms $f : Inst_0 \to Inst_1$, $g : Inst_0 \to Inst_2$ with $f \in \mathcal{M}_{AHL}$. The gluing of instantiations $Inst_3 = Inst_1 +_{Inst_0} Inst_2$ is a gluing of instantiated AHL-process nets if and only if $Inst_1$ and $Inst_2$ are composable w. r. t. $(Inst_0, f, g)$.*

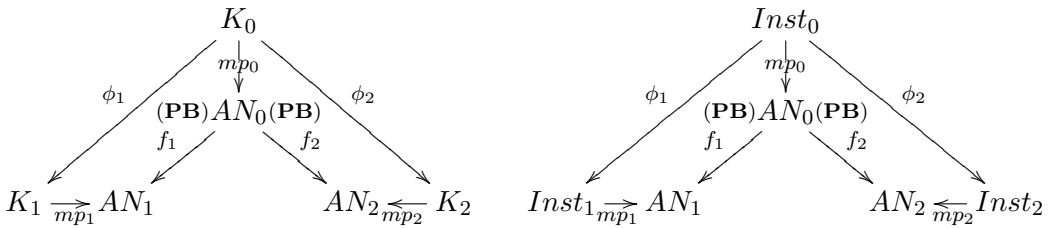
Proof. It suffices to show that the underlying gluing of the AHL-net parts of the instantiations leads to a gluing of AHL-process nets if and only if the instantiated AHL-process nets are composable. This follows directly from Fact 4.2.6, since composability of instantiated AHL-process nets means composability of the underlying AHL-process nets. □

Concept 4.5.9 (Composition of Concrete Scenarios)**.** For different scenarios it may be desirable to combine them, in order to obtain a new integrated scenario, where the single subcomponents are run parallel, sequentially, or that they are synchronised at some points and run otherwise in parallel. The composition of concrete scenarios can be modelled as gluing of the corresponding instantiated ahl-processes, where the interface defines all synchronisation points of the composition. If the interface is empty, the gluing leads to a new scenario, where the original scenarios run completely parallel. An example of the composition of two concrete scenarios is given in Example 4.5.16. Note that the AHL-processes $wave_1$ and $wave_2$ in the example correspond to different platforms. However, by composition $g_1 \circ wave_1$ and $g_2 \circ wave_2$, we obtain AHL-processes of the same platform $Platform_3$. △

Based on the restriction and gluing constructions of (instantiated) AHL-processes defined above, we can define a suitable condition under which we can continue the composition of AHL-nets via a span of $\mathcal{M}_{AHL}$-morphisms $f_1 : AN_0 \to AN_1$ and $f_2 : AN_0 \to AN_2$ to a composition of their processes. Two processes mp_1 and mp_2 of AN_1 and AN_2, respectively, "agree" on the net AN_0 if we can construct a common restriction of the processes leading to a process mp_0 of AN_0 which can be used as a composition interface for mp_1 and mp_2.

Definition 4.5.10 (Agreement of AHL-Processes)**.** Given two AHL-processes $mp_1 : K_1 \to AN_1$ and $mp_2 : K_2 \to AN_2$ and two $\mathcal{M}_{AHL}$-morphisms $f_1 : AN_0 \to AN_1$, $f_2 : AN_0 \to AN_2$. The processes mp_1 and mp_2 *agree* on mp_0 if there exist restrictions (mp_0, ϕ_i) of mp_i along

f_i for $i \in \{1,2\}$ such that for $mp_0 : K_0 \to AN_0$ the AHL-process nets K_1 and K_2 are composable w. r. t. (K_0, ϕ_1, ϕ_2) (see Definition 4.2.5). (mp_0, ϕ_1) and (mp_0, ϕ_2) are called *agreement restrictions* for mp_1 and mp_2.



(a) without instantiations (b) with instantiations

Figure 4.13: Agreement restrictions of AHL-processes

Accordingly, if $mp_1 : Inst_1 \to AN_1$ and $mp_2 : Inst_2 \to AN_2$ are instantiated AHL-processes[15], they *agree* if there exists an instantiated AHL-process $mp_0 : Inst_0 \to AN_0$ that is the common restriction of mp_1 and mp_2 along f_1 and f_2, respectively, as shown in Figure 4.13b, and we have that $Inst_1$ and $Inst_2$ are composable w. r. t. $(Inst_0, \phi_1, \phi_2)$ (see Definition 4.5.7). △

If two processes mp_1 of AN_1 and mp_2 of AN_2 agree, then they can be amalgamated. This means that they are composed to a process mp_3 of AN_3 which is the composition of AN_1 and AN_2, such that mp_1 and mp_2 are restrictions of mp_3.

Definition 4.5.11 (Amalgamation of AHL-Processes)**.** Let AN_3 be the gluing of AHL-nets AN_1 and AN_2 over $\mathcal{M}_{AHL}$-morphisms f_1 and f_2 as shown in diagram (PO) of Figure 4.14a, and $mp_i : K_i \to AN_i$ be AHL-processes for $i \in \{0,1,2,3\}$ such that mp_1 and mp_2 agree on mp_0. This means that (1) and (2) are pullbacks, and K_1 and K_2 are composable w. r. t. (K_0, ϕ_1, ϕ_2), i. e. there is a gluing of AHL-process nets $K_3 = K_1 +_{K_0} K_2$ as shown in the outer square of Figure 4.14a.

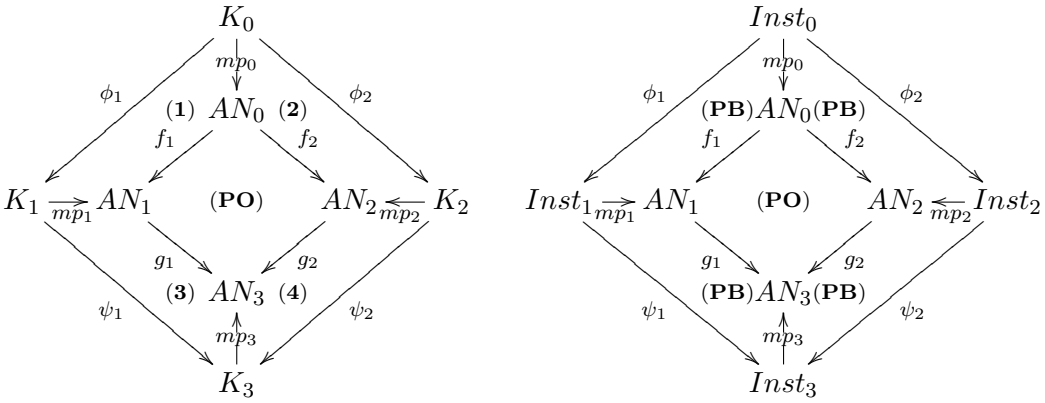
Then an AHL-process $mp_3 : K_3 \to AN_3$ is called *amalgamation* of mp_1 and mp_2 along mp_0, written $mp_3 = mp_1 +_{mp_0} mp_2$, if there exist restrictions (mp_1, ψ_1) and (mp_2, ψ_2) of mp_3 along g_1 and g_2 in (3) and (4).

Accordingly, an instantiated AHL-process $mp_3 : Inst_3 \to AN_3$ is called *amalgamation* of agreeing instantiated AHL-processes $mp_1 : Inst_1 \to AN_1$ and $mp_2 : Inst_2 \to AN_2$, if $Inst_3$ is the gluing of $Inst_1$ and $Inst_2$ as shown in the outer square of Figure 4.14b, and mp_1 and mp_2 are restrictions of mp_3 along g_1 and g_2, respectively. △

The results of amalgamation composition (by gluing) and decomposition (by restriction) constructions are unique up to isomorphism. In order to capture the bijective correspondence of these constructions we define isomorphism classes of AHL-processes and spans of AHL-processes analogously to isomorphism classes of open net processes and spans of these processes in [BCEH01].

An isomorphism between processes $mp : K \to AN$ and $mp' : K' \to AN$ of an AHL-net AN is an isomorphism $iso : K \to K'$ in the category **AHLNets** which is also a morphism in **Proc**(**AN**), i. e. diagram (1) in Figure 4.15 commutes. We denote the isomorphism class of a process mp as the set $[mp] = \{mp' \mid mp' \cong mp\}$ of all processes which are isomorphic to mp.

[15]Note that here and in the following we use a short-hand notation for instantiated AHL-processes as described in Remark 4.3.4. The notation $mp_1 : Inst_1 \to AN_1$ means that we have an instantiated AHL-process $(inst_1, mp_1 : K_1 \to AN_1)$ such that $Inst_1 = (inst_1, K_1)$ is an instantiated AHL-process net.



(a) without instantiations (b) with instantiations

Figure 4.14: Amalgamation of AHL-processes

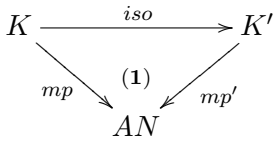


Figure 4.15: Isomorphism of Processes

An isomorphism of spans of processes $(mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2) \cong (mp'_1 \overset{\phi'_1}{\leftarrow} mp'_0 \overset{\phi'_2}{\rightarrow} mp'_2)$ means that there are process isomorphisms $iso_i : mp_i \rightarrow mp'_i$ such that the diagram in Figure 4.16 commutes.

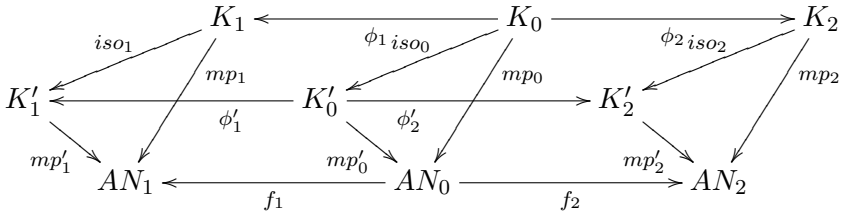


Figure 4.16: Isomorphism of spans of processes

Definition 4.5.12 (Sets of Isomorphism Classes)**.** The set of all isomorphism classes of processes of a given AHL-net AN is defined as

$$Proc(AN) = \{[mp] \mid mp : K \rightarrow AN \text{ is an AHL-process } \}$$

The set of all isomorphism classes of spans of agreeing AHL-processes with respect to a given span of AHL-morphisms is defined as

$$Proc(AN_1 \overset{f_1}{\leftarrow} AN_0 \overset{f_2}{\rightarrow} AN_2) =$$

$$\{\left[mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2\right] \mid \phi_1, \phi_2 \text{ are agreement projections of } mp_1, mp_2 \text{ along } f_1, f_2\}$$

△

Theorem 4.5.13 (Amalgamation Theorem for AHL-Processes)**.** *Given the gluing (pushout) $AN_3 = AN_1 +_{AN_0} AN_2$ of AHL-nets in (PO) of Figure 4.14 with $\mathcal{M}_{AHL}$-morphisms f_1 and f_2, then we have:*

1. ***Composition Construction.*** *Let $mp_i : K_i \to AN_i$ be AHL-processes for $i \in \{0,1,2\}$ such that mp_1 and mp_2 agree on mp_0, then the amalgamation $mp_3 = mp_1 +_{mp_0} mp_2$ exists and is an AHL-process $mp_3 : K_3 \to AN_3$.*

2. ***Decomposition Construction.*** *Let $mp_3 : K_3 \to AN_3$ be an AHL-process and let mp_1, mp_2 be restrictions of mp_3 along g_1 respectively g_2, and mp_0 restriction of mp_1 along f_1. Then mp_3 can be represented as amalgamation $mp_3 = mp_1 +_{mp_0} mp_2$.*

3. ***Bijective Correspondence.*** *There are composition and decomposition functions*

$$Comp : Proc(AN_1 \xleftarrow{f_1} AN_0 \xrightarrow{f_2} AN_2) \to Proc(AN)$$

and

$$Decomp : Proc(AN) \to Proc(AN_1 \xleftarrow{f_1} AN_0 \xrightarrow{f_2} AN_2)$$

establishing a bijective correspondence between $Proc(AN)$ and $Proc(AN_1 \xleftarrow{f_1} AN_0 \xrightarrow{f_2} AN_2)$.

4. ***Instantiations.*** *There exist corresponding composition and decomposition constructions for instantiated AHL-processes, similarly establishing a bijective correspondence.*

Proof Idea. 1. Agreement of mp_1 and mp_2 on mp_0 impies that mp_0 exists in Figure 4.14 such that (1) and (2) are pullbacks and the outer diagram can be constructed as pushout of AHL-nets leading to an AHL-process net K_3. The universal pushout property implies a unique $mp_3 : K_3 \to AN_3$ such that (3) and (4) commute. Finally, the (horizontal) VK-property for the weak adhesive HLR category $(\mathbf{AHLNets}, \mathcal{M})$ implies that (3) and (4) are pullbacks such that mp_1 and mp_2 become restrictions of mp_3 along g_1 respectively g_2 leading to the amalgamation $mp_3 = mp_1 +_{mp_0} mp_2$.

2. In this case we have given in Figure 4.14 the inner pushout (PO) and $mp_3 : K_3 \to AN_3$. Now mp_1, mp_2, mp_0 are constructed as pullbacks in (3), (4) and (1), respectively, and (2) can be shown to become pullback such that K_1 and K_2 are composable w. r. t. (K_0, ϕ_1, ϕ_2). Finally, the (horizontal) VK-property in Figure 4.14 implies that the outer diagram is pushout such that $mp_3 = mp_1 +_{mp_0} mp_2$ becomes the amalgamation of mp_1 and mp_2 along mp_0.

3. Follows from uniqueness (up to isomorphism) of pushout and pullback constructions.

4. Follows from the fact that we have corresponding definitions for the restriction, agreement and amalgamation of instantiated AHL-processes that are based on the underlying constructions for AHL-processes.

For the detailed proof see Section B.7 on page 247. □

Theorem 4.5.13 implies that we have a compositional process semantics of AHL-nets in the following sense.

Theorem 4.5.14 (Compositional Process Semantics of AHL-Nets)**.** *Given AHL-nets AN_i $(i = 0,1,2,3)$ with $AN_3 = AN_1 +_{AN_0} AN_2$ gluing (pushout) object in (PO) of Figure 4.14. Then each process $mp_3 : K_3 \to AN_3$ of AN_3 is uniquely (up to isomorphism) represented by a pair of (instantiated) AHL-processes (mp_1, mp_2) of AN_1 and AN_2, respectively which agree on process mp_0 of AN_0 obtained as common restriction of mp_1 and mp_2. This means that the process semantics of AN_3, defined by all processes mp_3 over AN_3, is completely determined by the process semantics of the components AN_1 and AN_2 with shared AN_0.*

Concept 4.5.15 (Views on Scenarios)**.** Regarding the compositionality of platforms, it should also be possible to restrict scenarios to the current view on the platform. This is particularly interesting in the context of cross-platform communication, where different platforms together form a common larger platform.

Due to the modelling of communication platforms using AHL-nets (see Concept 3.1.8), it is possible to model the embedding of parts of a communication platform into a larger context using $\mathcal{M}_{AHL}$-morphisms. The injectivity of an $\mathcal{M}_{AHL}$-morphisms $f : \mathit{Platform}_1 \to \mathit{Platform}_2$ ensures that $\mathit{Platform}_1$ is indeed a sub net of $\mathit{Platform}_2$, and the isomorphic data type part ensures that the smaller platform operates on the same data as the super net.

We can model the restriction of abstract and concrete scenarios of communication platforms to smaller parts of that platform using the restriction of AHL-processes and instantiated AHL-processes, respectively. Moreover, scenarios that operate only on a part of a communication platform can be extended to scenarios of the whole platform using the extension of (instantiated) AHL-processes along the embedding of the platform.

Regarding the gluing of communication platforms, using amalgamation of (instantiated) AHL-processes, it is possible to restrict scenarios of the gluing to scenarios of the single parts. Vice versa, it is possible to combine scenarios of the single parts to a new scenario of the gluing, provided that the scenarios agree on the interface of the gluing. This can be used for different purposes. On one hand, the gluing of communication platforms can represent the assembly of one platform from different subcomponents. The amalgamation of scenarios over such a gluing corresponds to different views on one scenario, where either we consider only a particular subset of resources and actions that corresponds to one of the subcomponents, or we consider the whole scenario corresponding to the whole platform. On the other hand, the gluing of communication platforms can represent the gluing of completely different communication platforms, forming a new communication platform that represents the communication options of both of the single platforms as well as, depending on the interface, the cross-platform communication between them.

Examples of the restriction, extension and amalgamation of scenarios are given in Example 4.5.16. △

Example 4.5.16 (Views on Scenarios)**.** An example of the amalgamation of instantiated AHL-processes is shown in Figure 4.17, where the interfaces are in the top-left area and the results of the gluings are depicted in the bottom-right area of the figure. The AHL-net $\mathit{Platform}_1$ consists of a single action *new wavelet* and its environment, whereas the AHL-net $\mathit{Platform}_2$ consists of a single action *insert* and its environment. The actions correspond the the actions with the same name in our AHL-net *Platform* introduced in Example 3.1.9. The interface of the gluing of these two platforms contains the resources u (users) and w (wavelets). The result of the gluing is an AHL-net $\mathit{Platform}_3$, consisting of actions *new wavelet* and *insert* operating on a common set of resources of users and wavelets.

A scenario of $\mathit{Platform}_3$ is depicted in the bottom-right of Figure 4.17. Note that for the sake of brevity we combined the graphical notations used for AHL-processes and instantiations. An example of the amalgamation of AHL-processes without instantiations can be obtained by simply removing all annotations that describe the assignment of values and transition assignments.

The scenario Wave_3 describes the subsequent creation of new wavelets with IDs 0 and 1, respectively, and the insertion of the text "`Hello Bob`" into the wavelet with ID 0 that happens parallel to the creation of the second wavelet with ID 1. The scenarios Wave_1 and Wave_2 are restrictions of the scenario Wave_3 to the sub-nets $\mathit{Platform}_1$ and $\mathit{Platform}_2$, respectively, of $\mathit{Platform}_3$. These scenarios consist only of those elements that are "visible" (or present) on the respective corresponding communication platform. Thus, the scenario Wave_1

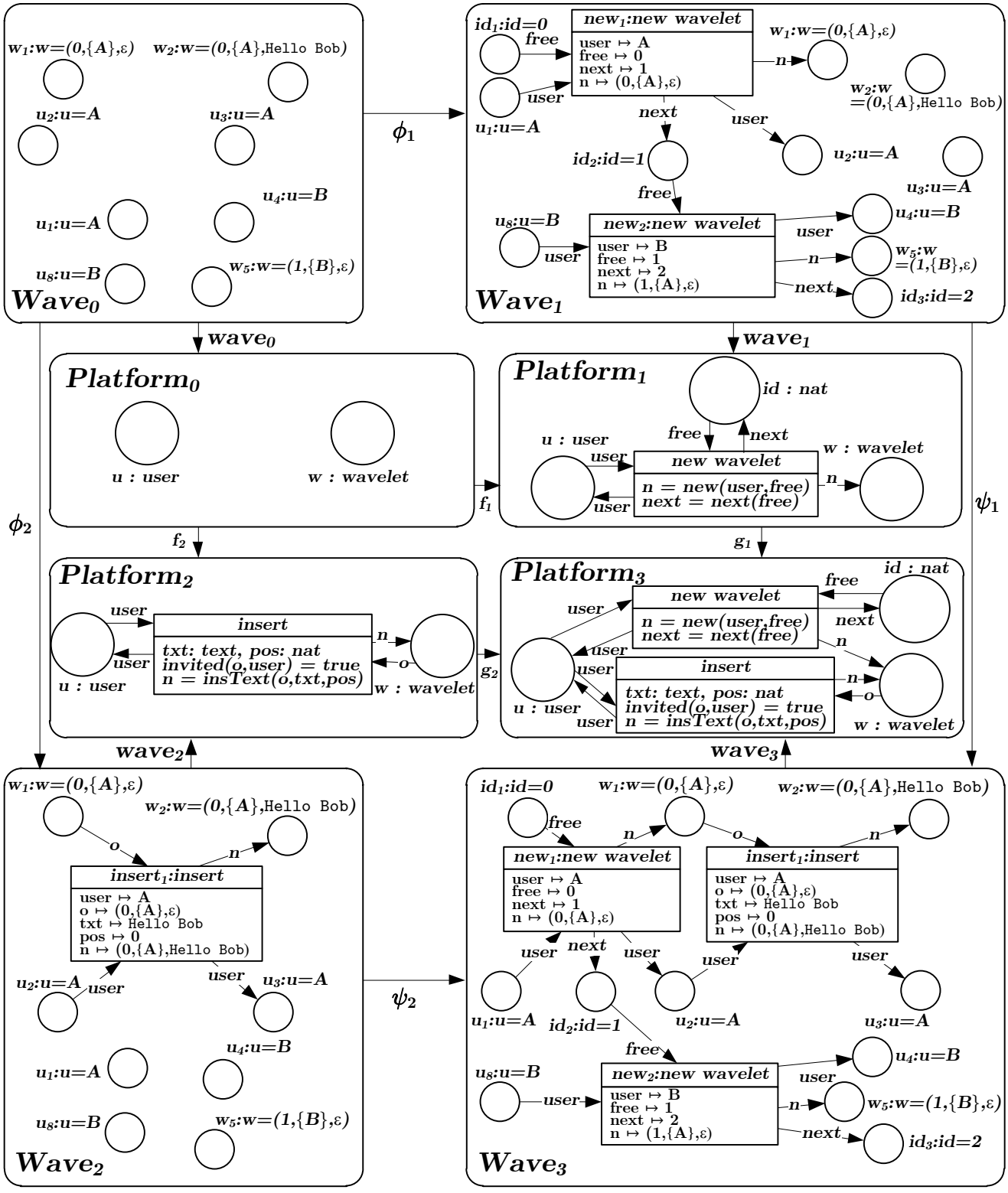


Figure 4.17: Amalgamation of scenarios

contains only occurrences of the *new wavelet* action, whereas the scenario $Wave_2$ contains only the occurrence of the *insert* action. Moreover, the scenario $Wave_2$ does not contain any values for free IDs, since this resource is not present in the corresponding platform. The

values for user and wavelet resources are retained in both of the sub-scenarios, since these resources are part of the interface. The two scenarios agree on the interface in the scenario $Wave_0$ which is the common restriction of $Wave_1$ and $Wave_2$ to the interface $Platform_0$. The scenario $Wave_3$ is the gluing of scenarios $Wave_1$ and $Wave_2$ over the interface $Wave_0$.

Note that we also have extensions of scenarios $g_1 \circ wave_1 : Wave_1 \to Platform_3$ and $g_2 \circ wave_2 : Wave_2 \to Platform_3$ that are scenarios of the AHL-net $Platform_3$, but the scenarios are restricted views of the scenario $wave_3$ that contain only those parts that are present in the respective smaller platform.

The amalgamation can be seen as model for the cross-platform communication between two different platforms. On one platform, it is only possible to create wavelets, but it is not possible to modify them. On the other platform, one can insert text into wavelets, but it is not possible to create new ones. There are two types of resources both of the two platforms know of: users and wavelets. Together these two platforms can be seen as a super-platform that allows the creation of new wavelets as well as the insertion of text. For the realisation of scenario $Wave_3$ it is necessary to use the exclusive feature of each of the platforms. So, if we restrict the scenario to the $Platform_2$, we do not see the creation of new wavelets, because this action is unknown on the platform. But since the platforms knows the resource w (the wavelets), we can see the result of the creation of wavelets[16]. Then the first new wavelet can be used on $Platform_2$ to insert the text "`Hello Bob`", resulting also in a new wavelet with that text on $Platform_1$, because wavelets (represented by the place w) are part of the interface, whereas it is not visible on that platform how the insertion of the text has happened, because this action is unknown on $Platform_1$. Considering the super-platform $Platform_3$, we know of all involved resources and actions, and therefore the amalgamated scenario $Wave_3$ is a complete description of the overall process of interactions on the single platforms. ♢

4.6 Structural Evolution of Concrete Scenarios

An approach for the rule-based transformation of instantiations of AHL-processes was already presented in [Gab09], but in that work we considered only AHL-nets and processes with a fixed data type part. For this reason, a major drawback of that approach was that the applicability of the transformation was highly dependent on the concrete data values and assignments in the original instantiation. In this work, we introduced the new concept of weak instantiations (see Definition 4.3.1) that can be used to define abstractly instantiated rules for instantiated AHL-(process) nets and processes, covering an abstract set of possible data values and assignments.

Definition 4.6.1 (Production for Instantiations)**.** A production for instantiations is given by a span $\varrho : Inst_L \xleftarrow{l} Inst_I \xrightarrow{r} Inst_R$ of weak instantiation morphisms $l, r \in \mathcal{M}_{AHL}$. The production is called *abstract*, if $Inst_L$, $Inst_I$ and $Inst_R$ are abstract instantiations[17], and the production is called *concrete*, if $Inst_L$, $Inst_I$ and $Inst_R$ are concrete instantiations[18].

We say that ϱ is a production for instantiated AHL-process nets, if the AHL-net part $Net(\varrho) : L \xleftarrow{l} I \xrightarrow{r} R$ is a production for AHL-process nets. △

Example 4.6.2 (Production for Instantiations)**.** An example of an abstract production ϱ for instantiations is shown in Figure 4.18, where the AHL-net part $Net(\varrho)$ of the production is the production ϱ_2 for AHL-processes in Figure 4.6 on page 69. The data type part of

[16] Of course, for this to work, the resources probably have to be submitted over a network somehow, but we do not concentrate on the corresponding technical details in this work.

[17] A weak instantiation is called abstract, if it has a term algebra $T_\Sigma(Y)$ as algebra part (see Definition 4.3.1).

[18] In a concrete instantiation all transition assignments are consistent transition assignments.

all components in the production is a Σ-$Wave$-term algebra $T_{\Sigma\text{-}Wave}(X_\varrho)$ with $X_{\varrho,user} = \{u_1, u_2\}$, $X_{\varrho,wavelet} = \{w_1, w_2, w_3\}$, $X_{\varrho,text} = \{t\}$ and $X_{\varrho,nat} = \{p\}$.

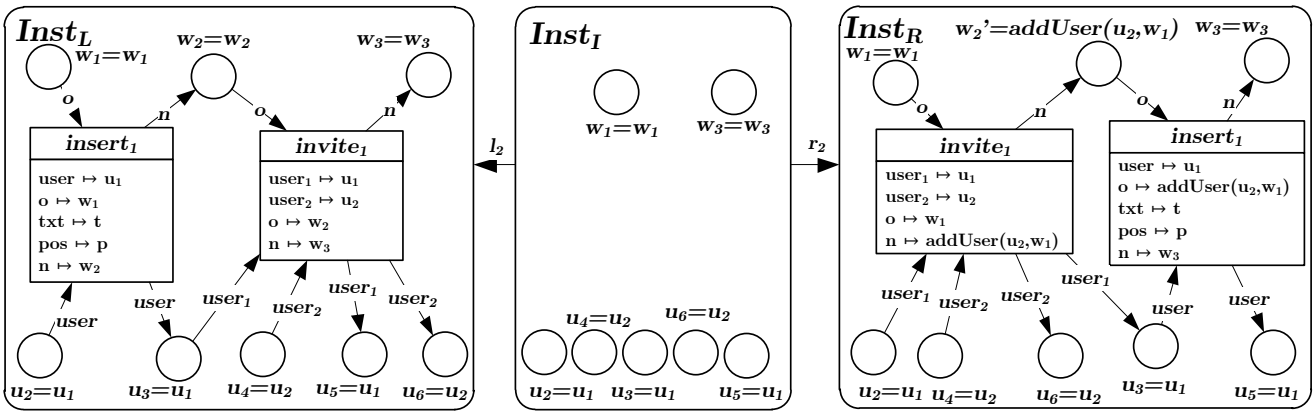


Figure 4.18: Abstract production for instantiations

As pointed out in Example 4.2.14, the production ϱ_2 is a pattern for changing the order of subsequent occurrences of *insert* and *invite user* actions (see Figure 3.2 on page 35). In our abstract production ϱ for instantiations, we enrich this pattern with information about what happens with the data when we permute the actions. Considering the $invite_1$ transition, the given information is rather superfluous: The user determined by the arc inscription $user_1$ invites the user determined by arc inscription $user_2$ into the wavelet bound by arc inscription o. The users do not change from left- to right-hand side, but the wavelet is a different one, since in the left-hand side we already inserted some text into the wavelet, while this did not yet happen in the right-hand side due to the changed order of actions.

Now, considering the transition $insert_1$, the information about the data is quite important, since the transition consists of two variables *txt* and *pos* in its firing conditions that do not occur as arc inscriptions in the environment of the transition. Therefore the values of these variables are not determined by values for surrounding places, and thus, it is possible to change the values for these variables – effectively changing the inserted text or its position. With the assignments given for transitions $insert_1$ in the left- and right-hand side we specify that we do not wish to change the inserted text or its position, but we want to insert exactly the same text t at exactly the same position p, regardless of the order of actions.

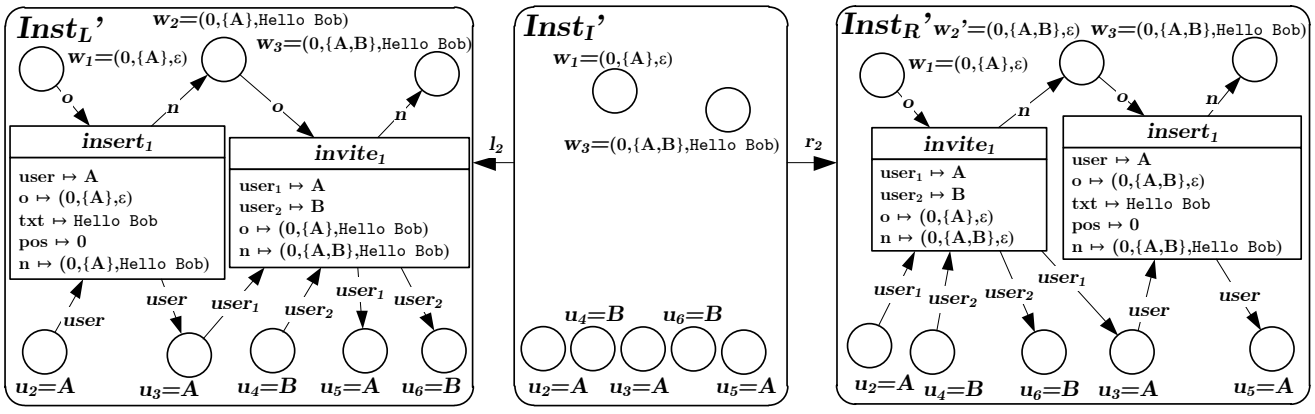


Figure 4.19: Concrete production for instantiations

Furthermore, a concrete production ϱ' for instantiations is depicted in Figure 4.19. The

AHL-net part of this production is also the production ϱ_2 in Figure 4.6, but instead of the term algebra it has the algebra A in Table 3.2 as data type part. While the left-hand side of the abstract production ϱ specifies the abstract case that some user u_1 inserts some arbitrary text into a wavelet and afterwards invites another user u_2, in the left-hand side of the concrete production ϱ' we specify that the user Alice (A) inserts the text "`Hello Bob`" into an empty wavelet and afterwards invites the user Bob (B). In the right-hand side of the production ϱ', Alice does the same but in a different order, i. e. first she invites Bob and afterwards she inserts the text "`Hello Bob`". ♢

Since in general the components of a production for instantiations are weak instantiations, we perform direct transformations as double-pushouts in the category of weak instantiations. For the transformation of concrete instantiations it is then additionally required that the original, context net and result are concrete instantiations.

Definition 4.6.3 (Direct Transformation of (Weak) Instantiations)**.** Given a production for instantiations $\varrho : Inst_L \xleftarrow{l} Inst_I \xrightarrow{r} Inst_R$, a weak instantiation $Inst$ and a (match) morphism $m : Inst_L \to Inst$. Then a *direct transformation of weak instantiations* $Inst \overset{(\varrho,m)}{\Longrightarrow} Inst'$ in **wInst** is given by pushouts (1) and (2) in Figure 4.20 in the category **wInst** of weak instantiations. We say that $Inst \overset{(\varrho,m)}{\Longrightarrow} Inst'$ is a *direct transformation of (concrete) instantiations* if $Inst$, $Inst_0$ and $Inst'$ are concrete instantiations. △

$$\begin{array}{rccccc} \varrho: & Inst_L & \xleftarrow{l} & Inst_I & \xrightarrow{r} & Inst_R \\ & {\scriptstyle m}\downarrow & (1) & {\scriptstyle c}\downarrow & (2) & {\scriptstyle n}\downarrow \\ & Inst & \xleftarrow{d} & Inst_0 & \xrightarrow{e} & Inst' \end{array}$$

Figure 4.20: Direct transformation $Inst \Rightarrow Inst'$ using production ϱ at match m

Remark 4.6.4 (Direct Transformation of (Weak) Instantiations). Note that the notion of weak instantiations is the broadest notion of instantiations. Abstract and concrete instantiations can be both considered as weak instantiations. Further, the categories **wInst** of weak instantiations and **Inst** of concrete instantiations are both $\mathcal{M}$-adhesive categories (see Fact A.5.9 on page 211), and pushouts of weak and concrete instantiations can be constructed via pushout of the underlying AHL-nets. Therefore, it is easy to see that the direct transformation of weak instantiations exists if and only if the direct transformation of underlying AHL-nets via the AHL-net part of the production exists, as shown in Fact A.5.11 on page 213.

Moreover, a direct transformation of a concrete instantiation using a concrete production means that all objects in Figure 4.20 are concrete instantiations, and thus, since **Inst** is a full subcategory of **wInst**, in that case diagrams (1) and (2) are pushouts in **Inst**. Hence, also the direct transformation of concrete instantiations using a concrete production exists if and only if the underlying transformation of AHL-nets exists, as shown in Fact A.5.12 on page 214.

However, the most interesting case of transformation is the one where we consider the direct transformation of a concrete instantiation using an abstract production. We use the *data-shifting* of abstract productions defined in the following to "instantiate" the abstract production, leading to a production with the same data type part as the matched concrete instantiation. △

In the following we define the data-shifting of abstract productions for instantiations. An abstract production can be shifted along a match with the effect of replacing the original data

type part of the production in the sense of the data-image construction for instantiations (see Definition 4.4.1) along the generalised homomorphism part of the match.

Definition 4.6.5 (Data-Shifting of Abstract Productions for Instantiations)**.** Given an abstract production for instantiations $\varrho : Inst_L \overset{l}{\leftarrow} Inst_I \overset{r}{\rightarrow} Inst_R$ and a match $m : Inst_L \to Inst$. We call a production for instantiations $\varrho' : Inst'_L \overset{l'}{\leftarrow} Inst'_I \overset{r'}{\rightarrow} Inst'_R$ together with $m' : L' \to Inst$ the *data-shifting of* ϱ *along* m, if

1. $Inst_L$, $Inst_I$ and $Inst_R$ are data-images of $Inst_L$, $Inst_I$ and $Inst_R$, respectively, along the data type part of m,
2. diagrams (1) and (2) in Figure 4.21 are pushouts in **wInst**,
3. diagram (3) in Figure 4.21 commutes, and
4. the data type part of m' is an isomorphism.

△

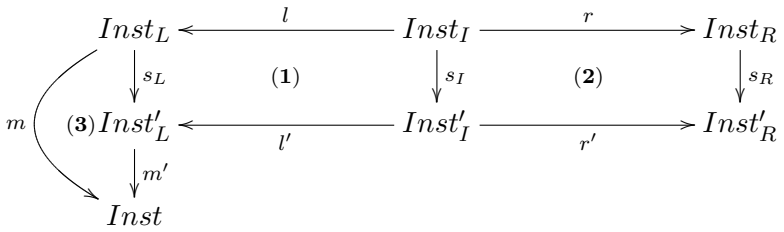


Figure 4.21: Data-shifting of abstract production

Note that the data-shifting of abstract productions exists along all matches, as shown in Fact A.5.16 on page 216. Moreover, for every direct transformation of weak instantiations $Inst \overset{\varrho,m}{\Longrightarrow} Inst'$ and data-shifting ϱ' with match m' along m, there exists a corresponding direct transformation of weak instantiations $Inst \overset{\varrho',m'}{\Longrightarrow} Inst'$, as shown in Lemma A.5.17 on page 217.

Example 4.6.6 (Data-Shifting of Abstract Productions for Instantiations)**.** For the abstract production ϱ for instantiations in Figure 4.18 we can find a match $m : Inst_L \to Inst$ into the instantiation $Inst$ in Figure 4.1 on page 72 that is an inclusion on the AHL-net part, i. e. we map the left-hand side to the environments of $insert_1$ and $invite_1$ in $Inst$. The data-shifting of ϱ along m is given by the production ϱ' in Figure 4.19.

The left-hand side $Inst'_L$ coincides exactly with the image of m. Note that this is only the case because m is an injective morphism. In general, for each non-injectively matched element, $Inst'_L$ contains a copy of its image in m. Further, $Inst'_I$ is the restriction of $Inst'_L$ to the AHL-net part I of $Inst_I$. This is always the case, because according to the definition of data-shifting there are morphisms $s_L : Inst_L \to Inst'_L$ and $s_I : Inst_I \to Inst'_I$, such that $Inst'_I$ is a pushout complement s_L and l. As pointed out in Fact A.5.11, the pushout complement of (weak) instantiations can be obtained as restriction along the corresponding AHL-part.

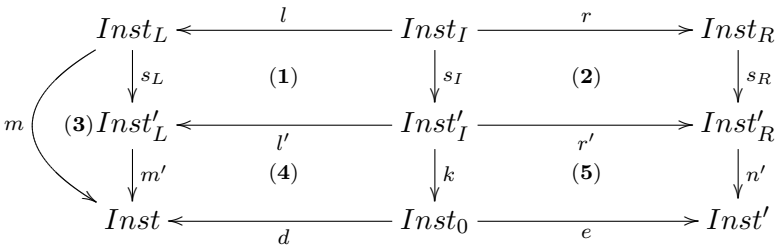
The right-hand side $Inst'_R$ contains some new values that are obtained by evaluation of the terms occurring in $Inst_R$ using the data type part of m. In the right-hand side, Alice (A) invites Bob (B) first, and then inserts the text "`Hello Bob`" into the empty wavelet where Bob is now invited. ♢

While abstract productions specify abstract patterns for modifications of instantiations, these abstract patterns can be *instantiated* to a given instantiation using the data-shifting along a match. This allows us to formulate the following instantiation condition which is satisfied if the result of the data-shifting is a concrete production. Then in the following Theorem 4.6.8 we show that the instantiation condition is a sufficient and necessary condition for the direct transformation of instantiations using abstract productions.

Definition 4.6.7 (Instantiation Condition for Abstract Productions)**.** Given an abstract production for instantiations $\varrho : Inst_L \overset{l}{\leftarrow} Inst_I \overset{r}{\rightarrow} Inst_R$, a concrete instantiation $Inst$ and a match $m : Inst_L \to Inst$. We say that ϱ and m satisfy the *instantiation condition for abstract productions* if the data-shifting of ϱ along m is a concrete production. △

Theorem 4.6.8 (Direct Transformation of Concrete Instantiations Using Abstract Productions)**.** *Given an abstract production for instantiations $\varrho : (inst_L, L) \overset{l}{\leftarrow} (inst_I, I) \overset{r}{\rightarrow} (inst_R, R)$, an instantiation $(inst, AN)$ and a (match) morphism $m : (inst_L, L) \to (inst, AN)$. There exists a direct transformation of instantiations $(inst, AN) \overset{(\varrho,m)}{\Longrightarrow} (inst', AN')$, iff ϱ and m satisfy the instantiation condition for abstract productions and there exists a direct transformation of AHL-nets $AN \overset{Net(\varrho),m}{\Longrightarrow} AN'$ using the underlying production for AHL-nets $Net(\varrho) : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$.*

Given that ϱ is a production for instantiated AHL-process nets and we have a match $m : (inst_L, L) \to (inst, K)$ into an instantiated AHL-process net. Then there exists a direct transformation of instantiated AHL-process nets $(inst, K) \overset{(\varrho,m)}{\Longrightarrow} (inst', K')$ iff ϱ and m satisfy the instantiation condition for abstract productions and there exists a direct transformation of AHL-process nets $K \overset{Net(\varrho),m}{\Longrightarrow} K'$ using the underlying production for AHL-process nets $Net(\varrho) : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$. Accordingly, the extension to AHL-processes has the same requirements as specified in Theorem 4.2.11.



Proof-Idea. As shown in Fact A.5.16, the data-shifting ϱ' of ϱ along m with pushouts (1) and (2) above and match $m' : (inst'_L, L') \to (inst, AN)$ exists. So, let ϱ and m satisfy the instantiation condition and let there be a direct transformation $AN \overset{Net(\varrho),m}{\Longrightarrow} AN'$ using the underlying production for AHL-nets $Net(\varrho) : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$. Then according to Fact A.5.12 there exists a direct transformation of weak instantiations $(inst, AN) \overset{\varrho,m}{\Longrightarrow} (inst', AN')$, and by Lemma A.5.17 there is also a direct transformation $(inst, AN) \overset{\varrho',m}{\Longrightarrow} (inst', AN')$. Then from the fact that ϱ' is a concrete production and by Fact A.5.12 we obtain that $(inst, AN) \overset{\varrho',m}{\Longrightarrow} (inst', AN')$ is a direct transformation of concrete instantiations, leading to a direct transformation $(inst, AN) \overset{\varrho,m}{\Longrightarrow} (inst', AN')$ by pushout composition.

Now, the other way around, given a direct transformation of concrete instantiations $(inst, AN) \overset{\varrho,m}{\Longrightarrow} (inst', AN')$ with pushouts (1)+(4) and (2)+(5), we obtain pushouts (4) and (5) by pushout decomposition which are also pullbacks due to the fact that $l', r' \in \mathcal{M}_{AHL}$,

because $l, r \in \mathcal{M}_{AHL}$ and $\mathcal{M}_{AHL}$-morphisms are closed under pushouts. This means that l' and r' and also m' have an isomorphic data type part, and by underlying pushouts and pullbacks in **Algs** we can derive that all morphisms in (4) and (5) have an isomorphic data type part. Hence, according to Fact A.6.9, the morphisms m', k and n' are W-creations, leading to concrete instantiations $Inst'_L$, $Inst'_I$ and $Inst'_R$ (see Remark 4.3.8).

The proof for instantiated AHL-process nets works completely similar due to the fact that direct transformations of AHL-process nets are also direct transformations of AHL-nets. For the detailed proof see Section B.8 on page 250. □

Concept 4.6.9 (Structural Evolution of Concrete Scenarios)**.** In Concept 3.2.16 we discussed the structural evolution of communication platforms that can be modelled using rule-based transformation of the corresponding AHL-nets. As the structural evolution of communication platforms involves adding, removing or changing resources and actions, accordingly it is possible that a scenario evolves by adding, removing or changing occurrences of the resources and actions in the scenario. The structural evolution of concrete scenarios can be modelled using direct transformations of the corresponding instantiated AHL-process (see Concept 4.3.6). An example of the structural evolution of a concrete scenario is given in Example 4.6.10, where we change the order of two subsequent actions occurring in the concrete scenario presented in Example 4.3.7. △

Example 4.6.10 (Structural Evolution of Concrete Scenarios)**.** Consider the concrete scenario *Inst* in Figure 4.8 on page 72 and the abstract production for instantiations shown in Figure 4.18. As described in Example 4.6.6, there is a match $m : Inst_L \to Inst$, matching the left-hand side of ϱ to the environments of $insert_1$ and $invite_1$ in *Inst*. The data-shifting ϱ' of ϱ along m shown in Figure 4.19 is a concrete production. This means that ϱ and m satisfy the instantiation condition.

Moreover, the AHL-net part $Net(\varrho)$ of ϱ is exactly the production ϱ_2 for AHL-process nets in Figure 4.6 on page 69. As presented in Example 4.2.14, there is a direct transformation $Wave \overset{\varrho_2,m_2}{\Longrightarrow} Wave_2$, and we have that *Wave* is the AHL-net part of *Inst* and m_2 corresponds exactly to the match morphism m. Hence, according to Theorem 4.6.8 there is also a corresponding direct transformation $Inst \overset{\varrho,m}{\Longrightarrow} Inst_2$ with the concrete instantiation $Inst_2$ shown in Figure 4.22 as result.

The direct transformation changes the order of the subsequent occurrences of the *insert* and *invite user* action occurrences in the instantiation. While in the instantiation *Inst* the empty wavelet is first changed to contain the text "`Hello Bob`" and afterwards the user Bob is invited to that wavelet, in the instantiation $Inst_2$, first the user Bob is invited to the wavelet that is still empty, and afterwards the text is inserted into that wavelet.

Note that there are also corresponding AHL-processes $wave : Wave \to Platform$ and $wave_2 : Wave_2 \to Platform$ with a direct transformation of AHL-processes $wave \overset{\varrho_2,m_2}{\Longrightarrow} wave_2$. From obvious combination of the direct transformations of the processes and instantiations we obtain a direct transformation of instantiated AHL-processes $(inst, wave) \overset{\varrho,m}{\Longrightarrow} (inst_2, wave_2)$. ♢

As the example suggests, for the data-shifting of an abstract production along a match with concrete instantiation as codomain, it is always the case that the left-hand side and interface of the data-shifting are concrete instantiations. This is quite reasonable, since for the modification of a concrete instantiation, it should not be necessary to check whether the original was in fact a concrete instantiation. Instead, it should be sufficient to check only the newly created parts of the instantiation, if these parts are consistent with the given instantiation. This is expressed in the following *consistent creation condition* which is also a

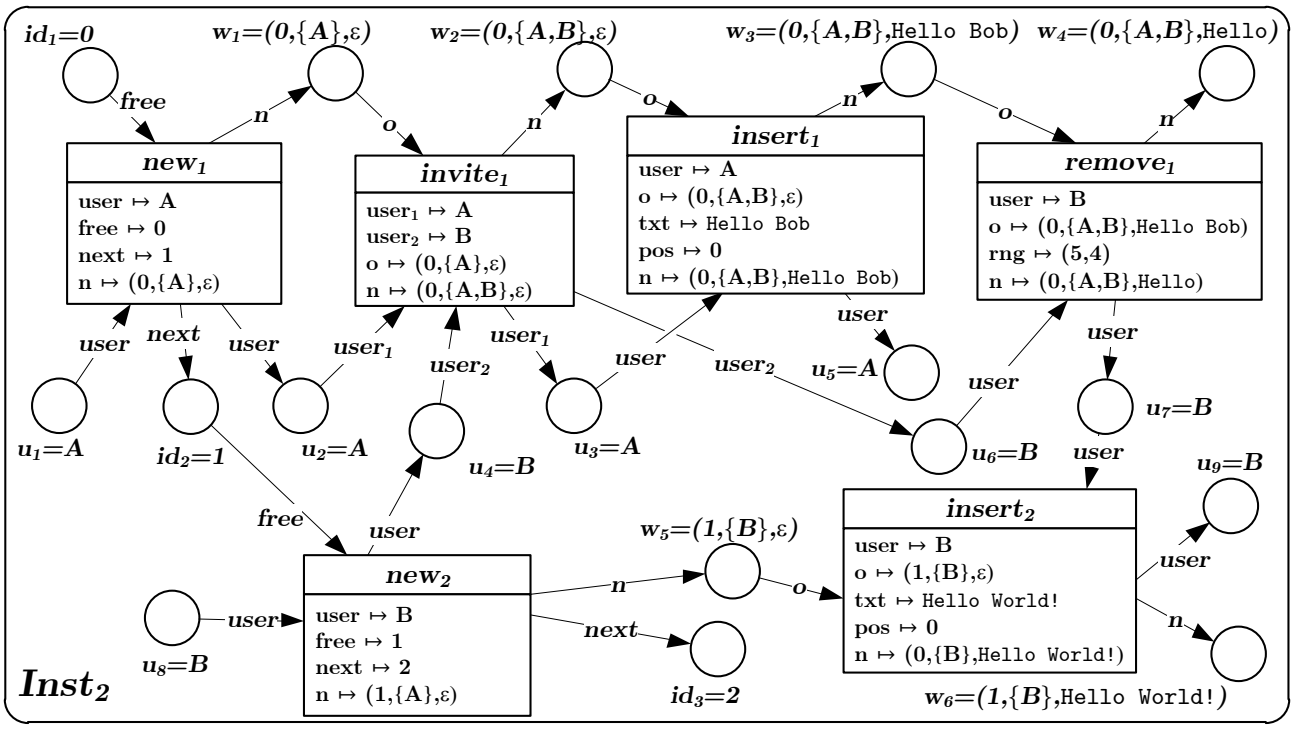


Figure 4.22: Concrete instantiation $Inst_2$

sufficient and necessary condition for the direct transformation of instantiations, since it is equivalent to the instantiation condition.

Definition 4.6.11 (Consistent Creation Condition for Abstract Productions)**.** Given an abstract production for instantiations $\varrho : (inst_L, L) \overset{l}{\leftarrow} (inst_I, I) \overset{r}{\rightarrow} (inst_R, R)$, where the data type part of l and r are identities, a concrete instantiation $(inst, AN)$ and a match $m : (inst_L, L) \to (inst, L)$ with $m = (m_\Sigma, m_P, m_T, m_A)$. We say that ϱ and m satisfy the *consistent creation condition* if for all transitions $t \in T_R \setminus r_T(T_I)$ and $inst_{R,T}(t) = (t, v)$ we have:[19]

$$m_A \circ v \circ (m_X|_{Var(t)})^{-1} \vDash \mathcal{P}_{fin}(m_\Sigma^\#)(cond_R(t)),$$

i. e. all equations in $\mathcal{P}_{fin}(m_\Sigma^\#)(cond_R(t))$ are valid under assignment $m_A \circ v \circ (m_X|_{Var(t)})^{-1}$. △

Remark 4.6.12 (Consistent Creation Condition). Note that if the production already has the right signature part, i. e. if m_Σ is an identity, then the consistent creation condition becomes much simpler to verify. Then for all transitions $t \in T_R \setminus r_T(T_I)$ and $inst_{R,T}(t) = (t, v)$ we require that $m_A \circ v \vDash cond_R(t)$.

Moreover, the consistent creation condition can be generalised to productions where the morphisms l and r do not have an identical data type part. Since the morphisms are required to be $\mathcal{M}_{AHL}$-morphisms, we achieve the same result (see Fact 4.6.13 below) by replacing m_A in the condition with $m_A \circ l_A \circ r_A^{-1}$, and similar replacements of m_X and m_Σ. △

Fact 4.6.13 (Equivalence of Consistent Creation Condition and Instantiation Condition)**.** *Given an abstract production for instantiations* $\varrho : (inst_L, L) \overset{l}{\leftarrow} (inst_I, I) \overset{r}{\rightarrow} (inst_R, R)$,

[19] Note that m_X is the variable part of **SigSets**-morphism $m_\Sigma = (m_\sigma, m_X)$ (see Definition 3.1.5).

where the data type part of l and r are identities, a concrete instantiation $(inst, AN)$ and a match $m : (inst_L, L) \to (inst, AN)$. Then ϱ and m satisfy the consistent creation condition if and only if they satisfy the instantiation condition.

Proof. Given the data-shifting ϱ' of ϱ along m, it can be shown that $(inst, AN)$ being a concrete instantiation is sufficient for $(inst'_L, L')$ and $(inst'_I, I')$ to be concrete instantiations as well. Moreover, we can show that the consistency of assignments in $(inst'_R, R')$ that correspond to images of transitions in $T_R \setminus r_T(T_I)$ follows from the consistent creation condition, and vice versa. Finally, the consistency of all other transition assignments can be derived from consistency of the corresponding preimages in $(inst'_I, I')$. For the detailed proof we refer to Section B.9 on page 251. □

4.7 Modelling and Evolution of Histories

In this section we consider the modelling of histories of waves (see Section 2.2), and also the modelling of the progression of histories, i. e. the evolution of histories based on interactions.

Definition 4.7.1 (Input and Output Places of Instantiated AHL-Process Net)**.** Given an instantiation $Inst = (inst, K)$ with $inst : Skel(K) \to Flat(K)$, we define the sets of input places $IN(Inst) = inst_P(IN(K))$ and output places $OUT(Inst) = inst_P(OUT(K))$ (see Definition 4.1.3 for definition of the input and output places of AHL-process nets). △

The following fact states the correspondence between instantiations and firing sequences in the codomain and domain of the corresponding AHL-process. The fact is based on the corresponding results in [Ehr05] for AHL-occurrence nets, and the preservation of firing behaviour by AHL-morphisms.

Fact 4.7.2 (Finite Strict AHL-Processes, Instantiations and Firing Behaviour)**.** *Given a finite strict AHL-process $mp : K \to AN$. We call a firing sequence in K* complete*, if it visits each transition in K exactly once. Then, we have the following:*

1. *For each complete firing sequence $s : M \to^* M'$ in K, there is an instantiation $Inst = (inst, K)$ with $IN(Inst) = M$ and $OUT(Inst) = M'$, and for each firing step $M_n \xrightarrow{(t,v)} M_{n+1}$ in s, there is $inst_T(t) = (t, v)$.*

2. *For each instantiation $Inst = (inst, K)$ and place $(a, p) \in inst_P(P_K)$ respectively transition $(t, v) \in inst_T(T_K)$, there is at least one complete firing sequence $s : IN(Inst) \to^* OUT(Inst)$ in K that visits (a, p) respectively (t, v) exactly once, where the order of transitions in s is compatible with the causal relation $<_K$.*

3. *For each firing sequence $s : M_0 \xrightarrow{(t_0,v_0)} M_1 \xrightarrow{(t_1,v_1)} \cdots \xrightarrow{(t_n,v_n)} M_{n+1}$ in K there is a corresponding firing sequence $mp(s) : (id \otimes mp_P)^{\oplus}(M_0) \xrightarrow{(f_T(t_0),v_0)} (id \otimes mp_P)^{\oplus}(M_1) \xrightarrow{(f_T(t_1),v_1)} \cdots \xrightarrow{(f_T(t_n),v_n)} (id \otimes mp_P)^{\oplus}(M_{n+1})$ in AN.*

Proof. 1. Since the AHL-process $mp : K \to AN$ is finite, the AHL-process net K is also an AHL-occurrence net. Therefore, the existence of the corresponding instantiation for a complete firing sequence in K follows from Fact 3.5 in [Ehr05].

2. Analogously, from Fact 3.5 we obtain a complete firing sequence $s : IN(Inst) \to M$ that corresponds to instantiation $Inst$. Since K is finite, from Fact 3.6 in [Ehr05] we obtain that $M = OUT(Inst)$.

3. This is a direct consequence of the fact that AHL-morphisms preserve firing behaviour (Fact 3.1.10), taking into account that mp is strict, and thus, (mp_Σ, mp_A) is an identity. □

Concept 4.7.3 (Modelling of Histories)**.** A history of a wave is a description of previous states and the current state of the wave as well as of all interactions that leaded from one state to another one. So, basically a history is a finite concrete scenario that uses the actual data types and values of the corresponding platform. Accordingly, a history can be modelled as a finite and strict instantiated AHL-process. Given a history $wave : (inst, Wave) \to Platform$, the current state of the wave can be obtained as the marking $OUT(inst, Wave)$, i. e. the output places of the AHL-process net $Wave$ equipped with their corresponding data values. △

Example 4.7.4 (Modelling of Histories)**.** The concrete scenario model $wave : (inst, Wave) \to Platform$ in Example 4.3.7 on page 72 is finite and can be seen as the history of a wave. The initial state of the wave can be derived from the input places:

$$IN(inst, Wave) = (0, id_1) \oplus (A, u_1) \oplus (B, u_8)$$

which means that initially there are only two users A (Alice) and B (Bob) and a free id 0. Outgoing from this state, the history describes that first a new wavelet with id 0 was created by Alice. Afterwards, Alice inserted the text "`Hello Bob`" into the wavelet, and Bob created another wavelet with id 1. The fact that the corresponding transitions in the AHL-process net can fire independently, means that these two actions occurred simultaneously, or, if we consider a near-real-time system such as Apache Wave (see Section 2.2), it is possible that one action occurred shortly before or after the other one, but we still consider this as simultaneous. After that, user Alice invited Bob to the first wavelet, who then removed the text "` Bob`" from that wavelet. Finally, Bob inserts the text "`Hello World!`" to the wavelet with id 1. The resulting current state of the wave can be derived from the output places:

$$\begin{aligned} OUT(inst, Wave) =& (2, id_3) \oplus (A, u_5), \oplus (B, u_9) \\ & \oplus ((0, \{A, B\}, \texttt{Hello}), w_4) \oplus ((1, \{B\}, \texttt{Hello World!}), w_6) \end{aligned}$$

This means that the next free id is 2, and the wave now has two wavelets, one where Alice and Bob are invited, and another one where only Bob is invited. ♢

In order to model the progression of histories based on interactions, we need a way to append a scenario of interactions to the end of a history. In [EHGP09], the sequential composition of AHL-occurrence nets was defined. We transfer that definition to the sequential composition of AHL-process nets. Sequential composition means that at most output places of one net are glued to input places of the other one.

Definition 4.7.5 (Sequential and Parallel Composition of AHL-Processes)**.** Given three AHL-process nets I, K_1 and K_2, and two injective AHL-net morphisms $i_1 : I \to K_1$ and $i_2 : I \to K_2$ with $i_2 \in \mathcal{M}_{AHL}$. Then (K_1, K_2) *are sequentially composable w. r. t.* (I, i_1, i_2) if $T_I = \emptyset$, $i_1(P_I) \subseteq OUT(K_1)$ and $i_2(P_I) \subseteq IN(K_2)$.

If (K_1, K_2) are sequentially composable w. r. t. (I, i_1, i_2), then the gluing $K_3 = K_1 +_I K_2$ is called a *sequential composition*, and if $i_2(P_I) = IN(K_2)$, then the composition is called *strictly sequential*. Moreover, if I is an empty net, then the composition is called a *parallel composition*.

Given AHL-processes $mp_1 : K_1 \to AN$ and $mp_2 : K_2 \to AN$, then (mp_1, m_2) are *sequentially composable w. r. t.* (I, i_1, i_2) if (K_1, K_2) are sequentially composable w. r. t. (I, i_1, i_2) and $mp_1 \circ i_1 = mp_2 \circ i_2$. In this case also the composition of AHL-processes is called (strict) sequential or parallel, respectively. △

Fact 4.7.6 (Sequential and Parallel Composition of AHL-Processes)**.** *Given three AHL-process nets I, K_1 and K_2, and two injective AHL-net morphisms $i_1 : I \to K_1$ and $i_2 : I \to K_2$ with $i_2 \in \mathcal{M}_{AHL}$, such that (K_1, K_2) are sequentially composable w. r. t. (I, i_1, i_2). Then (K_2, K_1) are composable w. r. t. (I, i_2, i_1).*

Proof. Let (K_1, K_2) be sequentially composable w. r. t. (I, i_1, i_2). We have that $i_2 \in \mathcal{M}_{AHL}$, and it remains to show that the four composability requirements are satisfied:

1. *(No cycles)* For $p_1, p_2 \in P_I$ we have $i_1(p_1), i_1(p_2) \in OUT(K_1)$ which means that both of the images do not have a successor and thus $i_1(p_1) \not<_{K_1} i_1(p_2)$. Analogously, we have $i_2(p_1), i_2(p_2) \in IN(K_2)$ which means that these images do not have a predecessor and thus $i_2(p_1) \not<_{K_2} i_2(p_2)$. Hence, the induced causal relation $<_{(i_1,i_2)}$ (see Definition 4.2.4) is empty, and therefore it is a strict partial order.

2. *(Non-injective gluing)* This requirement is trivially satisfied, because i_1 is injective.

3. *(No conflicts)* For all $p \in P_I$ the implication $i_2(p) \notin IN(K_2) \Rightarrow i_i(p) \in IN(K_1)$ holds, because its premise is forbidden by sequential composability. Moreover, for all $p \in P_I$ the implication $i_2(p) \notin OUT(K_2) \Rightarrow i_1(p) \in OUT(K_1)$ holds, because its conclusion is satisfied by sequential composability. □

For instantiated AHL-processes it is shown in Fact 4.5.8 it is shown that the gluing exists if and only if the underlying AHL-processes are composable. Moreover, as shown in Fact 4.7.6, the parallel and sequential composition of AHL-processes is just a special case of the composition of AHL-processes. Therefore, the notions of parallel and sequential composition of instantiated AHL-processes follow directly from these results: A composition of instantiated AHL-processes can be called *parallel or sequential composition of instantiated AHL-processes*, if the underlying composition of AHL-process nets is parallel respectively sequential.

Note that the composition of AHL-processes is a special case of a non-deleting direct transformation of AHL-processes. Due to the restriction that the interface I of a sequential composition does not have any transitions, and the requirement that for a strictly sequential composition we also have that all places in I are exactly the input places of the posterior net, for every AHL-process $mp : K \to AN$, we can define a production for AHL-process nets, such that the application of that production with a suitable match corresponds exactly to the strictly sequential composition, where we append AHL-process mp to another one:

Definition 4.7.7 (Sequential Production for AHL-Process)**.** Given an AHL-process $mp : K \to AN$, the *sequential production $\vec{mp}$ of mp* is defined by a production for AHL-processes $\vec{mp}$: $(mp_I \xleftarrow{id} mp_I \xrightarrow{r} mp)$, where $mp_I : I \to AN$ is defined by

- $I = (\Sigma_K, IN(K), \emptyset, \emptyset, \emptyset, \emptyset, type_I, A_K)$ with $type_I = type_K|_{IN(K)}$,
- $r : I \hookrightarrow K \in \mathcal{M}_{AHL}$ is an inclusion, and
- $mp_I = mp \circ r$.

For an AHL-process $mp_1 : K_1 \to AN$, an injective morphism $m : mp_I \to mp_1$ is called *sequential match for production $\vec{mp}$*, if $m(P_I) \subseteq OUT(K_1)$. △

Fact 4.7.8 (Sequential Transformation of AHL-Processes)**.** *Given an AHL-process $mp : K \to AN$ with sequential production $\vec{mp}$: $(mp_I \xleftarrow{id} mp_I \xrightarrow{r} mp)$, and AHL-process $mp_1 : K_1 \to AN$, and a sequential match $m : mp_I \to mp_1$. Then there exists a direct transformation of AHL-processes $mp_1 \overset{\vec{mp},m}{\Longrightarrow} mp_2$.*

Proof. Since the left-hand side of the production $\overrightarrow{mp}$ is an identity, it suffices to show that the gluing of mp and mp_1 along r and m exists. Due to the fact that all morphisms are **Proc**(**AN**)-morphisms, we already have compatibility of the AHL-processes, and it only remains to show that the gluing of AHL-process nets K and K_1 along r and m exists. By definition of $\overrightarrow{mp}$ we have $r(P_I) = IN(K)$, and since m is a sequential match, we also have $m(P_I) \subseteq OUT(K_1)$. This means that (K, K_1) are (strictly) sequentially composable w. r. t. (I, r, m), and thus, by Fact 4.7.6 and Fact 4.2.6, we have that the composition of $K_2 = K +_{(I,r,m)} K_1$ exists. Hence, there is a direct transformation $mp_1 \overset{\overrightarrow{mp},m}{\Longrightarrow} mp_2$. □

The construction of sequential productions for AHL-processes above can be extended to a corresponding construction for instantiated AHL-processes, and we obtain similar result.

Definition 4.7.9 (Sequential Production for Instantiated AHL-Process)**.** Given a weakly instantiated AHL-process $mp : (inst, K) \to AN$, the *sequential production* $\overrightarrow{mp}$ *of* mp is defined by a production for instantiated AHL-processes $\overrightarrow{mp}$: $(mp_I \xleftarrow{id} mp_I \xrightarrow{r} mp)$, where $mp_I : (inst_I, I) \to AN$ is defined by

- I and r are obtained by construction of sequential production $\overrightarrow{mp'}$ for AHL-process $mp' = I \to AN$ (see Definition 4.7.7), and
- $(inst_I, I)$ is the restriction of $(inst, K)$ along r (see Definition 4.5.3).

For an instantiated AHL-process $mp_1 : (inst_1, K_1) \to AN$, an injective morphism $m : mp_I \to mp_1$ is called *sequential match for production* $\overrightarrow{mp}$, if $m(P_I) \subseteq OUT(K_1)$. △

Fact 4.7.10 (Sequential Transformation of Instantiated AHL-Processes)**.** *Given an instantiated AHL-process* $mp : (inst, K) \to AN$ *with sequential production* $\overrightarrow{mp}$: $(mp_I \xleftarrow{id} mp_I \xrightarrow{r} mp)$*, and instantiated AHL-process* $mp_1 : (inst_1, K_1) \to AN$*, and a sequential match* $m : mp_I \to mp_1$*. Then there exists a direct transformation of instantiated AHL-processes* $mp_1 \overset{\overrightarrow{mp},m}{\Longrightarrow} mp_2$ *if and only if* $\overrightarrow{mp}$ *and* m *satisfy the instantiation condition.*

Proof. Similar to the proof of Fact 4.7.8, we already have compatibility of the AHL-processes, and it suffices to show that the direct transformation of the corresponding instantiated AHL-process nets exists. According to Theorem 4.6.8, the direct transformation of instantiated AHL-process net $(inst_1, K_1)$ exists if and only if $\overrightarrow{mp}$ and m satisfy the instantiation condition and the underlying direct transformation of AHL-process nets exists. From Fact 4.7.8 we know that the underlying direct transformation of AHL-process nets $K_1 \Rightarrow K_2$ always exists, and thus, the direct transformation $(inst_1, K_1) \Rightarrow (inst_2, K_2)$ exists if and only if $\overrightarrow{mp}$ and m satisfy the instantiation condition. □

Concept 4.7.11 (Modelling Progression of Histories and Behaviour of Users and Robots)**.** The progression of the history of a wave means that outgoing from the current state of the wave, we perform more interactions, extending the history of the wave by these interactions and leading to a new current state. This can be modelled by the sequential transformation of the instantiated AHL-process model of the history, using a finite sequential production that describes the newly performed interactions. The preconditions of a production is described by its left-hand side. In a sequential production, the left-hand side consists exactly of the input places of the process of interactions that is to be added. Moreover, since the sequential transformation requires a sequential match, it is ensured that the pre-conditions of the sequential production are existent in the output places of the progressing history – which describes exactly the current state of the wave, as pointed out in Concept 4.7.3.

A robot can be seen as a special kind of user that interacts with waves in an automated way. The behaviour of a user or a robot can be modelled as a set of sequential productions for instantiated AHL-processes. The left-hand side of the productions describe the pre-conditions under which the user can perform the corresponding interaction, and the right-hand side describes these interactions that are performed and added to the history of a wave. An example of the progression of a history based on the behaviour of a robot is given in the following Example 4.7.12. △

Example 4.7.12 (Progression of History based on the Behaviour of a Robot)**.** In this example we describe the progression of the history of a wave based on the behaviour of a simple robot that finds and corrects typographical errors (or short typos). Note that for the "installation" of a more complex robot on a platform, it may be necessary to evolve the data or structure of the platform. In the case of our typo robot, we extend the data type part of our running example *Platform* (see Table 3.1 on page 36 for the signature and Table 3.2 on page 37 for the algebra) by the signature Σ-*Typo* in Table 4.2 and corresponding algebra B in Table 4.3.

Table 4.2: Signature Σ-*Typo*

sorts:	text, typodict, user
opns:	typoText : typodict $\to$ text
	correctText : typodict $\to$ text
	typoRobot : $\to$ user

Table 4.3: Σ-*Typo*-algebra B

$A_{text} = \{a, \dots, z, A, \dots, Z, \dots\}^*$
$A_{typodict} = \{(teh, the), (THe, The), (adn, and), (of\ of, of), \dots\}$
$typoText_A : A_{typodict} \to A_{text}$
$(t, c) \mapsto t$
$correctText_A : A_{typodict} \to A_{text}$
$(t, c) \mapsto c$
$typoRobot_A = \mathrm{TR} \in A_{user}$

The extension of the data type part consists of a dictionary of commonly occurring typos together with their correction. Moreover, the extension contains a new user constant that identifies the typo robot in the platform. The algebras $(\Sigma\text{-}Wave, A)$ and $(\Sigma\text{-}Typo, B)$ can be integrated by pushout of algebras over their common parts, leading to a new algebra $(\Sigma\text{-}Wave', A')$ and also a generalised homomorphism $h : (\Sigma\text{-}Wave, A) \to (\Sigma\text{-}Wave', A')$. In the following all AHL-processes are processes over the data-image *Platform*$'$ of the running example *Platform* along h.

The behaviour of the typo robot is described by the single abstract sequential production ϱ_{typo} in Figure 4.23. The left-hand side of the rule contains a wavelet and a user place, where the user place is denoted with the constant that is bound to the typo robot in algebra A'. Further, note that the term on the wavelet place can be evaluated in A' to any wavelet that contains one of the typos in the dictionary and where the typo robot is invited.For simplicity the typo robot does a very naive detection of typos, whereas in practice it would be much better to use for instance regular expressions for the detection of typos. The right-hand side of the rule describes the performance of the deletion of the found typo and the insertion of the corresponding correct text by the typo robot itself.

Now, consider the instantiated AHL-process $Inst_{Typo}$ in Figure 4.24 that models a history of a wave that has a typo in it. In the wave, a user A (Alice) creates a new wavelet with

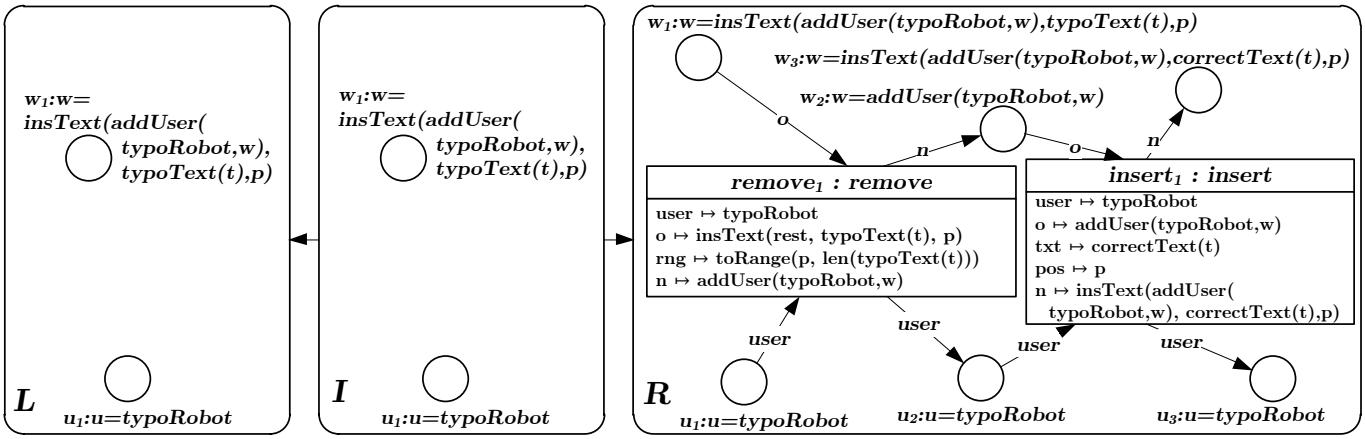


Figure 4.23: Sequential production ϱ_{typo} of the typo robot

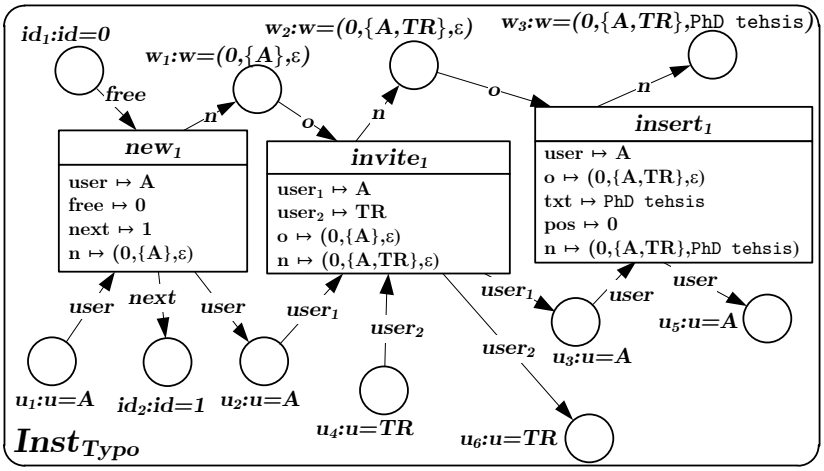


Figure 4.24: History $Inst_{Typo}$ of a wave with a typo

id 0, then invites the typo robot to that wavelet, and afterwards she inserts the incorrectly written text "`PhD tehsys`" to the wavelet. The resulting current state of the wave is that we have the two users Alice and the typo robot, and a wavelet with id 0, containing the text "`PhD tehsys`".

There is a sequential match $m : L \to Inst_{Typo}$ with $m(w_1) = w_3$ and $m(u_1) = u_6$. The data type part (m_Σ, m_A) with m_Σ being the identity on $\Sigma\text{-}Wave'$ and $m_A : T_{\Sigma\text{-}Wave'}(X_{Typo}) \to A'$ induces an assignment $v : X_{Typo} \to A'$ by restriction of m_A to the variables occurring in ϱ_{Typo}. The assignment v maps the variable w to the wavelet $(0, \{A\}, \texttt{PhD sys})$, the variable p to the natural number 4, and the variable t to the entry (teh, the) in the typo dictionary.[20]

There is a sequential transformation $Inst_{Typo} \Rightarrow Inst_{Correct}$ via ϱ_{Typo}, and the result is depicted in Figure 4.25. The data-image of the right-hand side along (m_Σ, m_A) is appended to the history, extending it by the deletion of the incorrect text part "`teh`" and insertion of the correction "`the`" at the same position. Consequently, we have a new current state of the corresponding wave, where we still have the two users Alice and the typo robot, but the wavelet with id 0 now contains the text "`PhD thesis`". Moreover, the correction of the typo

[20] Note that due to non-injectivity of the function $addUser_{A'}$ it is also possible to map the variable w to the wavelet $(0, \{A, TR\}, \texttt{PhD sys})$. However, this would not change anything in the example, since the variable w is always used in the context $addUser(typoRobot, w)$, having the same result for both choices of values for w.

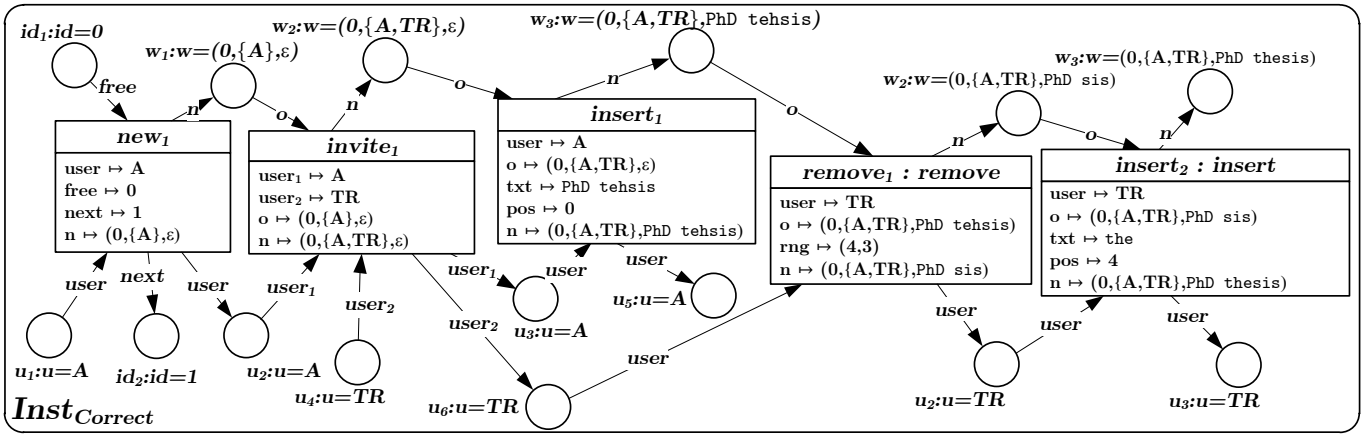


Figure 4.25: History $Inst_{Correct}$ of a wave where a typo was corrected

by occurrences of the *remove* and *insert* actions is now part of the history of the wave. ♢

4.8 Evolution of Scenarios Based on Platform Evolution

In Section 3.2 we presented the rule-based transformation of AHL-nets, and in Section 4.2 and Section 4.6 we presented the rule-based transformation of AHL-processes and instantiations. As pointed out in Concept 3.2.16, the transformation of AHL-nets can be used to model the evolution of communication platforms, and in Concept 4.2.13 and Concept 4.6.9 we described how transformation of (instantiated) AHL-processes can be used to model the evolution of abstract and concrete scenarios.

In this section we consider two ways to handle existing scenarios that correspond to a platform that is being modified. The extension of a scenario, on the hand, means that the unchanged scenario is still valid in the new modified platform. On the other hand, the scenario evolution based on platform is a modification of the scenario, where all changes of the platform are applied correspondingly to the scenario.

4.8.1 Extension of Scenarios

As mentioned in Section 2.3, it is possible that a communication platform is modified at runtime and there may already exist some scenarios that correspond to the old version of the platform. So we have the case that there is an AHL-process $wave : Wave \to Platform$ and a direct transformation of AHL-nets $Platform \Rightarrow Platform'$. In this section we show under which conditions the scenario $wave$ can be extended to a scenario $wave' : Wave \to Platform'$ of the new platform.

We regard $wave'$ as an extension of $wave$ if the two processes "agree" in the context net of the direct transformation $Platform \Rightarrow Platform'$ in the following sense.

Definition 4.8.1 (Extension of AHL-Process based on AHL-Net Transformation)**.** Given an AHL-net AN and an AHL-process $mp : K \to AN$. Let $AN \overset{o,m}{\Rightarrow} AN'$ be a direct transformation with pushouts (1) and (2) in **AHLNets** as depicted in Figure 4.26. Then we call $mp' : K \to AN'$ an *extension of* mp if there exists $mp_0 : K \to AN_0$ with $f \circ mp_0 = mp$ and $g \circ mp_0 = mp'$. △

Remark 4.8.2 (Extension of Instantiated AHL-Processes). Note that we do not explicitly consider instantiated AHL-processes in this section. However, the extension of an AHL-process as defined above leaves the domain of the process unchanged, and thus, an instantiated

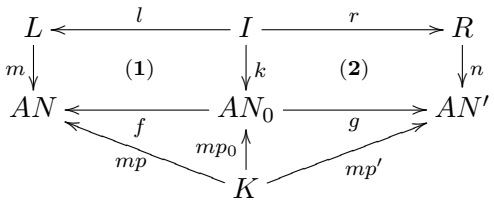


Figure 4.26: Extension of AHL-Process

AHL-process $(inst, wave)$ can be extended to to an instantiated AHL-process $(inst, wave')$, if and only if AHL-process $wave$ can be extended to $wave'$. △

The following extension condition is a sufficient and necessary condition for the extension mp' of an AHL-process mp based on an AHL-net transformation. In order to satisfy the extension condition, the transformation must not delete any place or transition that have an occurrence in the AHL-process mp.

Definition 4.8.3 (Extension Condition)**.** Given an AHL-net AN, an AHL-process $mp : K \to AN$ and a direct transformation $AN \overset{\varrho,m}{\Rightarrow} AN'$. We define the the set PP of process points as

$$PP = \{x \in P_L \mid \exists p \in P_K : mp_P(p) = m_P(x)\} \cup \{x \in T_L \mid \exists t \in T_K : mp_T(t) = m_T(x)\}$$

We say that mp and ϱ, m satisfy the extension condition if all process points are gluing points (see Definition 3.2.13), i. e. $PP \subseteq GP$. △

Theorem 4.8.4 (Extension of AHL-Process based on AHL-Net Transformation)**.** *Given an AHL-net AN, an AHL-process $mp : K \to AN$ and a direct transformation $AN \overset{\varrho,m}{\Rightarrow} AN'$ with pushouts (1) and (2) in* **AHLNets** *as depicted in Figure 4.26. There exists an extension $mp' : K \to AN'$ of mp if and only if mp and ϱ, m satisfy the extension condition.*

Proof-Idea. If the extension condition is satisfied, it can be explicitly shown that there exists a well-defined morphism $mp_0 : K \to AN_0$ defined by $mp_0 = f^{-1} \circ mp$. Then the extension $mp' : K \to AN'$ is obtained by composition $mp' = g \circ mp_0$, satisfying the required properties. Vice versa, the existence of an extension $mp' : K \to AN'$ implies the existence of a suitable morphism $mp_0 : K \to AN_0$ which can be used to show that all process points are gluing points. For a detailed proof see Section B.10 on page 253. □

Concept 4.8.5 (Extension of Scenarios based on Platform Evolution)**.** As pointed out in Section 2.4, due to the fact that platforms can evolve, we need a way to check if scenarios of the original platform are still in compliance with the result of the modification. Using our results from Theorem 4.8.4, we know that this can checked using the extension condition. If the extension condition is satisfied for a given scenario and a platform evolution, the scenario can be extended to a new scenario complying to the result of the platform evolution. According to Remark 4.8.2 this works for abstract scenarios as well as for concrete scenarios, because the extension of an instantiated AHL-processes does not change the corresponding AHL-process net or instantiation, and therefore, it coincides exactly with the extension of the corresponding AHL-process. △

Example 4.8.6 (Extension of Scenarios based on Platform Evolution)**.** Figure 4.27 shows an AHL-process $w : W \to Platform$ of our example platform presented in Example 3.1.9. In Example 3.2.17 we presented a direct transformation $Platform \Rightarrow Platform'$, using production ϱ shown in Figure 3.3 with inclusion match m. Considering this AHL-net transformation,

our set of process points PP contains all elements that have an occurrence in W. So, we have $PP = \{u, w\}$. According to Definition 3.2.13, the set of gluing points is $GP = \{u, w\}$, because u and w are preserved by ϱ. Hence, we have $PP \subseteq GP$ which means that w can be extended to a new AHL-process $w' : W \to \mathit{Platform}'$. The graphical representation of the new AHL-process corresponds exactly to the one depicted in Figure 4.27, due to the fact that the transformation $\mathit{Platform} \Rightarrow \mathit{Platform}'$ did not involve any renaming of elements.

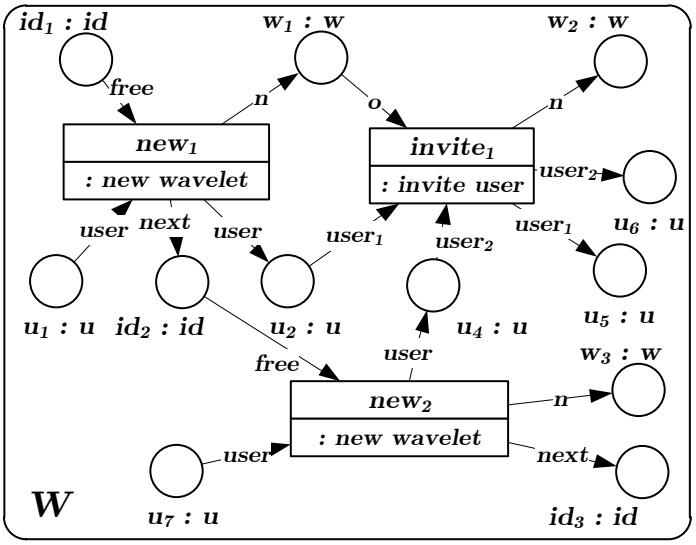


Figure 4.27: Extension of AHL-process

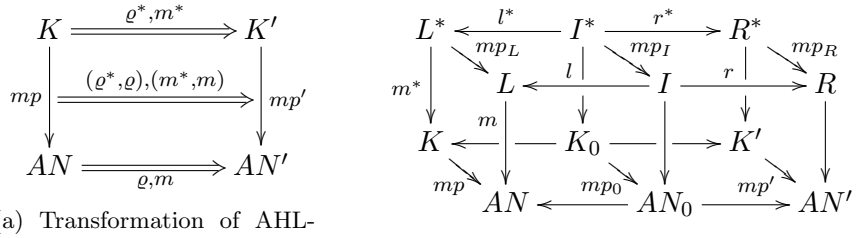
In Example 4.1.8 we introduced another AHL-process $\mathit{wave} : \mathit{Wave} \to \mathit{Platform}$ that is shown in Figure 4.1 on page 60. Since the AHL-process *wave* also contains occurrences of the actions *insert* and *remove* which are matched by m, the process points regarding this scenario are given by $PP = \{u, w, \mathit{insert}, \mathit{remove}\}$. This means that we have $PP \not\subseteq GP = \{u, w\}$ and hence, the scenario modelled by AHL-process *wave* cannot be extended to a scenario of the result of the platform evolution $\mathit{Platform}'$. ♢

4.8.2 Abstract Scenario Evolution Based on Platform Evolution

In [GE12a] we presented a general construction for the modification of scenarios based on a special kind of platform evolution, replacing one single action at a time. In this section we present a more general construction that allows for the replacement of multiple actions at a time. In order to formulate the reconfiguration of AHL-processes, we need productions and the direct transformation of AHL-processes.

Definition 4.8.7 (Transformation of AHL-Processes)**.** A *production for AHL-processes* is a span $(\varrho^*, \varrho) : mp_L \xleftarrow{(l^*,l)} mp_I \xrightarrow{(r^*,r)} mp_R$ of componentwise injective **AHLProcs**-morphisms with isomorphic data type parts as shown in the top of Figure 4.28b. We call $\varrho^* : L^* \xleftarrow{l^*} I^* \xrightarrow{r^*} R^*$ the *process part* and $\varrho : L \xleftarrow{l} I \xrightarrow{r} R$ the *system part* of (ϱ^*, ϱ).

Given a production $\varrho : mp_L \xleftarrow{(l^*,l)} mp_I \xrightarrow{(r^*,r)} mp_R$ for AHL-processes and a (match) morphism $(m^*, m) : mp_L \to mp$. Then a *direct transformation of AHL-processes* $mp \overset{(\varrho^*,\varrho),(m^*,m)}{\Longrightarrow} mp'$ is given by the commuting cube in Figure 4.28b where the front and back faces are direct transformations of AHL-nets and AHL-process nets, respectively. △



(a) Transformation of AHL-process

(b) Commuting Cube

Figure 4.28: Transformation of AHL-process

In the following we show how to construct productions for processes from a special type of production for AHL-nets, called action evolution. An action evolution is a direct transformation of AHL-nets that uses a special kind of production which in addition to the production for AHL-nets consists of a pattern describing the changes that should be made in corresponding processes in order to keep them consistent with the changes made in the system.

Definition 4.8.8 (Action Evolution Pattern for AHL-Processes)**.** A production $\varrho^* : L^* \overset{l^*}{\leftarrow} I^* \overset{r^*}{\rightarrow} R^*$ for AHL-process nets is called a *single action evolution pattern for AHL-processes* if it satisfies the following:

1. *(Single Action)* L^* contains only one transition and its environment, i. e. $T_{L^*} = \{t_{\varrho^*}\}$ and for all $p \in P_{L^*}$: $p \in {}^\bullet t_{\varrho^*} \cup t_{\varrho^*}{}^\bullet$,
2. *(Preserved Environment)* ϱ^* is non-deleting on places, i. e. $P_{L^*} = l^*_P(P_{I^*})$, and
3. *(Preserved Input and Output)* ϱ preserves input and output places, i. e. for all $p \in P_{I^*}$: $l^*(p) \in IN(L^*) \Rightarrow r^*(p) \in IN(R^*)$ and $l^*(p) \in OUT(L^*) \Rightarrow r^*(p) \in OUT(R^*)$.

Moreover, a production ϱ for AHL-process nets is called a *(multi) action evolution pattern for AHL-processes*, if it is a parallel production (i. e. the componentwise coproduct) $\varrho^* = \coprod_{i \in \mathcal{I}} \varrho^*_i$ of single action evolution patterns for AHL-processes $(\varrho^*_i)_{i \in \mathcal{I}}$, where $\mathcal{I}$ is a finite index set. △

Now, an action evolution is a special kind of transformation of AHL-nets, where the production that is used for the transformation is required to be adequately equipped with an action evolution pattern. Adequate means that for each transition (action) that is in the left-hand side of the system part of the production, the action evolution pattern consists of exactly one corresponding single action evolution pattern that describes the modification of occurrences of the action in question. Moreover, it is required that a pattern removes occurrences of actions if and only if the corresponding action is removed in the system part by the production. Finally, for the match that is used, we require that it is injective on transitions.

Definition 4.8.9 (Action Evolution)**.** A production (ϱ^*, ϱ) for AHL-processes as shown in Figure 4.29 is called a *production for action evolution* if

1. *(Action Evolution Pattern)* the process part ϱ^* is an action evolution pattern for AHL-processes,
2. *(Complete and Consistent Patterns)* the transition-component $mp_{L,T}$ of the left-hand side is a bijection, and
3. *(Consistent Removal)* diagram (1) is a pullback.

Given a production for action evolution (ϱ^*, ϱ), an AHL-net AN and a transition-injective match $m : L \to AN$ (i. e. m_T is injective), then a direct transformation $AN \overset{\varrho,m}{\Longrightarrow} AN'$ is called *action evolution with pattern* ϱ^*.

$$\begin{array}{ccccc} L^* & \xleftarrow{l^*} & I^* & \xrightarrow{r^*} & R^* \\ {\scriptstyle mp_L}\downarrow & (1) & \downarrow{\scriptstyle mp_I} & (2) & \downarrow{\scriptstyle mp_R} \\ L & \xleftarrow[l]{} & I & \xrightarrow[r]{} & R \end{array}$$

Figure 4.29: Production for action evolution

△

Example 4.8.10 (Action Evolution)**.** A production for action evolution (ϱ^*, ϱ) is depicted in Figure 4.30. The system part ϱ of the production is exactly the production for platform evolution presented in Example 3.2.3, describing the replacement of two actions *insert* and *mathitremove* by a new action *modify copy* that allows the insertion and removal of text while working on a copy of a wavelet.

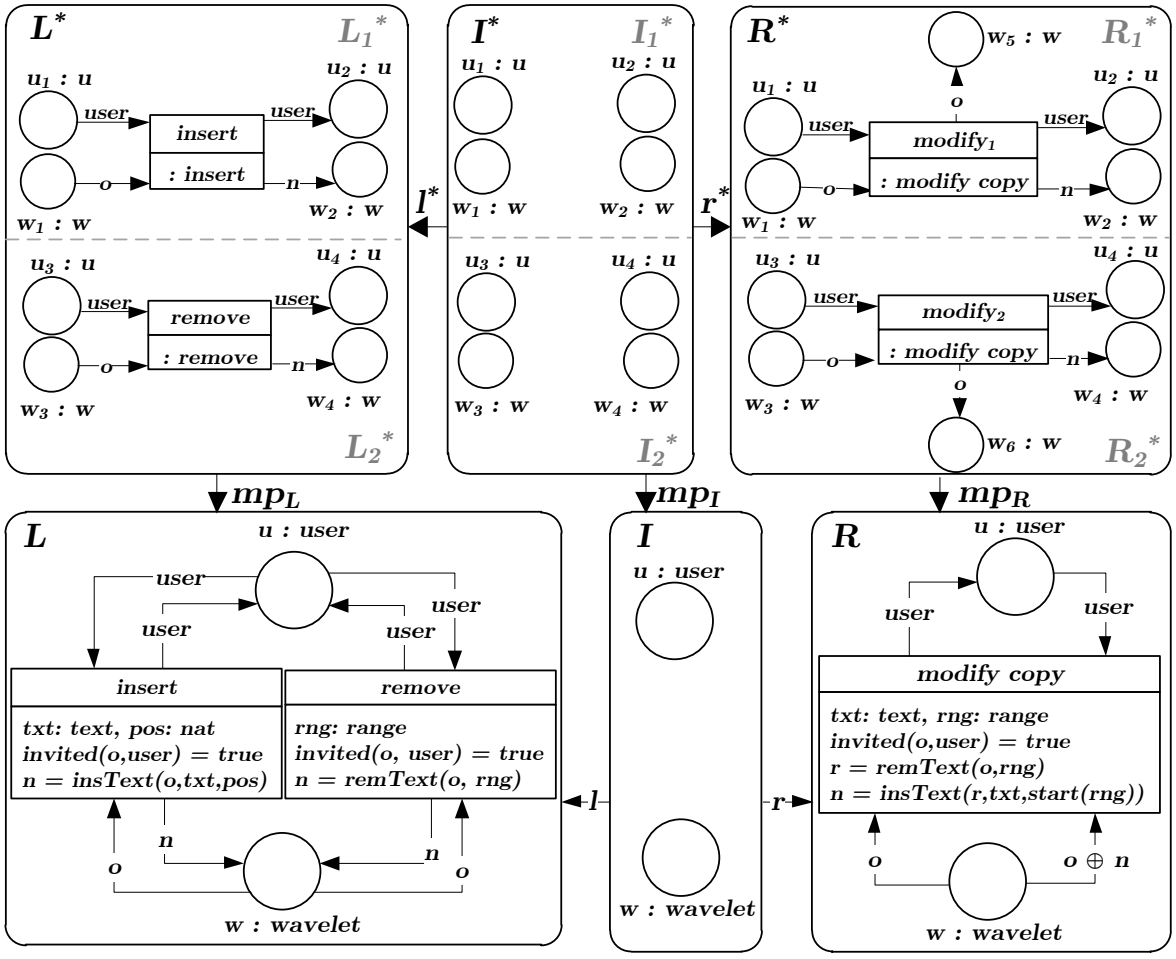


Figure 4.30: Production for action evolution (ϱ^*, ϱ)

The process part of the production is an action evolution pattern containing two single action evolution patterns – one for the action *insert* and one for the action *remove*. Both of these patterns specify the removal of the original action occurrence and the introduction of

a new occurrence of the new *modify copy* action, connected to the preserved environment of the removed occurrence.

Note that the illustration of the production ϱ^* is slightly inaccurate in the following sense: The parallel production ϱ^* is constructed as componentwise coproduct of AHL-process nets which according to Fact A.3.6 is constructed as disjoint union of AHL-nets. According to Definition A.3.1, the disjoint union of AHL-nets has as data type part the coproduct of the data type parts of all its elements. Assuming that both of the single action evolution patterns ϱ_1^* and ϱ_2^* have the signature Σ-*Wave* as signature part, the parallel production has Σ-*Wave* + Σ-*Wave* as signature part. Therefore, for instance, instead of the variable *user* occurring in the environment of the transitions, there are actually copies $(user, 1)$ in the environment of *insert* and $modify_1$, and $(user, 2)$ in the environment of *remove* and $modify_2$. However, in the illustration we omit these indices for a better readability.

Considering the platform evolution presented in Example 3.2.17, we have an action evolution $Platform \overset{\varrho,m}{\Longrightarrow} Platform'$ with pattern ϱ^*. Note that it is possible to specify different patterns. For instance, it is possible to specify that one or both of the actions is not replaced but rather removed. Further note that the single action requirement in Definition 4.8.8 is only required for the left-hand side of each of the single action evolution patterns, but not for the right-hand side. Therefore, an occurrence of a single action can be replaced by a complex process consisting of a multitude of actions. If we want to use actions that are already present in the original AHL-net, it is necessary to include these actions in the left-hand side, interface and right-hand side of the system part of the production. ♢

Remark 4.8.11 (Single Action Evolution). Note that each single action evolution pattern ϱ can also be considered as multi action evolution pattern, because it can be considered as coproduct $\varrho = \coprod_{i\in\mathcal{I}} \varrho_i$ with $\mathcal{I} = \{0\}$ and $\varrho_0 = \varrho$.

In [GE12a] we defined a construction for the transformation of scenarios based on single action evolutions of platforms. An important restriction of that construction was the requirement that the evolution of the platform can only change one single action. Since we aim to replace two actions *insert* and *remove* in our example above, it is not possible to use that construction. Moreover, the scenario evolution based on single action evolution only allows to insert occurrences of actions in the scenario that have been newly introduced to the platform. Therefore, even an iterated application of that construction cannot be used for our example, since both of the actions should be replaced by the same action *modify copy* which would already exist after the first iteration. △

In order to determine how the action evolution pattern of a production for action evolution has to be used for the evolution of a corresponding AHL-process net, we have to make a choice of matches as defined in the following. Note, however, that in a "proper" high-level net with a sufficiently complex data type part, like our example in Figure 3.2, it is possible to determine the "role" of each place by using distinctive term inscriptions for different places in the pre respectively post domain of a transition. In that case there is only one possible choice of matches for the elements of an action evolution to their occurrences in a corresponding AHL-process net, and therefore, there is only one possible interpretation for the handling of these occurrences, provided that the match has at least an injective signature part.

Definition 4.8.12 (Action Occurrences and Choices of Matches)**.** Given an action evolution $AN \overset{\varrho,m}{\Longrightarrow} AN'$ with pattern $\varrho^* = \coprod_{i\in\mathcal{I}} \varrho_i^*$, and a process $mp : K \to AN$. The family $occ = (occ_i)_{i\in\mathcal{I}}$ of all *action occurrences* is defined by

$$occ_i = \{t \in T_K \mid mp(t) = m \circ mp_L \circ \iota_i^L(t_{\varrho_i^*})\}$$

for all $i \in \mathcal{I}$, where ι_i^L is the coproduct injection $\iota_i^L : L_i^* \to L^*$ and $t_{\varrho_i^*}$ is the single transition in $T_{L_i^*}$ (see item 1 in Definition 4.8.8).

An AHL-morphism $m_{i,o} : L_i^* \to K$ is called *match for occurrence* o if $m_{i,o}(t_{\varrho_i^*}) = o$ and $m_{i,o}$ is compatible with m and mp, i. e. $mp \circ m_{i,o} = m \circ mp_L \circ \iota_i^L$.

A *choice of matches for occurrences* is a family $(m_{i,o} : L_i^* \to K)_{i \in \mathcal{I}, o \in occ_i}$ such that for every $i \in \mathcal{I}$ and $o \in occ_i$ we have that $m_{i,o}$ is a match for occurrence o.

$$\begin{array}{ccccc} L_i^* & \xrightarrow{\iota_i^L} & L^* & \xrightarrow{mp_L} & L \\ {\scriptstyle m_{i,o}}\downarrow & & & & \downarrow{\scriptstyle m} \\ K & & \xrightarrow[mp]{} & & AN \end{array}$$

△

Remark 4.8.13 (Set of Action Occurrences). Due to the fact that each single action evolution pattern contains exactly one single transition, the set occ_i of action occurrences in Definition 4.8.12 is isomorphic to the set

$$\{(t, t_{\varrho_i^*}) \in T_K \times T_{L_i^*} \mid mp(t) = m \circ mp_L \circ \iota_i^L(t_{\varrho_i})\}$$

This means that according to the standard construction of pullbacks in **Sets** and due to uniqueness of pullbacks up to isomorphism, the set of occurrences can be constructed as pullback over mp_T and $m_T \circ mp_{L,T} \circ \iota_{i,T}^L$ in **Sets**. △

Theorem 4.8.14 (Process Evolution based on Action Evolution)**.** *Given an action evolution* $AN \overset{\varrho,m}{\Longrightarrow} AN'$ *with pattern* $\varrho^* = \coprod_{i \in \mathcal{I}} \varrho_i^*$*, and a process* $mp : K \to AN$*.*

Then for every choice of matches for occurrences $(m_{i,o} : L_i^* \to K)_{i \in \mathcal{I}, o \in occ_i}$ *(see Definition 4.8.12) there exists a production* (ϱ^+, ϱ) *for AHL-processes and a direct transformation* $mp \overset{(\varrho^+,\varrho)}{\Longrightarrow} mp'$ *as depicted in Figure 4.31. In this case,* $mp \overset{(\varrho^+,\varrho)}{\Longrightarrow} mp'$ *is called process evolution of* mp *based on action evolution* $AN \overset{\varrho,m}{\Longrightarrow} AN'$ *with pattern* ϱ^**.*

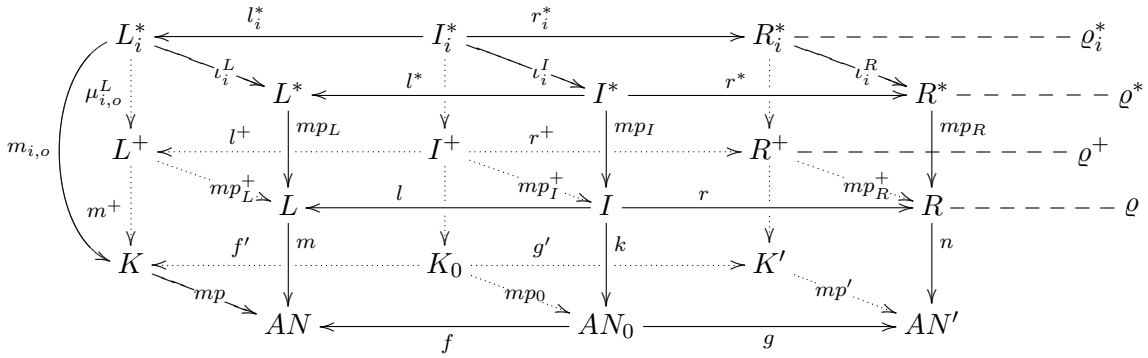


Figure 4.31: Process evolution based on action evolution

Construction and Proof-Idea.

1. The action evolution pattern ϱ^+ is constructed as parallel production (componentwise coproduct) $\varrho^+ = L^+ \overset{l^+}{\leftarrow} I^+ \overset{r^+}{\to} R^+ = \coprod_{i \in \mathcal{I}} \coprod_{o \in occ_i} \varrho_i^*$ with coproduct injections $\mu_{i,o}^X : X_i^* \to X^+$ for $X \in \{L, I, R\}$, which together with morphisms $mp_X \circ \iota_i^X$ induce unique morphisms $mp_X^+ : X^+ \to X$ such that $(\varrho^+, \varrho) = mp_L^+ \overset{(l^+,l)}{\leftarrow} mp_I^+ \overset{(r^+,r)}{\to} mp_R^+$ is a production for AHL-processes (see Definition 4.8.7).

2. A match m^+ is induced by coproduct $L^+ = \coprod_{i\in\mathcal{I}} \coprod_{o\in occ_i} L_i^*$ and matches $m_{i,o}$.

3. Then, using Lemma A.7.3, we obtain a direct transformation $K \overset{\varrho^+,m^+}{\Longrightarrow} K'$ of AHL-process nets in the lower back of Figure 4.31.

4. The process $mp_0 : K_0 \to AN_0$ can be obtained by construction of K_0 as pullback in the left bottom of Figure 4.31, and the process $mp' : K' \to AN'$ is induced by universal property of the pushout in the lower right back of the cube.

For the detailed proof see Section B.11 on page 254. □

Remark 4.8.15 (Process Evolution based on Action Evolution). The application of production ϱ^+ with match m^+ realises exactly the changes described by the single action evolution patterns ϱ_i^* on every corresponding occurrence in the AHL-process. △

Concept 4.8.16 (Abstract Scenario Evolution based on Platform Evolution)**.**
In Concept 4.8.5 we already considered existing abstract scenarios of a platform that is being modified. In the case that the extension condition is satisfied by the AHL-process modelling the given scenario, the scenario can be extended to the new modified platform. An extension of a scenario means that the unchanged scenario can also be considered as a scenario of the new platform as it is. Since it is possible that the extension condition is not satisfied, we need a way to adapt the scenario in order to make it again consistent with the changes that have been made to the corresponding platform. This can be done using the process evolution based on action evolution in Theorem 4.8.14, if the evolution of the platform only involves changes of actions. For this purpose, it is necessary that the production that is used for evolution of the platform is equipped with additional patterns that abstractly describe corresponding changes to the occurrences of each action. Given a choice of matches for all action occurrences, these patterns are then applied to the scenario, modifying the scenario according to the modification of the platform, and leading to a new scenario consistent with the new platform. An example of the evolution of abstract scenarios based on platform evolution is shown in Example 4.8.17. △

Example 4.8.17 (Abstract Scenario Evolution based on Platform Evolution)**.** As mentioned in Example 4.8.10, the direct transformation $Platform \overset{\varrho,m}{\Longrightarrow} Platform'$ in Example 3.2.17 can be considered as action evolution with pattern ϱ^*, where the action evolution pattern (ϱ^*, ϱ) is depicted in Figure 4.30 on page 105.

Now, consider the AHL-process $wave : Wave \to Platform$ in Figure 4.1 on page 60, modelling a scenario of the original platform of the action evolution. For the set of action occurrences regarding the action evolution and scenario, we have two occurrences of the *insert* action and one occurrence of the *remove* action. This means that we have $occ_1 = \{insert_1, insert_2\}$ and $occ_2 = \{remove_1\}$ as the first pattern contained in ϱ^* is a pattern for the *insert* action, and the second pattern describes changes to occurrences of the *remove* action.

Since the single components ϱ_i^* of ϱ^* and the AHL-process *Wave* share the same signature part Σ-*Wave*, for each index $i \in 1, 2$ and each occurrence $o \in occ_i$ there is exactly one choice of matches $m_{i,o} : L_i^* \to Wave$ into the scenario, because the matching of each transition is completely determined by the different arc inscriptions.

Following the construction in Theorem 4.8.14, we obtain a production ϱ^+ as depicted in Figure 4.32. For each index of a single action evolution pattern and each occurrence of that pattern, the production ϱ^+ consists of a copy of the corresponding pattern. Hence, ϱ^+

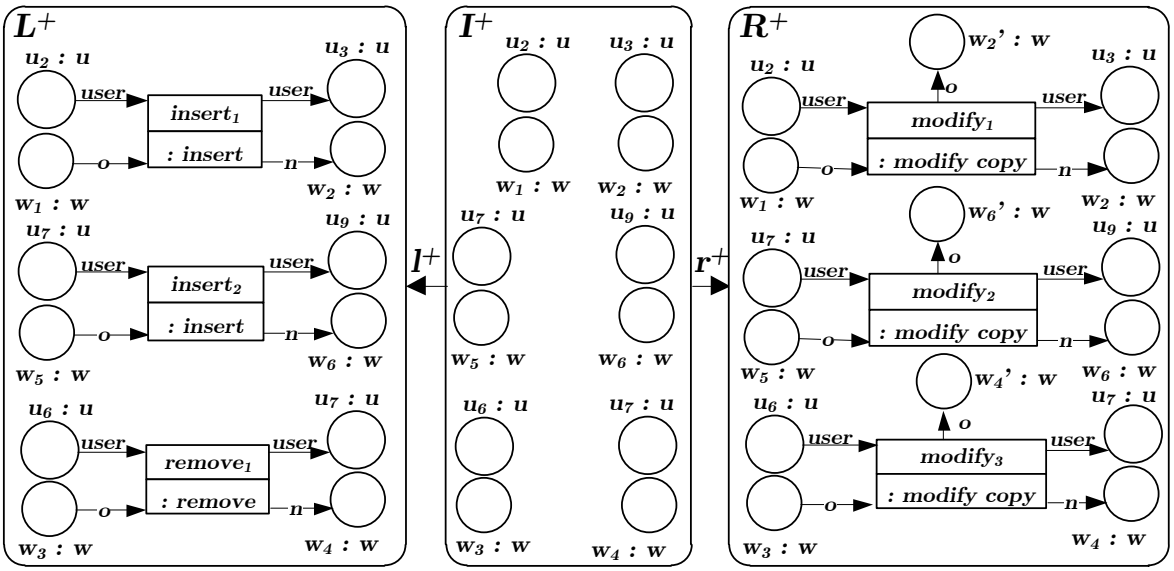


Figure 4.32: Production ϱ^+ for abstract scenario evolution based on action evolution

contains two copies of the *insert* pattern and one copy of the *remove* pattern. For illustration of the induced match m^+ we renamed all elements in ϱ^+ such that m^+ becomes an inclusion.

From Theorem 4.8.14 we know that ϱ^+ is applicable with match m^+, leading to a direct transformation of AHL-processes $wave \overset{(\varrho^+,\varrho),(m^+,m)}{\Longrightarrow} wave$. The result $wave'$ of the transformation is shown in Figure 4.33.

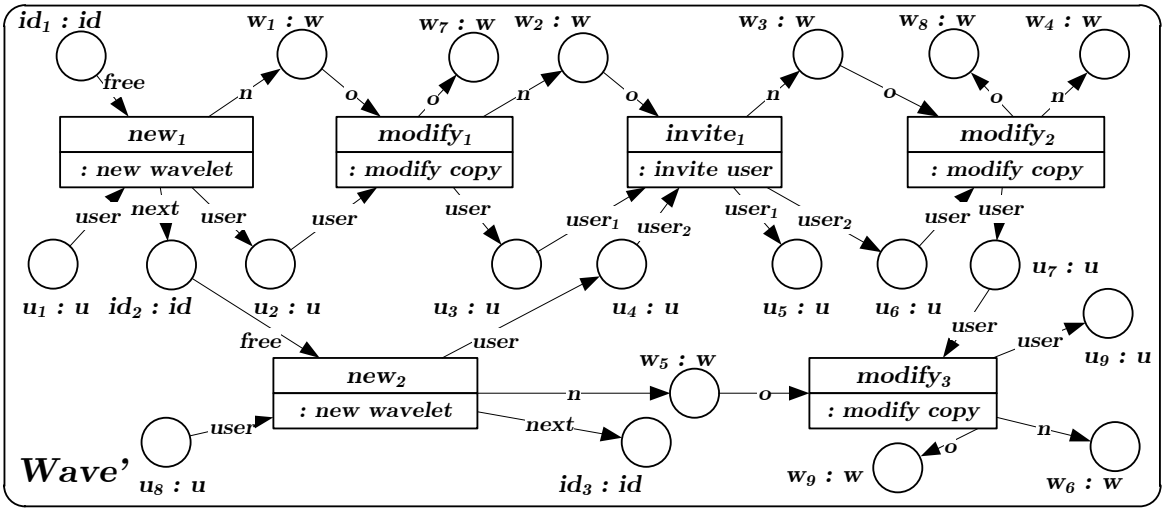


Figure 4.33: Scenario $wave'$

In the scenario $wave'$ the two occurrences of action *insert* and the occurrence of action *remove* have been replaced by new occurrences of the newly created action *modify copy*. The occurrences are connected to the places that formed the environments of the old action occurrences, as this was specified by the action evolution pattern (ϱ^*, ϱ). Hence, the replacement of the two actions *insert* and *remove* by the action *modify copy* on the platform layer has been successfully transferred to the abstract scenario layer. ♢

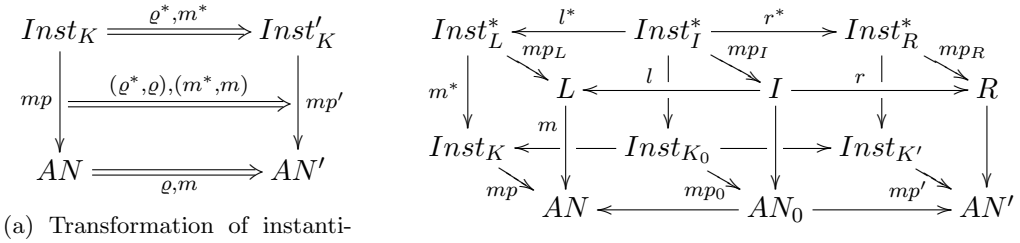
4.8.3 Concrete Scenario Evolution Based on Platform Evolution

In the previous Subsection 4.8.2, we presented an approach to transfer changes to actions of a platform also to existing abstract scenarios of that platform, leading to a new consistent scenario of the resulting platform, where all occurrences of modified actions are modified as well. In this subsection we extend our approach to allow the same also for concrete scenarios.

Remark 4.8.18. Note that in this subsection we make extensive use of the short notation for instantiated AHL-morphisms introduced in Remark 4.3.4, i. e. we often write $mp : Inst_K \to AN$ for an instantiated AHL-process $(inst, mp)$, where $Inst_K = (inst, K)$ is an instantiated AHL-process net and $mp : K \to AN$ is an AHL-process. △

Definition 4.8.19 (Transformation of Instantiated AHL-Processes)**.** A *production for instantiated AHL-processes* is a span $(\varrho^*, \varrho) : mp_L \overset{(l^*,l)}{\longleftarrow} mp_I \overset{(r^*,r)}{\longrightarrow} mp_R$ of injective **ProcInst**-morphisms with isomorphic data type parts as shown in the top of Figure 4.28b. We call $\varrho^* : Inst^*_L \overset{l^*}{\leftarrow} Inst^*_I \overset{r^*}{\to} Inst^*_R$ the *process part* and $\varrho : L \overset{l}{\leftarrow} I \overset{r}{\to} R$ the *system part* of (ϱ^*, ϱ).

Given a production $\varrho : mp_L \overset{(l^*,l)}{\longleftarrow} mp_I \overset{(r^*,r)}{\longrightarrow} mp_R$ for instantiated AHL-processes and a (match) morphism $(m^*, m) : mp_L \to mp$. Then a *direct transformation of instantiated AHL-processes* $mp \overset{(\varrho^*,\varrho),(m^*,m)}{\Longrightarrow} mp'$ is given by the commuting double-cube in Figure 4.34b where the front and back faces are direct transformations of AHL-nets and instantiated AHL-process nets, respectively. △



(a) Transformation of instantiated AHL-process

(b) Commuting Cube

Figure 4.34: Transformation of instantiated AHL-process

Definition 4.8.20 (Action Evolution Pattern for Instantiated AHL-Processes)**.** An abstract production $\varrho^* : Inst_{L^*} \overset{l^*}{\leftarrow} Inst_{I^*} \overset{r^*}{\to} Inst_{R^*}$ for instantiated AHL-process nets is called a *single action evolution pattern for instantiated AHL-processes* if the underlying production $PNet(\varrho^*) : L^* \overset{l^*}{\leftarrow} I^* \overset{r^*}{\to} R^*$ for AHL-process nets is a single action evolution pattern for AHL-processes (see Definition 4.8.8).

Moreover, a production ϱ^* for instantiated AHL-process nets is called a *(multi) action evolution pattern for instantiated AHL-processes*, if it is a parallel production (i. e. the componentwise coproduct) $\varrho^* = \coprod_{i\in\mathcal{I}} \varrho^*_i$ of single action evolution patterns for instantiated AHL-processes $(\varrho^*_i)_{i\in\mathcal{I}}$, where $\mathcal{I}$ is a finite index set. △

Corollary 4.8.21 (Action Evolution Pattern is Abstract Production)**.** *Every (multi) action evolution pattern for instantiated AHL-processes is an abstract production.*

Proof. Every single action evolution pattern for instantiated AHL-processes is by definition an abstract production. So, since a (multi) action evolution pattern is defined as parallel production, Lemma A.3.10 implies that also (multi) action evolution patterns are abstract. □

Definition 4.8.22 (Action Evolution with Instantiated Pattern)**.** A production (ϱ^*, ϱ) for instantiated AHL-processes as shown in Figure 4.35 is called a *production for action evolution with instantiated pattern* if the underlying production for AHL-processes $Proc(\varrho^*, \varrho)$ is a production for action evolution (see Definition 4.8.9).

Given a production for action evolution (ϱ^*, ϱ) with instantiated pattern, an AHL-net AN and a transition-injective match $m : L \to AN$ (i. e. m_T is injective), then a direct transformation $AN \overset{\varrho,m}{\Longrightarrow} AN'$ is called *action evolution with instantiated pattern* ϱ^*.

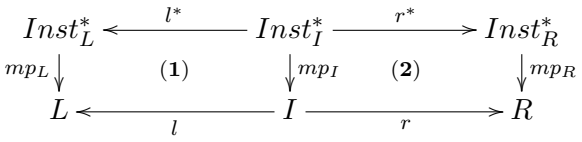


Figure 4.35: Production for action evolution

△

Example 4.8.23 (Action Evolution with Instantiated Pattern)**.** A production (ϱ^*, ϱ) for action evolution with instantiated pattern is shown in Figure 4.36. Note that the underlying production for AHL-processes is exactly the production for action evolution in Example 4.8.10. In this example, the action evolution pattern is instantiated with abstract values and transition assignments with respect to a term algebra. As already discussed in Example 4.8.10, the parallel production ϱ^* is constructed as componentwise coproduct of ϱ_1^* and ϱ_2^* which means that we also have coproducts of the data type part. This does not only apply to the signature, but also to the algebra – which in the case of abstract productions ϱ_1^* and ϱ_2^* are term algebras. In this example we choose the variable families $X_{\varrho_1^*}$ and $X_{\varrho_2^*}$ in the way that for each sort in the signature the corresponding sets are disjoint. Therefore, we do not need to annotate the elements with indices, because the coproduct (disjoint union) of the variable sets are isomorphic to the respective union.

Further note that all components of the production ϱ for the platform transformation is equipped with our Σ-*Wave* in Table 3.1 on page 36 and algebra A in Table 3.2 on page 37. So despite the fact that the production ϱ has an abstract data type part (i. e. term algebras), the interpretation of the abstract data values is clear, due to the generalised homomorphism $(mp_{L,\Sigma}, mp_{L,A})$ that is the data type part of AHL-morphism mp_L, and also the data type parts of mp_I, and mp_R.

In the instantiated pattern, we specify that the new occurrences of action *modify copy* do exactly the same as the original occurrences of *insert* and *remove*, respectively, that are replaced by it. The single action evolution pattern ϱ_1^* describes that an insertion of text t at position p by user a in a wavelet v is replaced by a modification by the same user in the same wavelet, where we replace the range $(p, zero)$ with text t. Considering the equations in transition *modify copy*, this means that we remove an empty range of length zero, and insert the text t at position p. So the only difference to the replaced action occurrence is that we also produce a copy of the original wavelet.

The single action evolution pattern ϱ_2^* describes that the deletion of text at range rng by user b in a wavelet x is replaced by a modification by the same user in the same wavelet, where we replace the range rng with an empty text (using the empty string ϵ). So also in the case of the replacement of *remove* with $modify_2$, we keep almost the same semantics with the only difference that we produce a copy of the original wavelet.

Considering this production (ϱ^*, ϱ) for action evolution with instantiated pattern and the platform evolution in Example 3.2.17 using the system part ϱ of our production, the platform

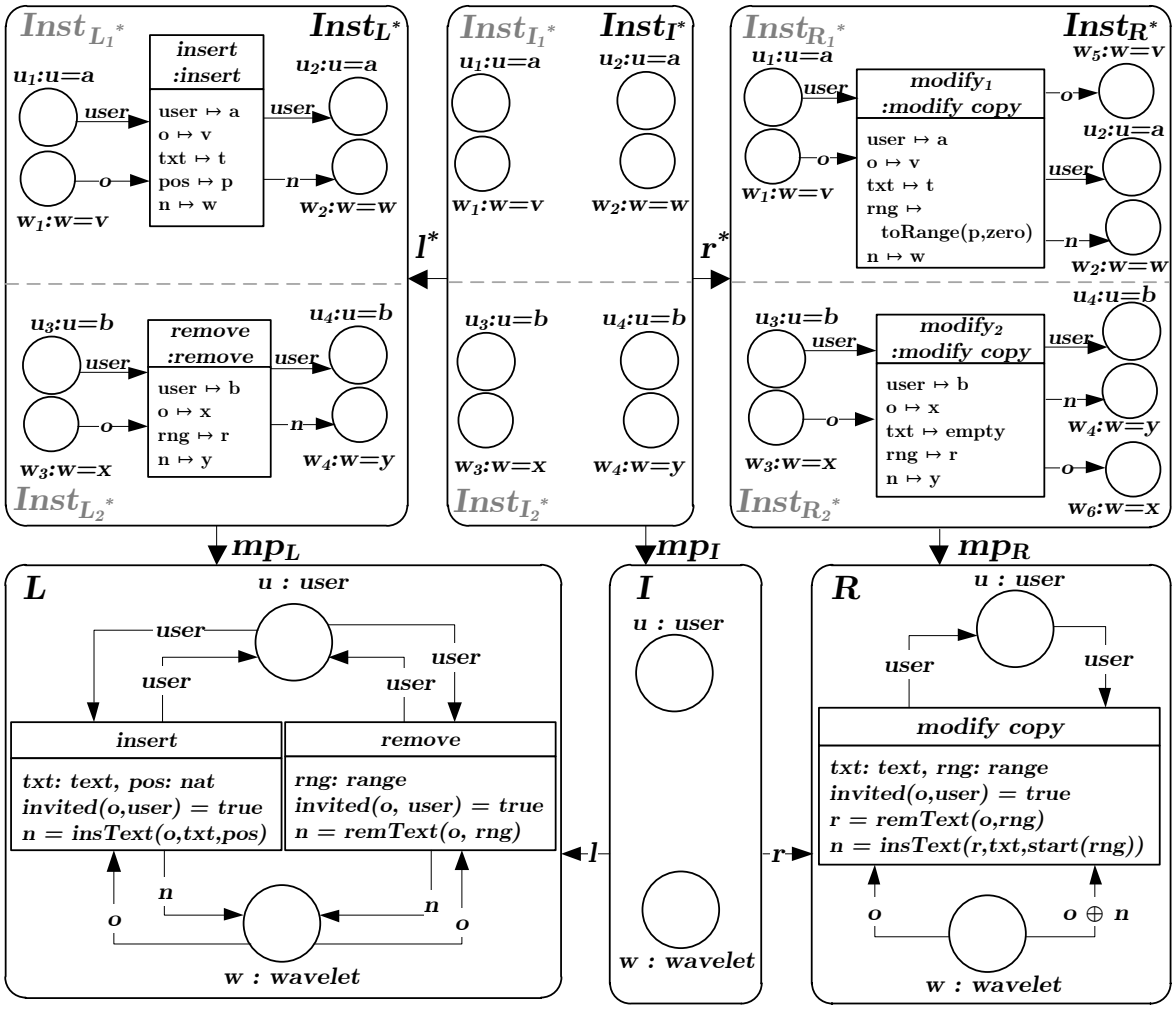


Figure 4.36: Production (ϱ^*, ϱ) for action evolution with instantiated pattern

evolution $Platform \overset{\varrho,m}{\Longrightarrow} Platform'$ can be seen as an action evolution with instantiated pattern ϱ^*. An example of the application of the pattern is given in Example 4.8.27. ♢

Definition 4.8.24 (Choice of Matches for Instantiated Pattern)**.** Given an action evolution $AN \overset{\varrho,m}{\Longrightarrow} AN'$ with instantiated pattern $\varrho^* = \coprod_{i\in\mathcal{I}} \varrho_i^*$, and an instantiated process $mp : Inst_K \to AN$.

A weak instantiation morphism $m_{i,o} : Inst_{L_i^*} \to Inst_K$ is called *match for occurrence* $o \in occ_i$ (see Definition 4.8.12) if $m_{i,o}(t_{\varrho_i}) = o$ and $m_{i,o}$ is compatible with m and mp, i. e. $mp \circ m_{i,o} = m \circ mp_L \circ \iota_i^L$.

A *choice of matches for occurrences* is a family $(m_{i,o} : Inst_{L_i^*} \to Inst_K)_{i\in\mathcal{I},o\in occ_i}$ such that for every $i \in \mathcal{I}$ and $o \in occ_i$ we have that $m_{i,o}$ is a match for occurrence o. △

Theorem 4.8.25 (Evolution of Instantiated Processes based on Action Evolution)**.** *Given an action evolution* $AN \overset{\varrho,m}{\Longrightarrow} AN'$ *with instantiated pattern* $\varrho^* = \coprod_{i\in\mathcal{I}} \varrho_i^*$*, and an instantiated process* $mp : Inst_K \to AN$*.*

Then for every choice of matches for occurrences $(m_{i,o} : Inst_{L_i^*} \to Inst_K)_{i\in\mathcal{I},o\in occ_i}$ *such that* ϱ_i^* *and* $m_{i,o}$ *satisfy the instantiation condition for abstract productions (see Definition 4.6.7), there exists a production* (ϱ^+, ϱ) *and a direct transformation of instantiated*

AHL-processes $mp \overset{(\varrho^+,\varrho)}{\Longrightarrow} mp'$ *as depicted in Figure 4.31. In this case,* $mp \overset{(\varrho^+,\varrho)}{\Longrightarrow} mp'$ *is called process evolution of* mp *based on action evolution* $AN \overset{\varrho,m}{\Longrightarrow} AN'$ *with instantiated pattern* ϱ^*.

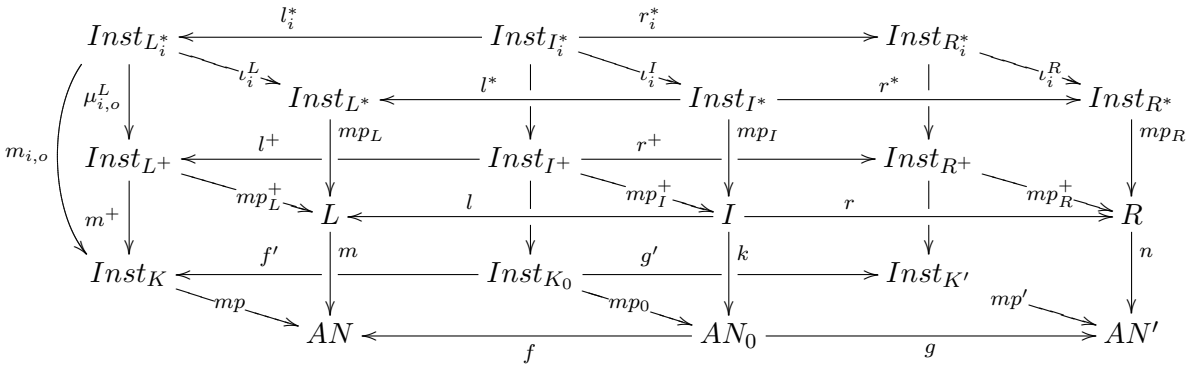


Figure 4.37: Evolution of instantiated processes based on action evolution

Construction and proof.

1. The production ϱ^+ is constructed as coproduct $\varrho^+ = \coprod_{i\in\mathcal{I}} \coprod_{o\in occ_i} \varrho_i^*$. Then, by Fact A.3.8 we have that $Net(\varrho^+) = \coprod_{i\in\mathcal{I}} \coprod_{o\in occ_i} Net(\varrho_i^*)$ which means that the AHL-net part of ϱ^+ is constructed similar to the corresponding construction in Theorem 4.8.14.

2. Accordingly, we obtain unique morphisms m^+, mp_L^+, mp_I^+ and mp_R^+, induced by coproducts $Inst_{L^+}$, $Inst_{I^+}$ and $Inst_{R^+}$, that coincide with the corresponding AHL-morphisms in the construction in Theorem 4.8.14.

3. The construction for all missing AHL-net parts in the cube is shown in Theorem 4.8.14.

4. The corresponding constructions for weak instantiations is obtained using Fact A.5.11, leading to a direct transformation of weakly instantiated AHL-processes $mp \overset{\varrho^+,m^+}{\Longrightarrow} mp'$.

5. Since all single action evolution patterns ϱ_i^* with matches $m_{i,o}$ ($i \in \mathcal{I}$, $o \in occ_i$) are required to satisfy the instantiation condition, by Fact A.3.12 we have that also the parallel production ϱ^+ with induced match m^+ satisfy the instantiation condition. Hence, using Theorem 4.6.8 we have that $Inst_K \overset{\varrho^+,m^+}{\Longrightarrow} Inst_{K'}$ is a direct transformation of concrete instantiations, and thus, $mp \overset{\varrho^+,m^+}{\Longrightarrow} mp'$ is a direct transformation of instantiated AHL-processes. □

Concept 4.8.26 (Concrete Scenario Evolution based on Platform Evolution)**.** In Concept 4.8.16 we already discussed the evolution of abstract scenarios based on the evolution of platforms. For the case that we have concrete scenarios of the original platform of a platform evolution, it does not suffice to transfer the evolution of the AHL-net model to the corresponding AHL-process model, but we also need a way to "translate" also the data values and assignments of the corresponding instantiation. This can be done using the evolution of instantiated processes based on action evolution in Theorem 4.8.25, if the evolution of the platform only involves changes of actions. For this purpose, it is required that the production that is used for the platform evolution is also equipped with additional information regarding the occurrences of actions in an instantiated AHL-process. Given a choice of matches for all action occurrences, these patterns are then applied to the concrete scenario, modifying the scenario according to the modification of the platform, and leading to a new scenario

consistent with the new platform. Note that in contrast to the evolution of abstract scenarios, the evolution of concrete scenarios based on platform evolution additionally requires that each pattern and every corresponding match in the given choice of matches satisfy the instantiation condition. An example of the evolution of concrete scenarios based on platform evolution is shown in Example 4.8.27. △

Example 4.8.27 (Concrete Scenario Evolution based on Platform Evolution)**.** As mentioned in Example 4.8.23, considering the production (ϱ^*, ϱ) for action evolution with instantiated pattern in Figure 4.36, and the platform evolution in Example 3.2.17 on page 45 using the system part ϱ of our production, the platform evolution $Platform \overset{\varrho,m}{\Longrightarrow} Platform'$ can be seen as an action evolution with instantiated pattern ϱ^*.

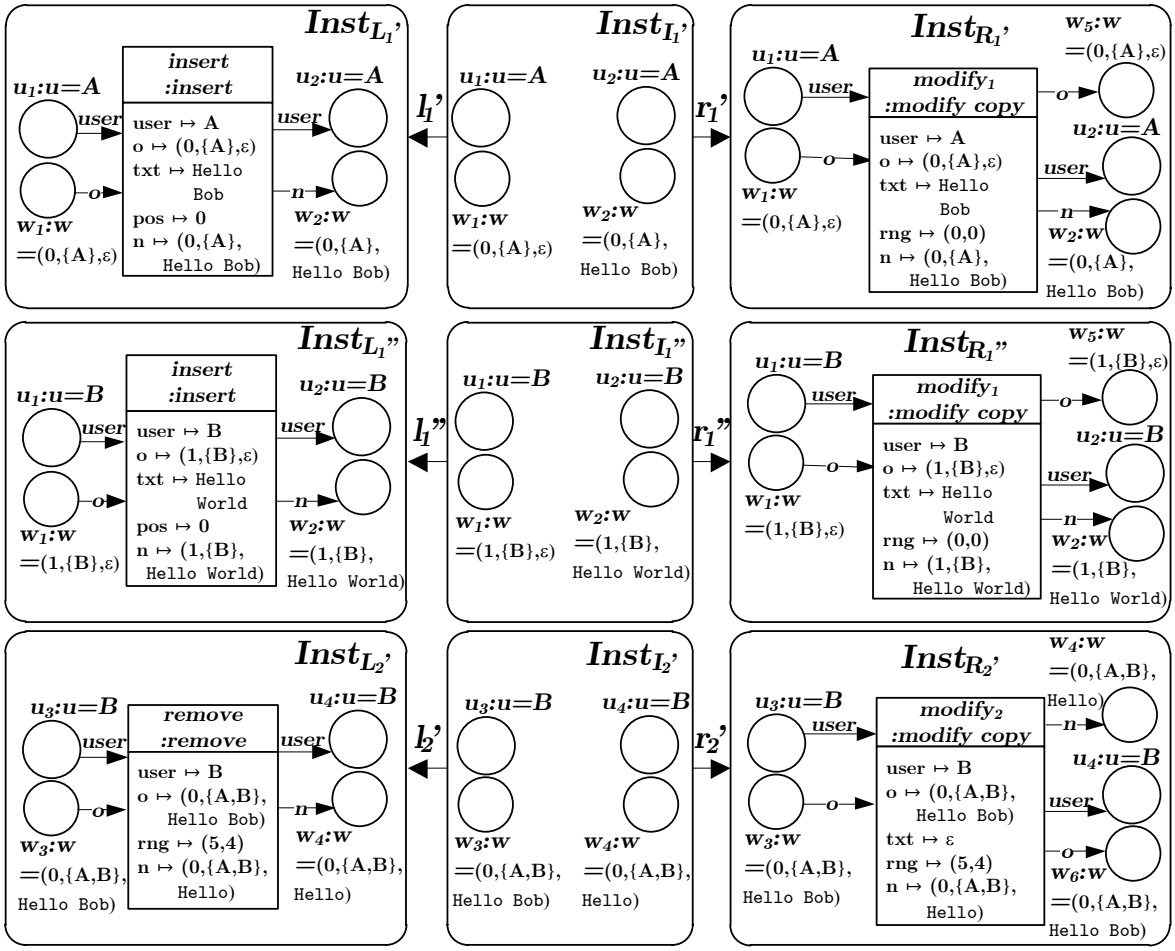


Figure 4.38: Data-shiftings of single action evolution patterns along choice of matches

Moreover, consider the instantiated AHL-process $wave : Inst \to Platform$ introduced in Example 4.3.7 on page 72. We have action occurrences $occ_1 = \{insert_1, insert_2\}$ of the first single action evolution pattern, and action occurrences $occ_2 = \{remove_1\}$ for the second single action evolution pattern contained in our instantiated multi action evolution pattern.

Since the single components ϱ_i^* of ϱ^* and the instantiated AHL-process net $Inst$ share the same signature part Σ-$Wave$, for each index $i \in 1, 2$ and each occurrence $o \in occ_i$ there is exactly one choice of matches $m_{i,o} : Inst_{L_i^*} \to Inst$ into the scenario, because the matching of each transition is completely determined by the different arc inscriptions.

In order to check whether we can apply our result in Theorem 4.8.25 to the given action

evolution, we have to check that each single action evolution pattern and corresponding match in the choice of matches satisfy the instantiation condition, i. e. the data-shiftings of the patterns along the matches have to be concrete productions. The data-shiftings are shown in Figure 4.38, where ϱ_1' at the top is the data-shifting of ϱ_1^* along $m_{1,insert_1}$, ϱ_1'' at the centre is the data-shifting of ϱ_1^* along $m_{1,insert_2}$, and ϱ_2' at the bottom is the data-shifting of ϱ_2^* along $m_{2,remove_1}$. Taking into account the firing conditions of the transitions which can be seen in Figure 4.36, we can see that the data-shiftings are concrete productions, since the firing conditions of the transitions are satisfied by the assignments given in the instantiations. In fact, as mentioned in Section 4.6, it is already clear that the left-hand sides and interfaces are concrete instantiations, because *Inst* is a concrete instantiation, and therefore it suffices to check that the assignments in the right-hand sides are consistent.

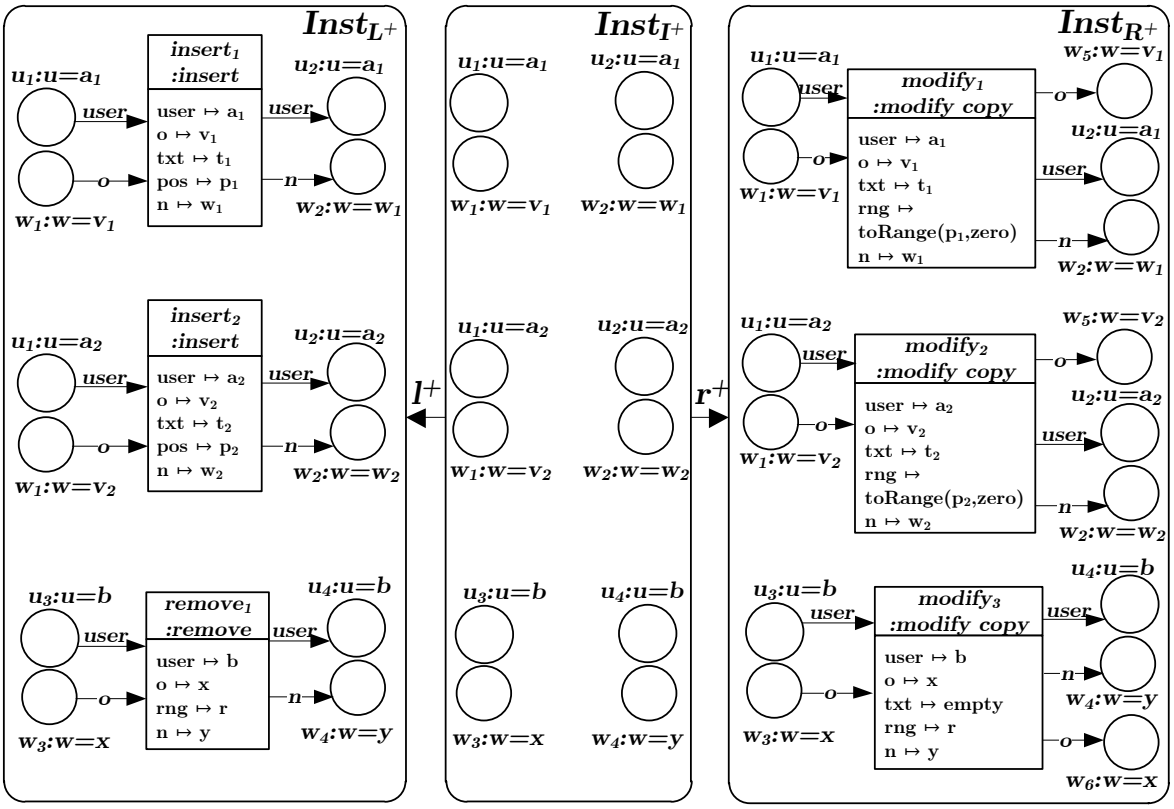


Figure 4.39: Production ϱ^+ for concrete scenario evolution based on action evolution

So, since the instantiation condition is satisfied by all single action evolution patterns and corresponding matches, we follow the construction in Theorem 4.8.25, leading to a parallel production ϱ^+ shown in Figure 4.39, consisting of copies of the single action evolution patterns for each corresponding action occurrence. Analogously to the previous examples, we do not explicitly denote the different copies of sorts, operation symbols or variables in the signature parts of the productions (e. g. in each component of ϱ^+ are three different copies of the variable *user* – one for each copy of a pattern), but we denote only the different copies of variables in the algebra part (e. g. in ϱ^+ are two copies a_1 and a_2 of the variable a from the algebra part of ϱ_1^*, as ϱ^+ contains two copies of that pattern).

Moreover, the construction of the parallel production ϱ^+ as componentwise coproduct induces a unique match morphism $m^+ : Inst_{L^+} \to Inst$ that is compatible with the single matches $m_{i,o}$ from our choice of matches. Further, note that the parallel production of the

three productions in Figure 4.38 is isomorphic to the data-shifting of ϱ^+ along m^+, and since that parallel production is a concrete production, we know that ϱ^+ can be applied to *Inst* at match m^+ with result *Inst*′ shown in Figure 4.40. We also obtain an AHL-process $wave' : Wave' \to Platform'$ that corresponds exactly to the AHL-process $wave'$ in Example 4.8.17, and we have that *Inst* and *wave* together form an instantiated AHL-process.

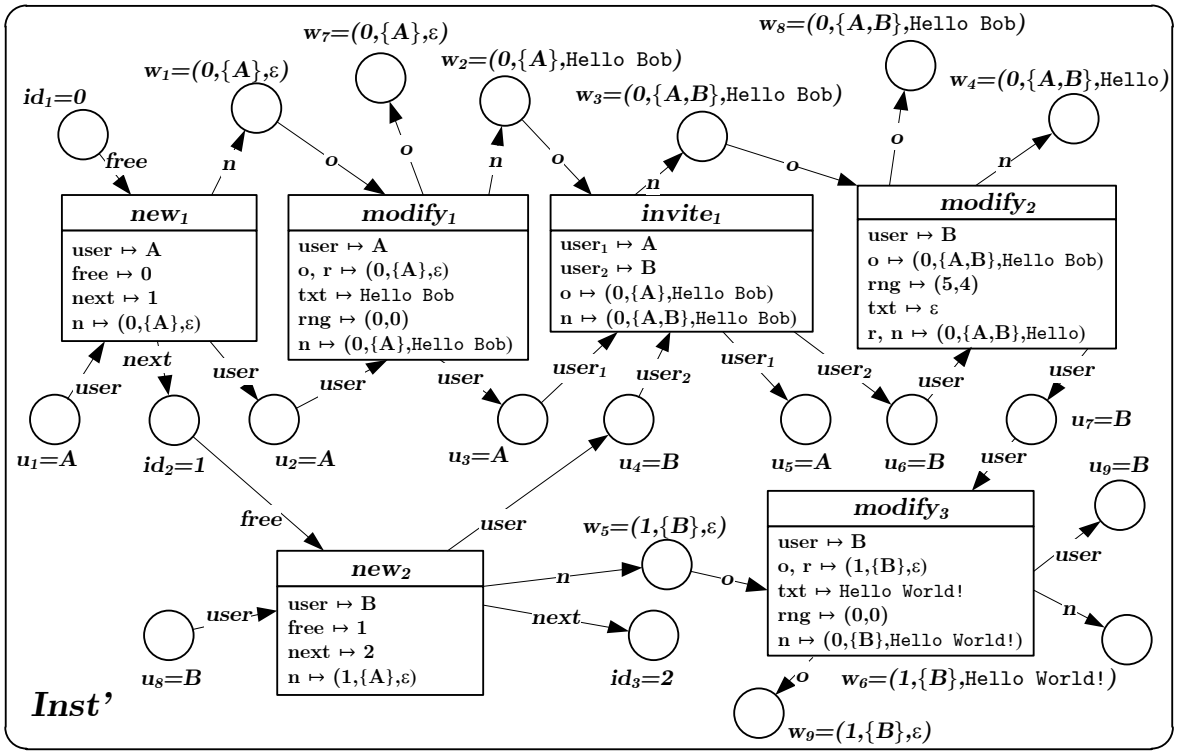


Figure 4.40: Instantiation *Inst*′

In the resulting scenario all occurrences of action *insert* and *remove* have been replaced with *modify copy* as specified by our production for action evolution with instantiated pattern. The new action occurrences have almost the same semantics as the previous action occurrences they replaced, with the only difference that they create a copy of the original wavelet. Hence, the replacement of the two actions *insert* and *remove* by the action *modify copy* on the platform layer has been successfully transferred to the concrete scenario layer. ♢

5 Analysis of Platforms and Scenarios

In this chapter, we present techniques for the analysis of communication platforms and scenarios. The Local Church-Rosser, Parallelism and Concurrency Theorems for AHL-nets in Section 5.1 are instantiations of corresponding general analysis results in the categorical framework of $\mathcal{M}$-adhesive categories (see Subsection A.1.3). These techniques can be used for the analysis of independent platform evolutions, and they provide means for the analysis and synthesis of parallel as well as concurrent evolutions of communication platforms.

Moreover, in Section 5.2, we introduce corresponding extensions of the Local Church-Rosser, Parallelism and Concurrency Theorems that adequately support the analysis of scenario evolutions.

Finally, in Section 5.3, we present the construction of a term equation system for the model of an abstract scenario, and we show that there is a 1-to-1 correspondence between concrete realisations of the abstract scenario and the solutions of the corresponding term equation system.

5.1 Independence of Platform Evolutions

Due to the fact that Apache Wave is open source it is possible that different developers perform their own evolutions outgoing from one platform. It is an interesting aspect to analyse whether such different evolutions are compatible with each other in the sense that each one of the evolutions can be also applied to the result of the respective other one leading to the same result as demonstrated in the following example.

Example 5.1.1 (Compatible Platform Evolutions)**.** In Example 3.2.17 we presented an evolution outgoing from AHL-net *Platform* in Figure 3.2 on page 35 using the production ϱ in Figure 3.3 on page 39. Another production ϱ_2 for AHL-nets is shown in Figure 5.1. The production describes that a transition *new wavelet* that is connected to *user*, *wavelet* and *id* places is deleted together with the connected *id* place.

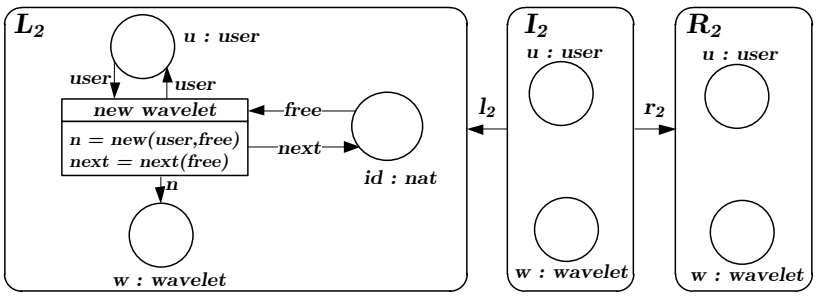


Figure 5.1: Production ϱ_2 for AHL-nets

The production ϱ_2 is also applicable to the AHL-net *Platform*, leading to another result

Platform$_2$ shown in Figure 5.2. In the figure, the production ϱ is called ϱ_1 and correspondingly the resulting AHL-net *Platform*$'$ from Example 3.2.17 is called *Platform*$_1$.

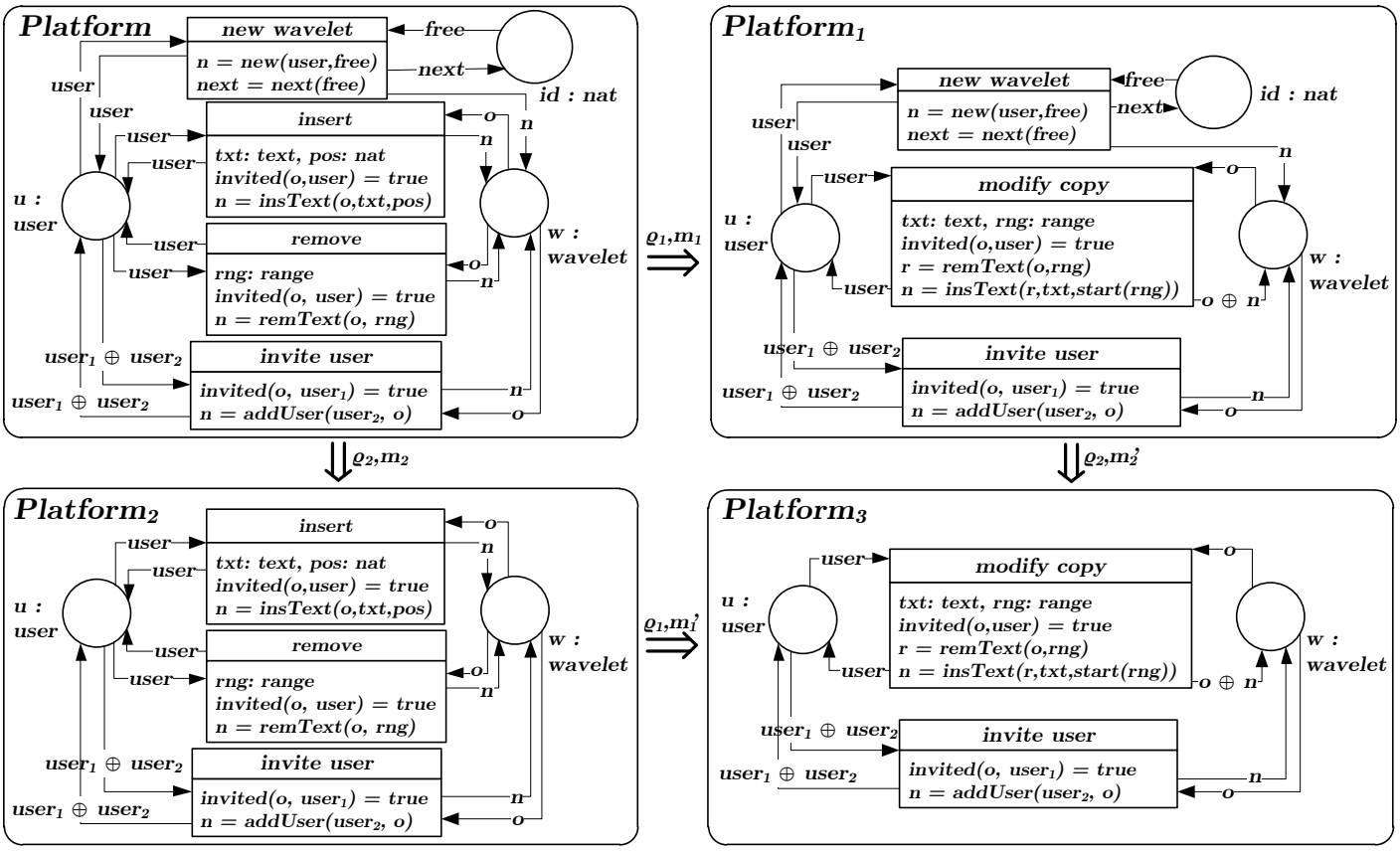


Figure 5.2: Compatible platform evolutions

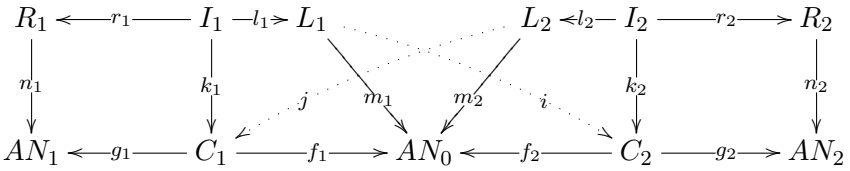
As illustrated in Figure 5.2, it is also possible to apply each of the productions also on the result of the transformation using the respective other productions: Removing the *new wavelet* action does not impede the replacement of the *insert* and *remove* actions with the *modify copy* action, and vice versa. Moreover, it is not only the case that the productions are still applicable after application of the respective other production, but we also obtain the same result, regardless of which production we apply first. So, the platform evolutions can be seen as compatible, or independent from each other. ♢

For the investigation of independent platform evolutions it proves to be useful that we model platforms using the well-researched modelling technique of algebraic high-level nets. Since we know that the category **AHLNets** together with the class $\mathcal{M}_{AHL}$ of injective AHL-morphisms with isomorphic data type part is $\mathcal{M}$-adhesive (see Fact A.1.11 in Subsection A.1.3), due to the results in [EGH10] we know that we can apply all analysis results from [EEPT06b] for weak adhesive HLR categories also for $\mathcal{M}$-adhesive categories, and thus, also for our category of AHL-nets.

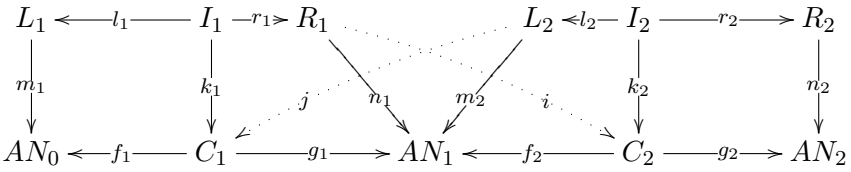
One of these results is the Local Church-Rosser Theorem (Theorem 5.12 in [EEPT06b]). In the following we show that direct transformations can be applied in any order (see Theorem 5.1.4), provided that they are parallel independent in the following sense as defined in [EEPT06b]:

Definition 5.1.2 (Parallel and Sequential Independence of AHL-Net Transformations)**.** Two direct transformations of AHL-nets $AN_0 \overset{\varrho_1,m_1}{\Longrightarrow} AN_1$ and $AN_0 \overset{\varrho_2,m_2}{\Longrightarrow} AN_2$ are called *parallel*

independent if there exist morphisms $i : L_1 \to C_2$ and $j : L_2 \to C_1$ such that $f_2 \circ i = m_1$ and $f_1 \circ j = m_2$.



Moreover, two direct transformations of AHL-nets $AN_0 \overset{\varrho_1,m_1}{\Longrightarrow} AN_1 \overset{\varrho_2,m_2}{\Longrightarrow} AN_2$ are called *sequentially independent* if there exist morphisms $i : R_1 \to C_2$ and $j : L_2 \to C_1$ such that $f_2 \circ i = n_1$ and $g_1 \circ j = m_2$.



△

Remark 5.1.3.

1. **(Characterization of Parallel Independence)** Parallel independence is equivalent to the fact that the matches only overlap in gluing points, i. e. $m_1(L_1) \cap m_2(L_2) \subseteq l_1(m_1(I_1)) \cap l_2(m_2(I_2))$.

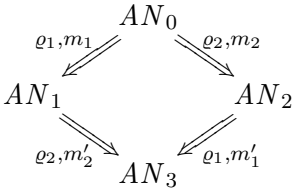
2. **(Relationship between Parallel and Sequential Independence)** Note that $AN_0 \overset{p_1}{\Rightarrow} AN_1 \overset{p_2}{\Rightarrow} AN_2$ are sequentially independent if and only if $AN_0 \overset{p_1^{-1}}{\Leftarrow} AN_1 \overset{p_2}{\Rightarrow} AN_2$ are parallel independent.

△

In the following we review the Local Church-Rosser Theorem for weak adhesive HLR categories from [EEPT06b], instantiated for our special case of AHL-nets.

Theorem 5.1.4 (Local Church-Rosser Theorem for AHL-Net Transformations)**.** *Given two parallel independent direct transformations $AN_0 \overset{\varrho_1,m_1}{\Longrightarrow} AN_1$ and $AN_0 \overset{\varrho_2,m_2}{\Longrightarrow} AN_2$ of AHL-nets, then there is an AHL-net AN_3 together with direct transformations $AN_1 \overset{\varrho_2,m_2'}{\Longrightarrow} AN_3$ and $AN_2 \overset{\varrho_1,m_1'}{\Longrightarrow} AN_3$ such that $AN_0 \overset{\varrho_1,m_1}{\Longrightarrow} AN_1 \overset{\varrho_2,m_2'}{\Longrightarrow} AN_3$ and $AN_0 \overset{\varrho_2,m_2}{\Longrightarrow} AN_2 \overset{\varrho_1,m_1'}{\Longrightarrow} AN_3$ are sequentially independent.*

Given two sequentially independent direct transformations $AN_0 \overset{\varrho_1,m_1}{\Longrightarrow} AN_1 \overset{\varrho_2,m_2'}{\Longrightarrow} AN_3$ of AHL-nets, there are an AHL-net AN_2 and direct transformations $AN_0 \overset{\varrho_2,m_2}{\Longrightarrow} AN_2 \overset{\varrho_1,m_1'}{\Longrightarrow} AN_3$ such that $AN_0 \overset{\varrho_1,m_1}{\Longrightarrow} AN_1$ and $AN_0 \overset{\varrho_2,m_2}{\Longrightarrow} AN_2$ are parallel independent.



Proof. The Local Church-Rosser Theorem has originally been shown for graph transformation systems in [ER76] and it is shown in [EEPT06b] in the categorical framework of "high-level replacement systems" based on weak adhesive HLR categories. In [EGH10] it is shown that the theorem is also valid in $\mathcal{M}$-adhesive categories, and the category $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ is an $\mathcal{M}$-adhesive category (see Fact A.1.11). Hence, the Local Church-Rosser Theorem is also valid for AHL-net transformations. □

Concept 5.1.5 (Independent Platform Evolutions)**.** We say that two platform evolutions are independent if they can be applied in any order leading to the same result. The question whether two platform evolutions are independent can be analysed by investigating whether the corresponding AHL-net transformations are parallel or sequentially independent. Using the result in Theorem 5.1.4, we know for independent platform evolutions that the evolutions can be performed in any order leading to the same result. Note that this also implies an analysis of the existence of conflicts between the platform evolutions. If the platform evolutions are not independent, then there is some conflict between them. Conflicts between direct transformations in an $\mathcal{M}$-adhesive transformation system (like the transformation system of AHL-nets) are usually expressed and analysed using *critical pairs*. The general theory of critical pairs for weak adhesive HLR categories (which is also applicable to $\mathcal{M}$-adhesive categories, see [EGH10]) is presented in Section 6.3 of [EEPT06b]. △

Example 5.1.6 (Independent Platform Evolutions)**.** The compatible platform evolutions from Example 5.1.1 shown in Figure 5.2 are an example of independent platform evolutions, because the corresponding AHL-net transformations are parallel and sequentially independent direct transformations. ♢

One of the benefits of sequentially independent direct transformations is the possibility to synthesise a parallel production such that the application of that production has the same effect as the sequential application of the transformations in an arbitrary order. Vice versa, it is possible to analyse a direct transformation using a parallel production into sequentially independent direct transformations which means that the direct transformation can be divided into smaller steps, allowing to consider particular portions of the modification separately.

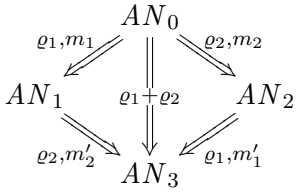
This is shown for the general case of weak adhesive HLR categories in the Parallelism Theorem in [EEPT06b]. In the following we review the Parallelism Theorem for our special case of AHL-net transformations.

Theorem 5.1.7 (Parallelism Theorem for AHL-Net Transformations)**.**

1. (Synthesis) *Given a sequentially independent direct transformation sequence* $AN_0 \Rightarrow AN_1 \Rightarrow AN_3$ *via productions* ϱ_1 *and* ϱ_2*, then there is a construction leading to a parallel transformation* $AN_0 \Rightarrow AN_3$ *via parallel production* $\varrho_1 + \varrho_2$[21]*, called a* synthesis construction.

2. (Analysis) *Given a parallel transformation* $AN_0 \Rightarrow AN_3$ *via* $\varrho_1 + \varrho_2$*, then there is a construction leading to two sequentially independent transformation sequences* $AN_0 \Rightarrow AN_1 \Rightarrow AN_3$ *via* ϱ_1 *and* ϱ_2*, and* $AN_0 \Rightarrow AN_2 \Rightarrow AN_3$ *via* ϱ_2 *and* ϱ_1*, called an* analysis construction.

3. (Bijective correspondence) *The synthesis and analysis constructions are inverse to each other up to isomorphism:*



[21] With $\varrho_1 + \varrho_2$ we denote the binary case of a parallel production $\coprod_{i \in \{1,2\}} \varrho_i$ as defined in Definition A.3.4.

Proof. The Parallelism Theorem is shown in [EEPT06b] in the categorical framework of "high-level replacement systems" based on weak adhesive HLR categories. In [EGH10] it is shown that the theorem is also valid in $\mathcal{M}$-adhesive categories, and the category (**AHLNets**, $\mathcal{M}_{AHL}$) is an $\mathcal{M}$-adhesive category (see Fact A.1.11). The Parallelism Theorem in [EEPT06b] requires that the weak adhesive HLR categories has coproducts compatible with $\mathcal{M}$, i. e. for morphisms $f, g \in \mathcal{M}$ it is required that also $f + g \in \mathcal{M}$. This is necessary for the well-defined construction of parallel productions. The well-definedness of parallel productions for AHL-nets (Definition A.3.4) is shown in Fact A.3.5. Hence, the Parallelism Theorem is also valid for AHL-net transformations. □

Example 5.1.8 (Parallel Platform Evolution)**.** Consider the sequential independent transformation sequence $Platform \overset{\varrho_1,m_1}{\Longrightarrow} Platform_1 \overset{\varrho_2,m_2'}{\Longrightarrow} Platform_3$ in Figure 5.2 on page 118. The production ϱ_1 is shown in Figure 3.3 on page 39 and the production ϱ_2 is shown in Figure 5.1 on page 117. Using the synthesis construction from Theorem 5.1.7, we obtain a parallel production $\varrho_1 + \varrho_2$ as shown in Figure 5.3, and we know that there is a direct transformation $Platform \Rightarrow Platform_3$ via production $\varrho_1 + \varrho_2$.

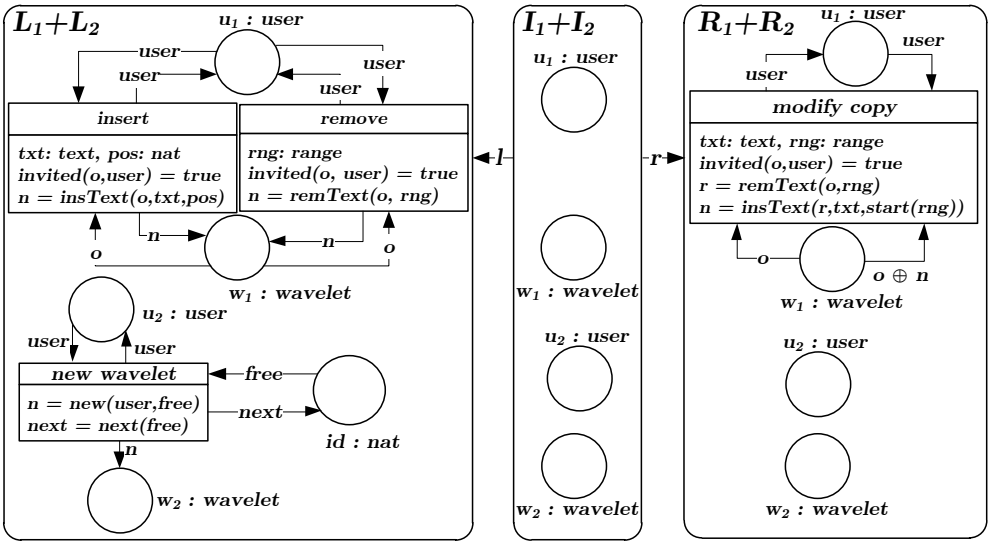


Figure 5.3: Parallel production $\varrho_1 + \varrho_2$

From item 2 of Theorem 5.1.7 we know that the parallel production can again be analysed into sequential independent transformation sequences $Platform \overset{\varrho_1,m_1}{\Longrightarrow} Platform_1 \overset{\varrho_2,m_2'}{\Longrightarrow} Platform_3$ and $Platform \overset{\varrho_2,m_2}{\Longrightarrow} Platform_2 \overset{\varrho_1,m_1'}{\Longrightarrow} Platform_3$ as shown in Figure 5.2. ♢

As we have seen, independence of platform evolutions allows us to change the order of application of the evolutions as well as to perform the evolutions as one parallel evolution at once. This has several benefits for the modelling of the evolution of platforms. On the one hand, it is possible to merge several different evolutions into one update of a platform. On the other hand, it is possible to split an update of a platform into several evolutions, allowing us to investigate and exclude problematic parts.

Of course, there are also platform evolutions that are not independent. In the case that two evolutions are parallel dependent, this means that we have to choose one of the evolutions, because the application of one evolution prevents the application of the other one. In the case the the evolutions are sequential dependent, it means that it is possible to apply both evolutions, but it is not possible to change the order of applications. However,

it would still be desirable to have also the possibility to synthesise or analyse concurrent evolutions that correspond to the sequential application of different sequentially dependent platform evolutions. An example of sequentially dependent platform evolutions is given in the following.

Example 5.1.9 (Sequentially Dependent Platform Evolutions)**.** Consider again the production ϱ in Figure 3.3 on Figure 3.3, and also the production ϱ_3 in Figure 5.4. The production describes the deletion of a transitions *invite user* and *modify copy* and the introduction of a new transition *modify copy*′ that is almost equal to the *modify copy* transition, apart from the fact that it does not require the involved user to be invited in the wavelet the user modifies.

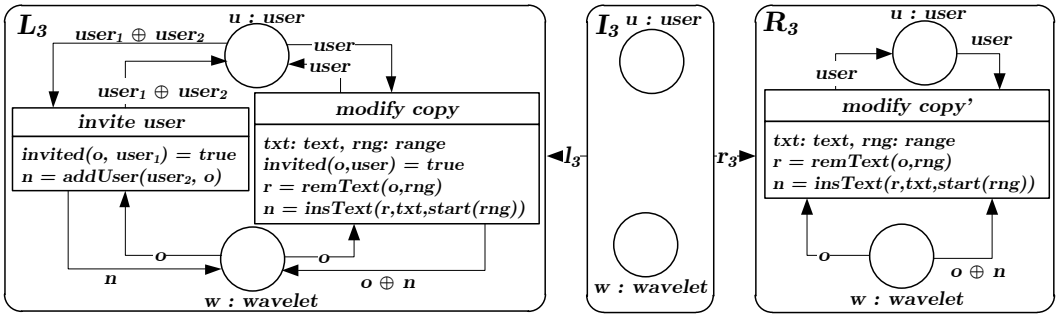


Figure 5.4: Production ϱ_3 for platform evolution

The production can be applied to the AHL-net *Platform*$_1$ in Figure 5.2 at an inclusion match m_3. The result of the direct transformation *Platform*$_1 \overset{\varrho_3,m_3}{\Longrightarrow}$ *Platform*$_4$ is depicted in Figure 5.5.

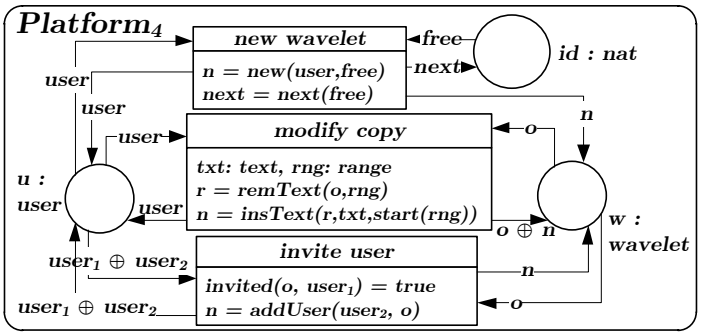
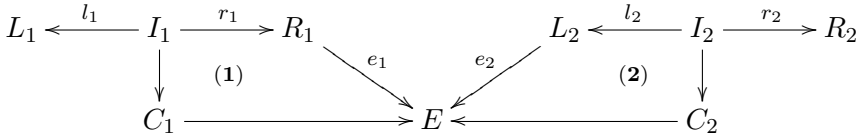


Figure 5.5: AHL-net *Platform*$_4$

Considering also the top row of Figure 5.2, we have a sequence of sequentially dependent direct transformations *Platform* $\overset{\varrho_1,m_1}{\Longrightarrow}$ *Platform*$_1$ $\overset{\varrho_3,m_3}{\Longrightarrow}$ *Platform*$_4$. This can be seen by the simple fact that the production ϱ_3 is not applicable to the AHL-net *Platform*, because the original platform does not consist of an action *modify copy*. The application of production ϱ_3 depends on the previous application of the production ϱ_1 which can be expressed using the following notion of E-dependency relations which were presented in [EEPT06b] for the general framework of weak adhesive HLR categories with $\mathcal{E}'$-$\mathcal{M}'$ pair factorisations (see also Definition A.4.2). ♢

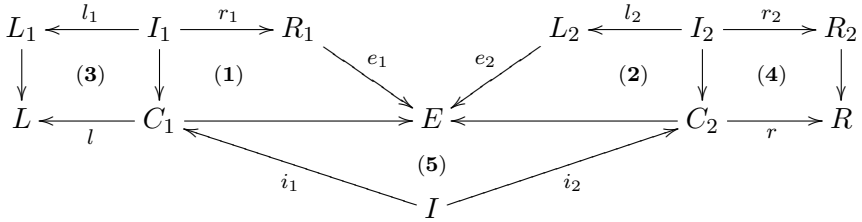
Definition 5.1.10 (E-Dependency Relation of AHL-Net Transformations)**.** Given the class $\mathcal{E}'$ of pairs of jointly epimorphic AHL-morphisms, and let ϱ_1 and ϱ_2 be two productions for AHL-nets with $\varrho_i = L_i \overset{l_i}{\leftarrow} I_i \overset{r_i}{\rightarrow} R_i$ for $i = 1, 2$. An AHL-net E with AHL-morphisms

$e_1 : R_1 \to E$ and $e_2 : L_2 \to E$ is an *E-dependency relation* for ϱ_1 and ϱ_2 if $(e_1, e_2) \in \mathcal{E}'$ and the pushout complements (1) and (2) below exist:

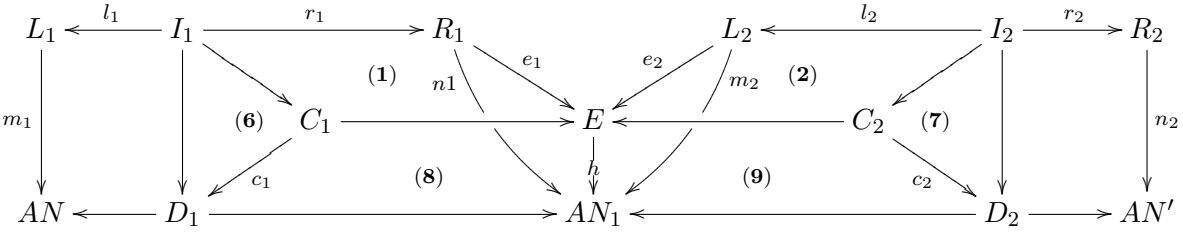


△

Definition 5.1.11 (*E*-Concurrent Production and *E*-Related Transformation of AHL-Nets)**.** Given an *E*-dependency relation $(e_1, e_2) \in \mathcal{E}'$ for the productions ϱ_1 and ϱ_2, the *E-concurrent production* $\varrho_1 *_E \varrho_2$ is defined by $\varrho_1 *_E \varrho_2 = L \xleftarrow{l \circ i_1} I \xrightarrow{r \circ i_2} R$ as shown in the following diagram, where (3) and (4) are pushouts and (5) is a pullback:



A transformation sequence $AN \overset{\varrho_1, m_1}{\Longrightarrow} AN_1 \overset{\varrho_2, m_2}{\Longrightarrow} AN'$ is called *E-related* if there exists $h : E \to AN_1$ with $h \circ e_1 = n_1$ and $h \circ e_2 = m_2$ and there are morphisms $c_1 : C_1 \to D_1$ and $c_2 : C_2 \to D_2$ such that (6) and (7) commute and (8) and (9) are pushouts:



△

The following Concurrency Theorem for AHL-net transformations is an instantiation of the Concurrency Theorem for weak adhesive HLR categories in [EEPT06b].

Theorem 5.1.12 (Concurrency Theorem for AHL-Net Transformations)**.** *Let* $R_1 \overset{e_1}{\to} E \overset{e_2}{\leftarrow} L_2$ *be an E-dependency relation for the productions for AHL-nets* ϱ_1 *and* ϱ_2*, and* $\varrho_1 *_E \varrho_2$ *the corresponding E-concurrent production.*

1. (Synthesis) *Given an E-related transformation sequence* $AN \Rightarrow AN_1 \Rightarrow AN'$ *via* ϱ_1 *and* ϱ_2*, then there is a* synthesis construction *leading to a direct transformation* $AN \Rightarrow AN'$ *via* $\varrho_1 *_E \varrho_2$*.*
2. (Analysis) *Given a direct transformation* $AN \Rightarrow AN'$ *via* $\varrho_1 *_E \varrho_2$*, then there is an* analysis construction *leading to an E-related transformation sequence* $AN \Rightarrow AN_1 \Rightarrow AN'$ *via* ϱ_1 *and* ϱ_2*.*
3. (Bijective correspondence) *The synthesis and analysis constructions are inverse to each other up to isomorphism.*

Proof. The Concurrency Theorem is shown in [EEPT06b] in the categorical framework of "high-level replacement systems" based on weak adhesive HLR categories. In [EGH10] it is shown that the theorem is also valid in $\mathcal{M}$-adhesive categories, and the category (**AHLNets**, $\mathcal{M}_{AHL}$) is an $\mathcal{M}$-adhesive category (see Fact A.1.11). Therefore, the items 1 and 2 follow

directly from the corresponding items in the Concurrency Theorem in [EEPT06b]. For the third item, in [EEPT06b] it is required that the morphism class $\mathcal{E}'$ consists only of jointly epimorphic pairs which is the case for our choice of morphism class $\mathcal{E}'$. Hence, the Concurrency Theorem is valid for AHL-net transformations. □

Fact 5.1.13 (Construction of E-Related AHL-Net Transformations)**.** *For each pair of direct transformations* $AN \overset{\varrho_1,m_1}{\Longrightarrow} AN_1 \overset{\varrho_2,m_2}{\Longrightarrow} AN'$ *we have an* E*-dependency relation* E *such that* $AN \overset{\varrho_1,m_1}{\Longrightarrow} AN_1 \overset{\varrho_2,m_2}{\Longrightarrow} AN'$ *is* E*-related. Given the comatch* $n_1 : R_1 \to AN_1$ *of* $AN \overset{\varrho_1,m_1}{\Longrightarrow} AN_1$*, the* E*-dependency relation is obtained as* $\mathcal{E}'$-$\mathcal{M}'_{AHL}$ *pair factorisation (see Fact A.4.4)* $(e_1, e_2) \in \mathcal{E}'$ *and* $h \in \mathcal{M}'_{AHL}$ *of* n_1 *and* m_2*.*

Proof. The construction of E-related transformations is shown in Fact 5.29 of [EEPT06b] for weak adhesive HLR categories that have the $\mathcal{M}$-$\mathcal{M}'$ pushout-pullback decomposition property. The $\mathcal{M}$-adhesive category $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ satisfies all HLR properties that are required for the proof, and we have shown in Fact A.4.11 that $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ with the class $\mathcal{M}'_{AHL}$ of all monomorphisms has the $\mathcal{M}$-$\mathcal{M}'$ pushout-pullback decomposition property. Hence, the construction works also for E-related transformations of AHL-nets. □

Example 5.1.14 (Concurrent Platform Evolution)**.** Consider again the sequentially dependent transformation sequence $Platform \overset{\varrho_1,m_1}{\Longrightarrow} Platform_1 \overset{\varrho_3,m_3}{\Longrightarrow} Platform_4$ from Example 5.1.9. We can use the construction Fact 5.1.13 to obtain an E-dependency relation $(e_1, e_2) \in \mathcal{E}'$ for ϱ_1 and ϱ_3. The construction of the E-dependency relation and the corresponding E-concurrent production is depicted in Figure 5.6, where all morphisms are inclusions, and we omitted all transition assignments. Note that the production ϱ_1 and ϱ_2 as well as the platforms all have the signature Σ-*Wave* in Table 3.1 and the algebra A in Table 3.2 as data type part. So all AHL-nets involved in the construction do also have the same data type part.

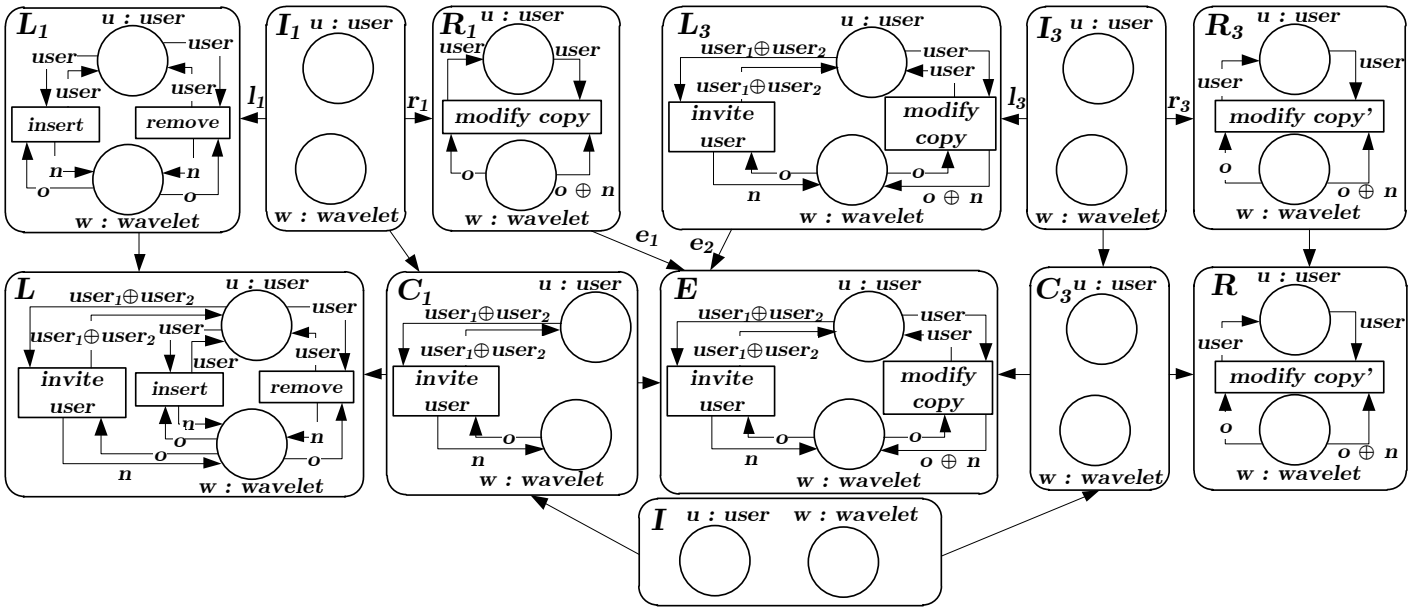


Figure 5.6: Construction of E-concurrent production $\varrho_1 *_E \varrho_3$

The coproduct $R_1 + L_2$ contains one instance of the *invite user* transition and two instances of the *modify copy* transition, because it is the disjoint union of R_1 and L_2. Since both of the *modify copy* transitions in R_1 and L_2 are mapped to the same transition in $Platform_1$ by n_1 and m_2, respectively, in the $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisation of $[n_1, m_2] : R_1 + L_2 \to Platform_1$,

the two transitions are identified. As a result, the morphism e_2 is an identity. Therefore, by pushout complement of e_2 and l_3 and consecutive pushout over $I_3 \to C_3$ and r_3, we obtain trivial pushouts where all vertical morphisms are identities.

The pushout complement of e_1 and r_1 yields an AHL-net where the transition *modify copy* has been removed from E, because the transition is matched by n_1, but it does not have a preimage in I_1. Then the pushout construction over $I_1 \to C_1$ and l_1 leads to an AHL-net where L_1 and C_1 are merged at the common places u and w. So the result L contains all three transitions *insert*, *remove* and *invite user*, connected to u and w.

Finally, the pullback over $C_1 \to E$ and $C_3 \to E$ consists only of the places u and w, since there are no common transitions shared by C_1 and C_3. The E-concurrent production $\varrho_1 *_E \varrho_3$ is obtained by composition $L \leftarrow I \to R$ of all morphisms along the bottom of Figure 5.6. The E-concurrent production describes the deletion of transitions *insert*, *remove* and *invite user* and the creation of a new transition *modify copy*$'$.

It can be verified that $Platform \Rightarrow Platform_4$ is an E-related transformation that can be obtained as a single direct transformation via $\varrho_1 *_E \varrho_4$ without the intermediate creation and deletion of transition *modify copy*. ♢

5.2 Independence of Scenario Evolutions

In the previous section, we reviewed the Local Church-Rosser, Parallelism and Concurrency Theorems for weak adhesive HLR and $\mathcal{M}$-adhesive categories [EEPT06b, EGH10], and we showed that these results can also be used for the analysis of the evolution of communication platforms. In this section we show that the same results, however, with some additional conditions, are also applicable to the evolution of scenarios.

Remark 5.2.1 (Analysis Results for $\mathcal{M}$-adhesive Categories). The Local Church-Rosser, Parallelism and Concurrency Theorems as well as the construction of E-related transformations, presented in Section 5.1, are shown in [EEPT06b] in the general framework of weak adhesive HLR categories. In [EGH10] it is shown that the results are also valid for $\mathcal{M}$-adhesive categories $(\mathbf{C}, \mathcal{M})$, provided that the following additional requirements are satisfied:

1. For the Parallelism Theorem it is required that $(\mathbf{C}, \mathcal{M})$ has binary coproducts compatible with $\mathcal{M}$ (see Definition A.3.2).

2. For the bijective correspondence of analysis and synthesis construction in the Concurrency Theorem it is required that every morphism pair $(e_1, e_2) \in \mathcal{E}'$ is jointly epimorphic, i. e. for each $\mathbf{C}$-morphisms f and g with $f \circ e_1 = g \circ e_1$ and $f \circ e_2 = g \circ e_2$ it follows that $f = g$.

3. For the construction of E-related transformations it is required that $(\mathbf{C}, \mathcal{M})$ has $\mathcal{E}'$-$\mathcal{M}'$ pair factorisations (see Definition A.4.2) such that the $\mathcal{M}$-$\mathcal{M}'$ pushout-pullback decomposition property holds (see Definition A.4.10).

Unfortunately there are no suitable classes $\mathcal{M}$ of monomorphisms such that the categories of (instantiated) AHL-processes become $\mathcal{M}$-adhesive categories. However, AHL-processes can be regarded as a special kind of *typed AHL-nets*, i. e. for each AHL-net AN the category $\mathbf{Proc}(\mathbf{AN})$ is a subcategory of the slice category $\mathbf{AHLNets} \setminus AN$, and the category $\mathbf{AHLProcs}$ is a subcategory of the arrow category $\mathbf{AHLNets}^{\to}$. Since $\mathcal{M}$-adhesive categories as well as weak adhesive HLR categories are closed under slice and functor category construction[22] (see Theorem 4.15 in [EEPT06b]), the categories $(\mathbf{AHLNets} \setminus AN, \mathcal{M}_{AHL})$

[22] The arrow category $\mathbf{AHLNets}^{\to}$ is isomorphic to the functor category $[\mathbf{2}, \mathbf{AHLNets}]$, where $\mathbf{2}$ is the small category $\bullet \to \bullet$ with two objects and one non-trivial morphism between them.

and $(\mathbf{AHLNets}^{\to}, \mathcal{M}_{AHL}^{\to})$ are $\mathcal{M}$-adhesive categories, where $\mathcal{M}_{AHL}^{\to} = \{(m_1, m_2) \mid m_1, m_2 \in \mathcal{M}_{AHL}\}$.

Moreover, the categories $(\mathbf{wInst}, \mathcal{M}_{AHL})$ and $(\mathbf{Inst}, \mathcal{M}_{AHL})$ of (weak) instantiations are $\mathcal{M}$-adhesive categories as shown in Fact A.5.9, and we have the following additional properties:

1. Coproducts in the slice category $\mathbf{AHLNets} \setminus AN$ can be constructed in **AHLNets**, and therefore are compatible with $\mathcal{M}_{AHL}$ (see Fact A.3.3).

   Coproducts in the arrow category (functor category) $\mathbf{AHLNets}^{\to}$ can be constructed point-wise in **AHLNets** which by definition of $\mathcal{M}_{AHL}^{\to}$ and compatibility of coproducts in **AHLNets** with $\mathcal{M}_{AHL}$ implies that coproducts in $\mathbf{AHLNets}^{\to}$ are compatible with $\mathcal{M}_{AHL}^{\to}$.

   Coproducts in **wInst** and **Inst** are compatible with $\mathcal{M}_{AHL}$ as shown in Fact A.3.8.

2. For the categories $\mathbf{AHLNets} \setminus AN$, **wInst** and **Inst**, the class $\mathcal{E}'$ can be chosen as the class of all jointly epimorphic AHL-morphisms. For the category $\mathbf{AHLNets}^{\to}$ we choose the class $\mathcal{E}'^{\to} = \{(e_1, e_2) \mid e_1, e_2 \in \mathcal{E}'\}$ which is a class of jointly epimorphic $\mathbf{AHLNets}^{\to}$-morphisms.

3. As shown in Fact A.4.7, the category $\mathbf{AHLNets} \setminus AN$ has $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisations for the classes $\mathcal{E}$ of all epimorphisms and $\mathcal{M}'_{AHL}$ of all monomorphisms in **AHLNets**, and the category $\mathbf{AHLNets}^{\to}$ has $\mathcal{E}^{\to}$-$\mathcal{M}'^{\to}_{AHL}$ factorisations for the classes $\mathcal{E}^{\to}$ of all epimorphisms and $\mathcal{M}'^{\to}_{AHL}$ of all monomorphisms in $\mathbf{AHLNets}^{\to}$.

   Due to the fact that $\mathbf{AHLNets} \setminus AN$ and $\mathbf{AHLNets}^{\to}$ have coproducts, this means that $\mathbf{AHLNets} \setminus AN$ has $\mathcal{E}'$-$\mathcal{M}'_{AHL}$ pair factorisations that can be constructed similar to the $\mathcal{E}'$-$\mathcal{M}'_{AHL}$ pair factorisations in **AHLNets** (see Fact A.4.4), and analogously $\mathbf{AHLNets}^{\to}$ has $\mathcal{E}'^{\to}$-$\mathcal{M}'^{\to}_{AHL}$ pair factorisations. The fact that **wInst** and **Inst** have $\mathcal{E}'$-$\mathcal{M}'_{AHL}$ pair factorisations is shown in Fact A.4.6.

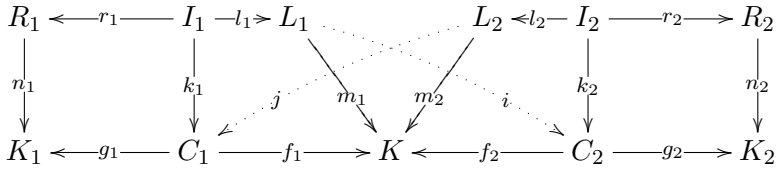
   Furthermore, in Fact A.4.11 it is shown that for classes $\mathcal{M}_{AHL}$ and $\mathcal{M}'_{AHL}$ the pushout pullback decomposition property holds. Since pushouts and pullbacks in $\mathbf{AHLNets}^{\to}$ are constructed componentwise in **AHLNets**, the property also holds for $\mathcal{M}_{AHL}^{\to}$ and $\mathcal{M}'^{\to}_{AHL}$ with each component being an $\mathcal{M}_{AHL}$ respectively $\mathcal{M}'^{\to}_{AHL}$-morphism.

Hence, the analysis results presented in Section 5.1 for the $\mathcal{M}$-adhesive category of AHL-nets $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ can also be applied to the $\mathcal{M}$-adhesive categories $(\mathbf{AHLNets} \setminus AN, \mathcal{M}_{AHL})$ for all AHL-nets AN, $(\mathbf{AHLNets}^{\to}, \mathcal{M}_{AHL}^{\to})$, $(\mathbf{wInst}, \mathcal{M}_{AHL})$ and $(\mathbf{Inst}, \mathcal{M}_{AHL})$.

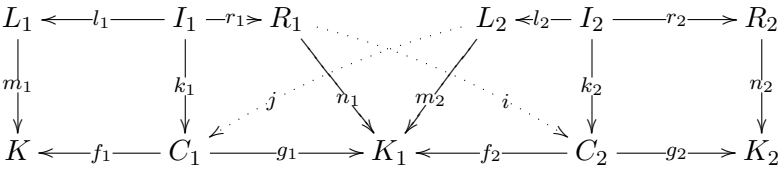
Transformations of (instantiated) AHL-processes are merely special cases of transformations in the categories above with the additional requirement that the AHL-process net properties (see Definition 4.1.1) are not violated by the (corresponding part of the) transformation. Thus, in order to make the analysis results also applicable to (instantiated) AHL-processes, it suffices to investigate the analysis of transformations of AHL-process nets, as we do in the following. △

For the parallel independence of AHL-net transformations, it is required that the left-hand side of each production can be also matched into the context net of the transformation using the respective other production. For the parallel independence of AHL-process nets, it is additionally required that the induced match into the result of the respective other transformation satisfies the transformation condition for AHL-process nets (see Definition 4.2.10 on page 66). Analogously, also for the sequential independence, we additionally require the satisfaction of the transformation condition for the corresponding matches.

Definition 5.2.2 (Strong Parallel and Sequential Independence of AHL-Process Net Transformations)**.** Two direct transformations of AHL-process nets $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1$ and $K \overset{\varrho_2,m_2}{\Longrightarrow} K_2$ are called *strongly parallel independent* if there exist morphisms $i : L_1 \to C_2$ and $j : L_2 \to C_1$ such that $f_2 \circ i = m_1$ and $f_1 \circ j = m_2$, and we have that ϱ_1 and $g_2 \circ i$ as well as ϱ_2 and $g_1 \circ j$ satisfy the transformation condition for AHL-process nets.



Moreover, two direct transformations of AHL-nets $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2}{\Longrightarrow} K_2$ are called *strongly sequentially independent* if there exist morphisms $i : R_1 \to C_2$ and $j : L_2 \to C_1$ such that $f_2 \circ i = n_1$ and $g_1 \circ j = m_2$, and we have that ϱ_1^{-1} and $g_2 \circ i$ as well as ϱ_2 and $f_1 \circ j$ satisfy the transformation condition for AHL-process nets.

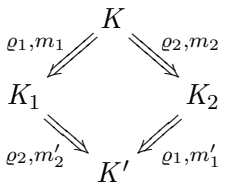


△

Remark 5.2.3 (Relationship between Strictly Parallel and Sequential Independence). Note that $AN_0 \overset{p_1}{\Rightarrow} AN_1 \overset{p_2}{\Rightarrow} AN_2$ are strongly sequentially independent if and only if $AN_0 \overset{p_1^{-1}}{\Leftarrow} AN_1 \overset{p_2}{\Rightarrow} AN_2$ are strongly parallel independent. △

Theorem 5.2.4 (Local Church-Rosser Theorem for AHL-Process Net Transformations)**.** *Given two strongly parallel independent direct transformations $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1$ and $K \overset{\varrho_2,m_2}{\Longrightarrow} K_2$ of AHL-process nets, then there is an AHL-process net K' together with direct transformations $K_1 \overset{\varrho_2,m_2'}{\Longrightarrow} K'$ and $K_2 \overset{\varrho_1,m_1'}{\Longrightarrow} K'$ of AHL-process nets such that $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2'}{\Longrightarrow} K'$ and $K \overset{\varrho_2,m_2}{\Longrightarrow} K_2 \overset{\varrho_1,m_1'}{\Longrightarrow} K'$ are strongly sequentially independent.*

Given two strongly sequentially independent direct transformations $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2'}{\Longrightarrow} K'$ of AHL-process nets, there are an AHL-process net K_2 and direct transformations $K \overset{\varrho_2,m_2}{\Longrightarrow} K_2 \overset{\varrho_1,m_1'}{\Longrightarrow} K'$ of AHL-process nets such that $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1$ and $K \overset{\varrho_2,m_2}{\Longrightarrow} K_2$ are strongly parallel independent.



Proof-Idea. The proof works very similar to the corresponding proof of the Local Church-Rosser Theorem for weak adhesive HLR categories in [EEPT06b]. The fact that the resulting transformations are direct transformations of AHL-process nets is ensured by satisfaction of the transformation condition due to the fact that the independences are strong (see Definition 5.2.2). Strictness of the resulting independences follows from the fact that the transformation condition is not only sufficient but also necessary for the existence of direct transformations of AHL-process nets. For the detailed proof we refer to Section B.12 on page 258. □

Remark 5.2.5 (Local Church-Rosser Diagram of AHL-Process Net Transformations). In the proof of Theorem 5.2.4 we show that the matches that are used for the synthesised direct transformations are exactly those morphisms for which we require the satisfaction of the transformation conditions in the definitions of strongly parallel and strongly sequentially independent transformations. Since satisfaction of the transformation condition is not only a sufficient but also a necessary condition for the direct transformation of AHL-process nets, this means that given a *local Curch-Rosser diagram* of parallel and sequentially independent transformations of AHL-process nets as in Theorem 5.2.4, then the transformations are also strong parallel and strong sequentially independent. △

Concept 5.2.6 (Independent Scenario Evolutions)**.** We say that two scenario evolutions are independent if they can be applied in any order leading to the same result. The question whether two scenario evolutions are independent can be analysed by investigating whether the corresponding AHL-process net transformations are strongly parallel or strongly sequentially independent. Using the result in Theorem 5.2.4 and the considerations in Remark 5.2.1, we know for independent scenario evolutions that the evolutions can be performed in any order leading to the same result. Note that this also implies an analysis of the existence of conflicts between the platform evolutions. If the platform evolutions are not independent, then there is some conflict between them. Conflicts between direct transformations in an $\mathcal{M}$-adhesive transformation system (like the transformation system of AHL-nets) are usually expressed and analysed using *critical pairs*. The general theory of critical pairs for weak adhesive HLR categories (which is also applicable to $\mathcal{M}$-adhesive categories, see [EGH10]) is presented in Section 6.3 of [EEPT06b]. △

Example 5.2.7 (Independent Scenario Evolutions)**.** Consider the productions ϱ_1 and ϱ_2 in Figure 5.7 that are productions for AHL-processes of the AHL-net *Platform* in Figure 3.2 on page 35. Figure 5.8 shows two direct transformations $W_0 \overset{\varrho_1,m_1}{\Longrightarrow} W_1$ and $W_0 \overset{\varrho_2,m_2}{\Longrightarrow} W_2$ of an AHL-process $w_0 : W_0 \to \mathit{Platform}$ via ϱ_1 and ϱ_2, respectively, at inclusion matches.

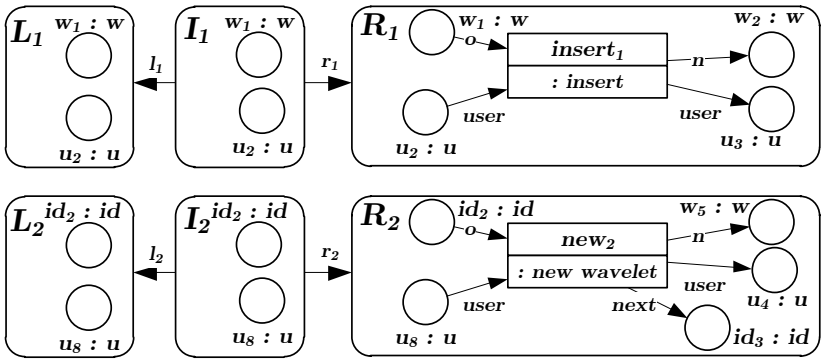


Figure 5.7: Productions ϱ_1 and ϱ_2 for AHL-processes of AHL-net *Platform*

Since the productions ϱ_1 and ϱ_2 are non-deleting, the required morphisms $i : L_1 \to C_2$ and $j : L_2 \to C_1$ for the parallel independence of the productions exist, because we have $C_2 \cong W_0 \cong C_1$, and accordingly i and j are isomorphic to m_1 and m_2, respectively. Moreover, since ϱ_1 and m_1 as well as ϱ_2 and m_2 satisfy the transformation condition, the same holds for ϱ_1 and i, and ϱ_2 and j. Thus, the direct transformations are not only parallel independent, but strongly parallel independent.

This means that there are also direct transformations $W_1 \Longrightarrow W_3$ and $W_2 \Longrightarrow W_3$ via ϱ_2 and ϱ_1, respectively, as shown in Figure 5.8, and we have that the transformation sequences $W_0 \Rightarrow W_1 \Rightarrow W_3$ and $W_0 \Rightarrow W_2 \Rightarrow W_3$ both are strongly sequentially independent. ♢

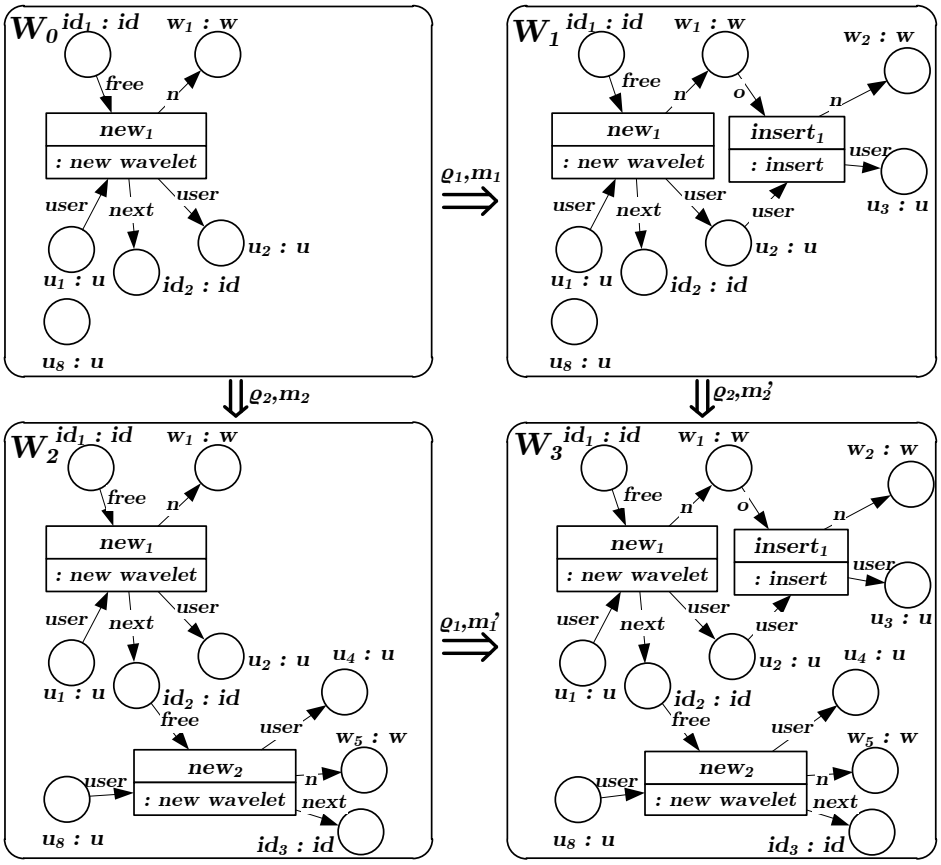


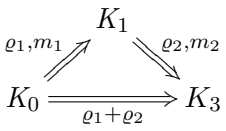
Figure 5.8: Compatible scenario evolutions via productions ϱ_1 and ϱ_2

Theorem 5.2.8 (Parallelism Theorem for AHL-Process Net Transformations)**.**

1. (Synthesis) *Given a sequentially independent direct transformation sequence of AHL-process nets $K_0 \Rightarrow K_1 \Rightarrow K_3$ via productions ϱ_1 and ϱ_2, then there is a construction leading to a parallel transformation of AHL-process nets $K_0 \Rightarrow K_3$ via parallel production $\varrho_1 + \varrho_2$, called a* synthesis construction.

2. (Analysis) *Given a parallel transformation of AHL-process nets $K_0 \Rightarrow K_3$ via $\varrho_1 + \varrho_2$ at match $[m_1, m_2]$[23] such that ϱ_1 and m_1 satisfy the transformation condition for AHL-process nets (Definition 4.2.10), then there is a construction leading to a sequentially independent transformation sequence of AHL-process nets $K_0 \Rightarrow K_1 \Rightarrow K_3$ via ϱ_1 and ϱ_2, called an* analysis construction.

3. (Bijective correspondence) *The synthesis and analysis constructions are inverse to each other up to isomorphism:*



[23] $[m_1, m_2]$ denotes a mediating morphism for $m_1 : L_1 \to K_0$ and $m_2 : L_2 \to K_0$ such that for coproduct injections $\iota_1 : L_1 \to L_1 + L_2$ and $\iota_2 : L_2 \to L_1 + L_2$: $[m_1, m_2] \circ \iota_1 = m_1$ and $[m_1, m_2] \circ \iota_2 = m_2$. Note that for every morphism $m : L_1 + L_2 \to K_0$, we obtain a corresponding pair of morphisms m_1 and m_2 such that $m = [m_1, m_2]$ by composition of m with the coproduct injections.

Proof. 1. *(Synthesis)* This follows directly from item 1 of Theorem 5.1.7 and the fact that K_3 is an AHL-process net.

2. *(Analysis)* From item 2 of Theorem 5.1.7 we obtain a sequentially independent transformation sequence of AHL-nets $K_0 \Rightarrow K_1 \Rightarrow K_3$ via ϱ_1 at match m_1 and ϱ_2 at match m_2. Then, ϱ_1 and m_1 satisfying the transformation condition for AHL-process nets by Theorem 4.2.11 implies that $K_0 \Rightarrow K_1$ is a transformation of AHL-process nets, and together with the fact that also K_3 is an AHL-process net we obtain that also $K_1 \Rightarrow K_3$ is a transformation of AHL-process net.

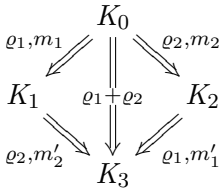
3. *(Bijective correspondence)* The synthesis and analysis constructions in Theorem 5.1.7 are inverse to each other. Moreover, given a sequentially independent direct transformation sequence of AHL-process nets $K_0 \Rightarrow K_1 \Rightarrow K_3$ via productions ϱ_1 and ϱ_2 at matches m_1 and m_2, respectively, by Theorem 4.2.11 we have that ϱ_1 and m_1 satisfy the transformation condition for AHL-process nets which is necessary to analyse the corresponding synthesised parallel transformation.

□

Corollary 5.2.9 (Parallelism Theorem for AHL-Process Nets and Strong Independence)**.**

1. (Strong Independence: Synthesis) *Given a strongly sequentially independent direct transformation sequence of AHL-process nets $K_0 \Rightarrow K_1 \Rightarrow K_3$ via productions ϱ_1 and ϱ_2, then the synthesis construction in Theorem 5.2.8 provides a direct transformation $K_0 \Rightarrow K_3$ via $\varrho_1 + \varrho_2$ at match $[m_1, m_2]$ such that ϱ_1 and m_1 as well as ϱ_2 and m_2 satisfy the transformation condition for AHL-process nets.*

2. (Strong Independence: Analysis) *Given a direct transformation $K_0 \Rightarrow K_3$ via $\varrho_1 + \varrho_2$ at match $[m_1, m_2]$ such that ϱ_1 and m_1 as well as ϱ_2 and m_2 satisfy the transformation condition for AHL-process nets, then the analysis construction in Theorem 5.2.8 provides two strongly sequentially independent transformation sequences $K_0 \Rightarrow K_1 \Rightarrow K_3$ via ϱ_1 and ϱ_2, and $K_0 \Rightarrow K_2 \Rightarrow K_3$ via ϱ_2 and ϱ_1.*



Proof. Note that given a parallel transformation of AHL-process nets $K_0 \Rightarrow K_3$ via $\varrho_1 + \varrho_2$ at match $m_1 + m_2$ such that ϱ_2 and m_2 satisfy the transformation condition for AHL-process nets, then the analysis construction in Theorem 5.2.8 provides also a sequentially independent transformation sequence of AHL-process nets $K_0 \Rightarrow K_2 \Rightarrow K_3$ via ϱ_2 at match m_2 and ϱ_1 at match m_1. This follows from the fact that coproducts are commutative up to isomorphism, i. e. for parallel production $\varrho_1 + \varrho_2$ and match $m_1 + m_2$, we have $\varrho_1 + \varrho_2 \cong \varrho_2 + \varrho_1$ and $m_1 + m_2 \cong m_2 + m_1$.

1. *(Strong Independence: Synthesis)* It suffices to show that ϱ_1 and m_1 as well as ϱ_2 and m_2 satisfy the transformation condition for AHL-process nets. In the general proof of the Parallelism Theorem (Theorem 5.18 of [EEPT06b]) it is shown that the match $[m_1, m_2]$ of the synthesised parallel transformation corresponds to the matches m_1 and m_2 of parallel independent transformations $K_0 \Rightarrow K_1$ via ϱ_1 and $K_0 \Rightarrow K_2$ via ϱ_2, obtained by the Local Church-Rosser Theorem (see Theorem 5.1.4). Due to the fact that the

sequentially independence of the transformation sequence $K_0 \Rightarrow K_1 \Rightarrow K_3$ is strong, by Theorem 5.2.4 we have that the transformations $K_0 \Rightarrow K_1$ via ϱ_1 and $K_0 \Rightarrow K_2$ via ϱ_2 are strongly parallel independent direct transformations of AHL-process nets which in turn by Theorem 4.2.11 implies that ϱ_1 and m_1 as well as ϱ_2 and m_2 satisfy the transformation condition.

2. *(Strong Independence: Analysis)* Using the symmetry shown above, we can use the analysis construction in Theorem 5.2.14 to obtain two sequentially independent transformation sequences $K_0 \Rightarrow K_1 \Rightarrow K_3$ via ϱ_1 and ϱ_2, and $K_0 \Rightarrow K_2 \Rightarrow K_3$ via ϱ_2 and ϱ_1, and as pointed out in Remark 5.2.5 the transformation sequences are strongly sequentially independent.

□

Example 5.2.10 (Parallel Scenario Evolution)**.** Consider again the independent scenario evolutions in Example 5.2.7, modelled by strongly sequentially direct transformations of AHL-process nets $W_0 \Rightarrow W_1 \Rightarrow W_3$ via productions ϱ_1 and ϱ_2 depicted in Figure 5.7, and $W_0 \Rightarrow W_2 \Rightarrow W_3$ via ϱ_2 and ϱ_1.

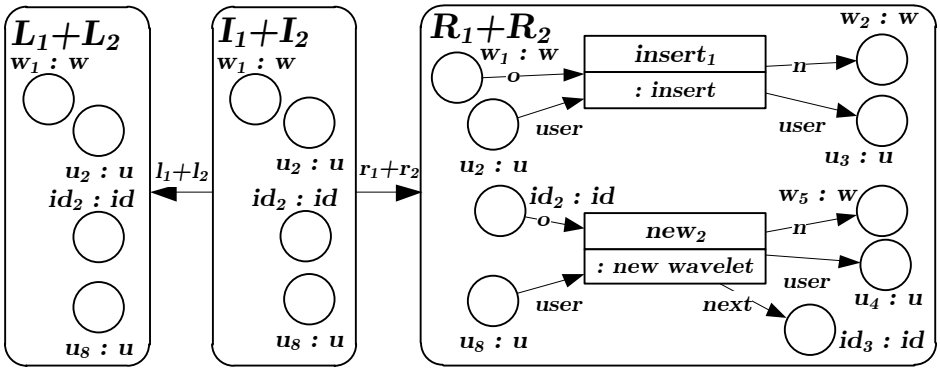


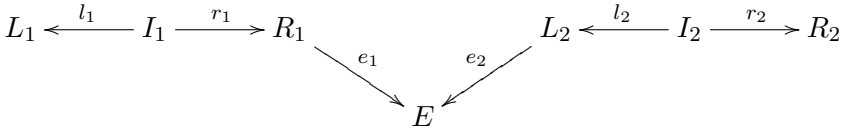
Figure 5.9: Parallel scenario production $\varrho_1 + \varrho_2$

The parallel production $\varrho_1 + \varrho_2$ is shown in Figure 5.9, and there is a direct transformation of AHL-process nets $W_0 \overset{\varrho_1+\varrho_2}{\Longrightarrow} W_3$. ♢

In the following we apply also the Concurrency Theorem to transformations of AHL-process nets. In order to do this, it is necessary to introduce a notion of *strong* E-dependency relations as defined below.

Definition 5.2.11 (Strong E-Dependency Relation of AHL-Process Net Transformations)**.** Given the class $\mathcal{E}'$ of pairs of jointly epimorphic AHL-morphisms, and let ϱ_1 and ϱ_2 be two productions for AHL-process nets with $\varrho_i = L_i \overset{l_i}{\leftarrow} I_i \overset{r_i}{\rightarrow} R_i$ for $i = 1, 2$. An AHL-process net E with AHL-morphisms $e_1 : R_1 \rightarrow E$ and $e_2 : L_2 \rightarrow E$ is a *strong E-dependency relation* for ϱ_1 and ϱ_2 if $(e_1, e_2) \in \mathcal{E}'$, and ϱ_1^{-1} and e_1 as well as ϱ_2 and e_2 satisfy the transformation condition for AHL-process nets.

An E-related transformation, where the E-dependency relation is strong, is called a *strongly E-related transformation.*

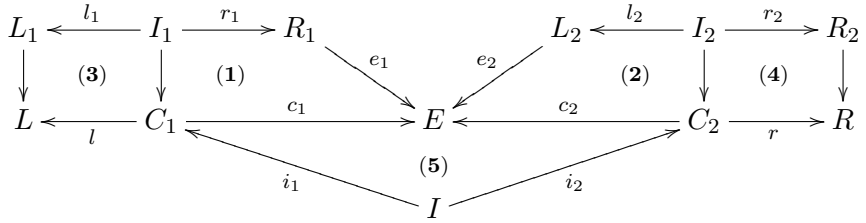


△

Remark 5.2.12 (Strong E-Dependency Relation and E-Concurrent Production).

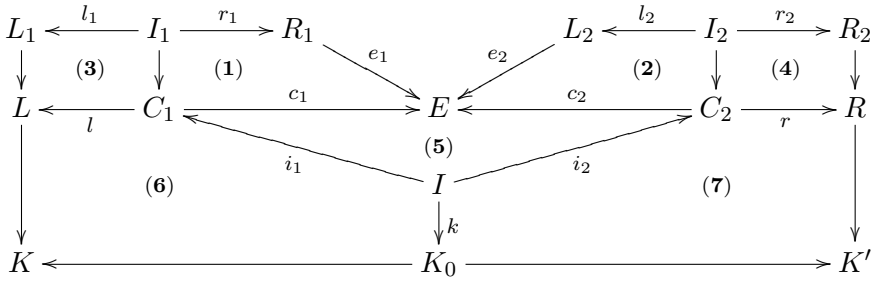
1. Note that a strong E-dependency relation for AHL-process nets is also an E-dependency relation for AHL-nets. The satisfaction of the transformation condition for AHL-process nets by Theorem 4.2.11 that direct transformations of E via ϱ_1^{-1}, e_1 and ϱ_2, e_2 exist, and this means that there are also corresponding pushout complements of r_1 and e_1 as well as l_2 and e_2.

2. Further note that for a strong E-dependency relation (e_1, e_2) the E-concurrent production $\varrho_1 *_E \varrho_2$ (see Definition 5.1.11 again is a production for AHL-process nets. The fact that L and R are AHL-process nets follows from the fact that ϱ_1^{-1}, e_1 and ϱ_2, e_2 satisfy the transformation condition, and therefore, the direct transformations of E with pushout (1)-(4) below lead to AHL-process nets. Moreover, $\mathcal{M}_{AHL}$-morphism $l \circ i_1$ is transition-injective which by Lemma A.7.1 implies that also I is an AHL-process net.



△

Definition 5.2.13 (Strongly E-Concurrent AHL-Process Net Transformations)**.** Given a strong E-dependency relation $(e_1, e_2) \in \mathcal{E}'$ for the productions for AHL-process nets ϱ_1 and ϱ_2, and the E-concurrent production $\varrho_1 *_E \varrho_2$ with pushouts (1)-(4) below. A direct transformation $K \Rightarrow K'$ via $\varrho_1 *_E \varrho_2$ with pushouts (6) and (7) is called *strongly E-concurrent transformation*, if E and K_0 are composable w. r. t. $(I, k, c_1 \circ i_1)$:



△

Theorem 5.2.14 (Concurrency Theorem for AHL-Process Net Transformations)**.** *Let $R_1 \xrightarrow{e_1} E \xleftarrow{e_2} L_2$ be a strong E-dependency relation for the productions for AHL-process nets ϱ_1 and ϱ_2, and $\varrho_1 *_E \varrho_2$ the corresponding E-concurrent production.*

1. (Synthesis) *Given a strongly E-related transformation sequence of AHL-process nets $K \Rightarrow K_1 \Rightarrow K'$ via ϱ_1 and ϱ_2, then there is a* synthesis construction *leading to a strongly E-concurrent direct transformation of AHL-process nets $K \Rightarrow K'$ via $\varrho_1 *_E \varrho_2$.*

2. (Analysis) *Given a strongly E-concurrent direct transformation of AHL-process nets $K \Rightarrow K'$ via $\varrho_1 *_E \varrho_2$, then there is an* analysis construction *leading to a strongly E-related transformation sequence of AHL-process nets $K \Rightarrow K_1 \Rightarrow K'$ via ϱ_1 and ϱ_2.*

3. (Bijective correspondence) *The synthesis and analysis constructions are inverse to each other up to isomorphism.*

Proof-Idea. Since direct transformations of AHL-process nets are also direct transformations of AHL-nets, the constructions of the required direct transformations work similar to the ones for AHL-nets in Theorem 5.1.12. It remains to show that the resulting direct transformations are in fact direct transformations of AHL-process nets, and the result of the synthesis construction is a strongly E-concurrent direct transformation. The fact that the context nets of the direct transformations are AHL-process nets can mainly be derived from the horizontal $\mathcal{M}_{AHL}$-morphisms of the direct transformations which have AHL-process nets as codomains, using Lemma A.7.1. The fact that the synthesis construction provides an E-concurrent direct transformation follows from the fact that composability is a necessary condition for the gluing of AHL-process nets. For the detailed proof we refer to Section B.13 on page 260. □

Fact 5.2.15 (Construction of Strongly E-Related AHL-Process Net Transformations)**.** *For each pair of direct transformations of AHL-process nets* $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2}{\Longrightarrow} K'$ *we have a strong* E*-dependency relation* E *such that* $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2}{\Longrightarrow} K'$ *is strongly* E*-related. Given the comatch* $n_1 : R_1 \to K_1$ *of* $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1$*, the strong* E*-dependency relation is obtained as* $\mathcal{E}'$*-*$\mathcal{M}'_{AHL}$ *pair factorisation (see Fact A.4.4)* $(e_1, e_2) \in \mathcal{E}'$ *and* $h \in \mathcal{M}'_{AHL}$ *of* n_1 *and* m_2*.*

Proof-Idea. Given a sequence of direct transformations of AHL-process nets $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2}{\Longrightarrow} K'$, using Fact A.4.4 we can construct the $\mathcal{E}'$-$\mathcal{M}'_{AHL}$ pair factorisation $(e_1, e_2) \in \mathcal{E}'$ and $h \in \mathcal{M}'_{AHL}$ of n_1 and m_2 with $h \circ e_1 = n_1$ and $h \circ e_2 = m_2$.

Moreover, by subsequent constructions of pullbacks and using the fact that (**AHLNets**, $\mathcal{M}_{AHL}$) with $\mathcal{M}'_{AHL}$ has the $\mathcal{M}$-$\mathcal{M}'$ pushout-pullback decomposition property (Fact A.4.11), we obtain direct transformations of AHL-process nets $E \Rightarrow L$ via ϱ_1^{-1} at match e_1, and $E \Rightarrow R$ via ϱ_2 at match e_2. Then, by Theorem 4.2.11 we obtain that ϱ_1^{-1} and e_1 as well as ϱ_2 and e_2 satisfy the transformation condition of AHL-process nets which by Definition 5.2.11 means that (e_1, e_2) is a strong E-dependency relation such $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2}{\Longrightarrow} K'$ is strongly E-related.

For the detailed proof we refer to Section B.14 on page 261. □

Remark 5.2.16. Note that for the application of the Concurrency Theorem to direct transformations of AHL-process nets it is required to consider *strong* E-dependency relations. However, as we have shown in Fact 5.2.15 this is not a restriction for the applicability of the Concurrency Theorem, since for every sequence of direct transformations of AHL-process nets we obtain a strong E-dependency relation E such that the transformation sequence is strongly E-related.

The restriction to *strongly* E-concurrent transformations, on the other hand, actually restricts the analysis of E-concurrent transformations of AHL-process nets. Nonetheless, for an E-concurrent transformation $K \Rightarrow K'$ via E-related production $\varrho_1 *_E \varrho_2$ that is not strongly E-concurrent, from the proof of item 2 of Theorem 5.2.14 we can derive that for an analysis $K \overset{\varrho_1}{\Rightarrow} K_1 \overset{\varrho_2}{\Rightarrow} K'$ of $K \Rightarrow K'$, we have that K_1 is not an AHL-process net, because composability is a necessary condition of the gluing of AHL-process nets. This means that the restriction to strongly E-concurrent transformations is a restriction to exactly those E-concurrent transformations that can actually be analysed into a proper sequence of AHL-process net transformations. △

Example 5.2.17 (Concurrent Scenario Evolutions)**.** Consider the production ϱ_1 in Figure 5.7 on page 128 and the scenario $w_4 : W_4 \to \mathit{Platform}$ depicted in the top-left of Figure 5.10. We can apply the inverse rule $\varrho_1^{-1} : R_1 \overset{r_1}{\leftarrow} I_1 \overset{l_1}{\to} L_1$ to the AHL-process w_4 and obtain a new AHL-process $w_5 : W_5 \to \mathit{Platform}$, and a subsequent application of ϱ_1 yields again the AHL-process w_4.

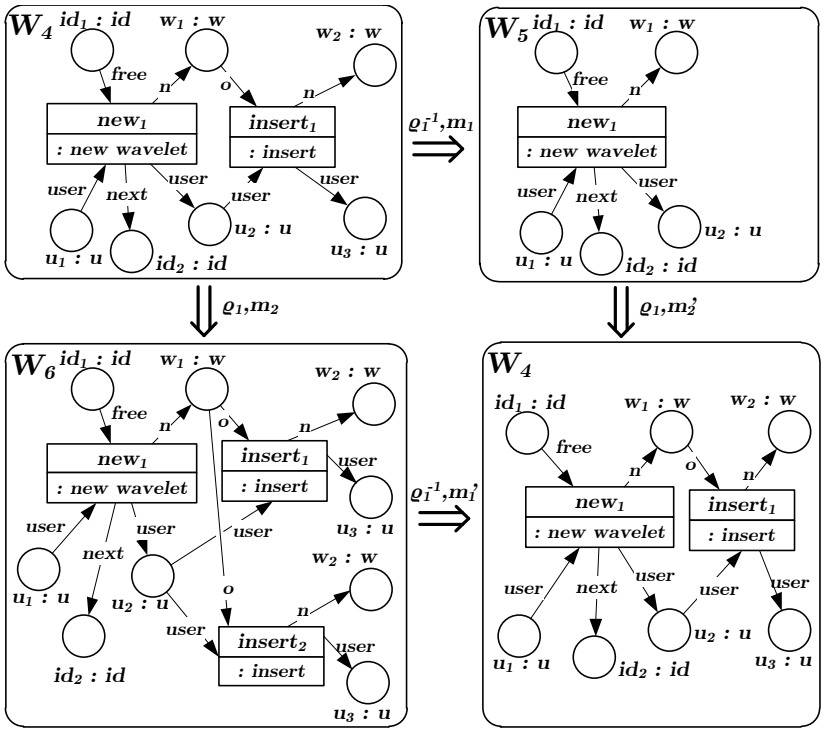


Figure 5.10: Not strongly sequentially independent direct transformations

The sequence of direct transformations of AHL-process nets $W_4 \overset{\varrho_1^{-1},m_1}{\Longrightarrow} W_5 \overset{\varrho_1,m_2}{\Longrightarrow} W_4$ is sequentially independent which can be seen by the fact that we can also apply ϱ_1 with a corresponding match to AHL-process w_4, leading to a typed AHL-net $w_6 : W_6 \to \mathit{Platform}$, and another application of ϱ_1^{-1} to w_6 leads again to the AHL-process w_4. However, the direct transformation sequence $W_4 \overset{\varrho_1^{-1},m_1}{\Longrightarrow} W_5 \overset{\varrho_1,m_2}{\Longrightarrow} W_4$ is not *strongly* sequentially independent which can be seen by the fact that W_6 is not an AHL-process net, because it has forward conflicts at places w_1 and u_2, and thus, typed AHL-net w_6 is not an AHL-process.

Following the construction of strong E-dependency relations and E-concurrent productions, we obtain the production $\varrho_1^{-1} *_E \varrho_1 = (L \overset{l \circ i_1}{\longleftarrow} I \overset{r \circ i_2}{\longrightarrow} R)$ as depicted in Figure 5.11. The resulting production removes an occurrence of an *insert* transition and inserts a new occurrence at the same position.

A direct transformation $W_4 \overset{\varrho_1^{-1} *_E \varrho_1}{\Longrightarrow} W_4$ is shown in the bottom of Figure 5.11. It can be seen that the direct transformation is strongly E-concurrent, because the gluing of W_0 and E via $(I, c_1 \circ i_1, k)$ leads to the AHL-process net W_0, implying that W_0 and E are composable w. r. t. $(I, c_1 \circ i_1, k)$. This means that the direct transformation $W_4 \overset{\varrho_1^{-1} *_E \varrho_1}{\Longrightarrow} W_4$ can again be analysed into the strongly E-related transformation sequence $W_4 \overset{\varrho_1^{-1},m_1}{\Longrightarrow} W_5 \overset{\varrho_1,m_2}{\Longrightarrow} W_4$. ♢

5.3 Concrete Realisations of Abstract Scenarios

In Chapter 4 we presented the modelling of abstract and concrete scenarios using AHL-processes and instantiated AHL-processes, respectively. For a concrete scenario model it is quite simple to obtain an abstraction, omitting the instantiation part of the instantiated AHL-process net. The other way around it is not that simple. In order to obtain a realistic concretisation – a realisation – of an abstract scenario, we cannot just enrich the AHL-process

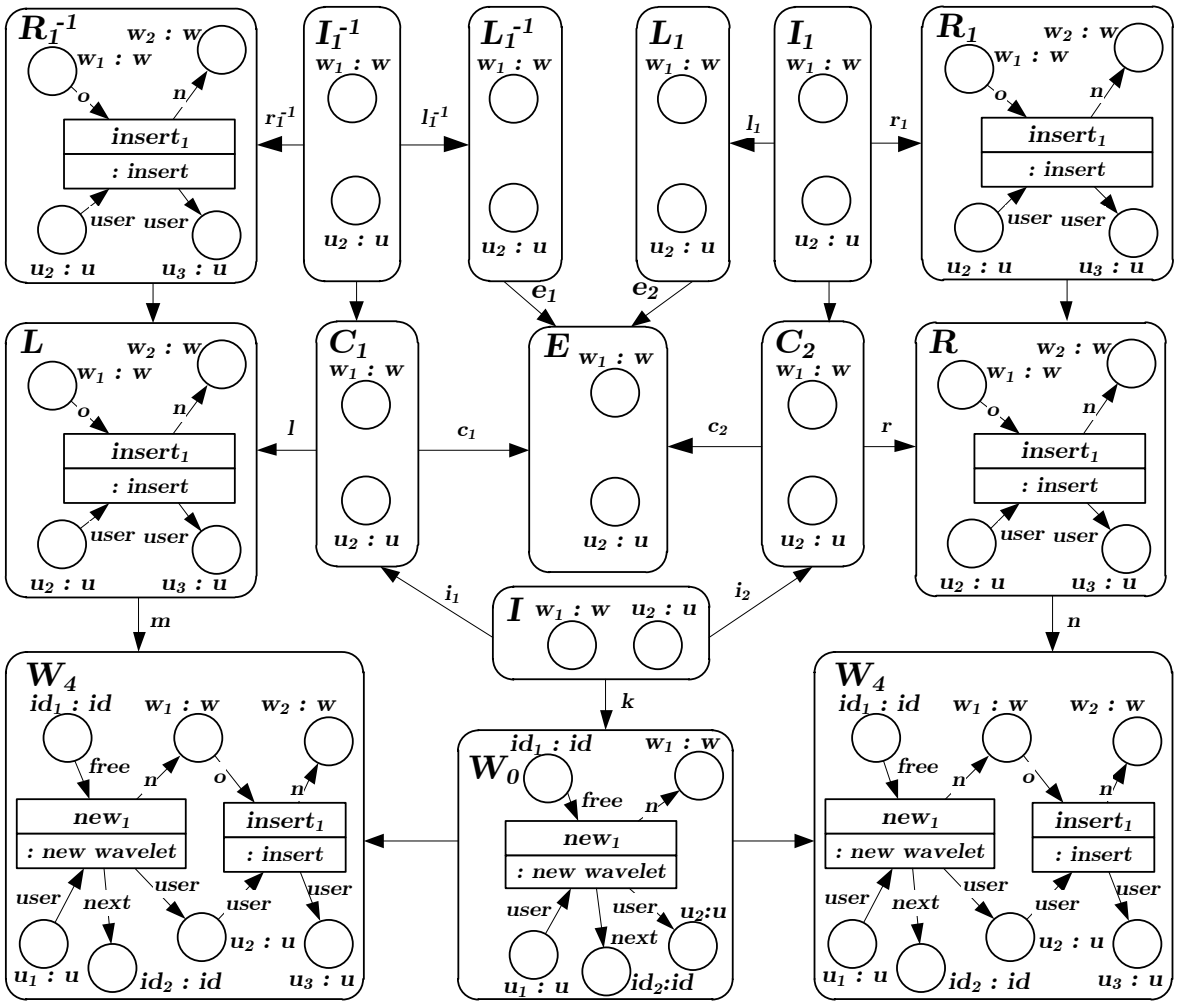


Figure 5.11: Construction of E-concurrent production and strongly E-concurrent transformation

with arbitrary data values and assignments, because it is important that the assignments are consistent. In the following we introduce realisation variables and conditions, that can be used to obtain all realisations of an abstract scenario.

Remark 5.3.1 (Mathematical Constructions).

1. In the following we use a construction $V_\cup$ that turns a (free commutative) sum into a set, i. e. for a sum $S = \sum_{i=1}^{n} s_i$ we have $V_\cup(S) := \{s_1, \ldots, s_n\}$.
2. Note that for a set of variables $X = (X_s)_{s \in S}$ and a signature $S = (S, OP)$, there is a family of inclusions $in : X \hookrightarrow T_\Sigma(X)$. Thus, for each family of functions $f = (f_s : X_{1,s} \to X_{2,s})_{s \in S}$, there is a corresponding assignment $in \circ f : X_1 \to T_\Sigma(X_2)$. Accordingly, we can define a free extension of variable mappings $f : X_1 \to X_2$ to homomorphisms $f^* := (f \circ in)^* : T_\Sigma(X_1) \to T_\Sigma(X_2)$. In the following, we use this extension also for equations, in the way that $f^*(e_l, e_r) := (f^*(e_l), f^*(e_r))$.

△

Definition 5.3.2 (Realisation Variables and Conditions for AHL-Process)**.** Given an AHL-process net K with data type part (Σ, A) with $\Sigma = (S, OP; X)$. We define the family of

realisation variables $(\hat{X}_s)_{s\in S}$ by

$$\hat{X}_s = \biguplus_{t\in T_K} Var(t)_s \uplus P_{K,s} \quad (s \in S)$$

with coproduct injections $d_{t,s} : Var(t)_s \to \biguplus_{t\in T} Var(t)_s$, and sorted sets of places $P_{K,s} = \{p \in P_K \mid type_K(p) = s\}$.

Further, we define the set of *realisation conditions* $Real_K$ by

$$Real_K = Cond_K \cup Pre_K \cup Post_K \subseteq \mathcal{P}_{fin}(Eqns(\Sigma; T_\Sigma(\hat{X})))$$

where the sets of equations $Cond_K$, Pre_K and $Post_K$ are defined by

$$Cond_K = \bigcup_{t\in T_K} \mathcal{P}_{fin}(d_t^*)(cond_K(t)),$$

$$Pre_K = \bigcup_{t\in T_K} V_\cup((d_t^* \otimes id_{P_K})^\oplus(pre_K(t))), \text{ and}$$

$$Post_K = \bigcup_{t\in T_K} V_\cup((d_t^* \otimes id_{P_K})^\oplus(post_K(t))).$$

An assignment $\hat{v} : \hat{X} \to A$ of the realisation variables is called *solution for the realisation conditions*, if $(A, \hat{v}) \vDash Real_K$. △

The following theorem states that for each AHL-process net K, we can compute the set of all its concrete instantiations as solutions of a term equation system.

Theorem 5.3.3 (Concrete Realisation of AHL-Process)**.** *Given an AHL-process net K with data type part (Σ, A), then we have the following:*

1. (Synthesis) *For each solution $\hat{v} : \hat{X} \to A$ for the realisation conditions $Real_K$ there exists a concrete instantiation $(inst, K)$.*
2. (Analysis) *For each concrete instantiation $(inst, K)$ there exists a solution $\hat{v} : \hat{X} \to A$ for the realisation conditions $Real_K$.*
3. (Bijective correspondence) *Given the sets $Inst(K)$ of all concrete instantiations of K and $Sol(K)$ of all solutions of the realisation conditions of K, the construction of solutions from instantiations and vice versa establish a bijective correspondence $Inst(K) \cong Sol(K)$.*

Proof-Idea.

1. *(Instantiation)* Given a solution for the realisation conditions $\hat{v} : \hat{X} \to A$, we can define a P/T-morphism $inst : Skel(K) \to Flat(K)$ by
   - $inst_P(p) = (\hat{v}(p), p)$, and
   - $inst_T(t) = (t, \hat{v} \circ d_t)$.

   It can be shown that $(inst, K)$ is a concrete instantiation.
2. *(Solution)* Given a concrete instantiation $(inst, K)$, an assignment $\hat{v} : \hat{X} \to A$ can be defined as follows:
   - For $p \in P_{K,s}$ with $inst_P(p) = (a, p)$ let $\hat{v}(p) = a$, and

- for $\hat{x} \in \biguplus_{t \in T_K} Var(t)_s$ with $t \in T_K$ and $x \in Var(t)_s$ such that $d_{t,s}(x) = \hat{x}$ and $inst_T(t) = (t, v)$, let $\hat{v}(\hat{x}) = v(x)$.

It can be shown that $\hat{v}$ is a solution for the realisation conditions.

3. *(Bijective correspondence)* The constructions above are inverse to each other, establishing a bijective correspondence $Inst(K) \cong Sol(K)$.

For the detailed proof we refer to Section B.15 on page 262. □

Example 5.3.4 (Concrete Realisation of Abstract Scenario)**.** Consider the abstract scenario $wave : Wave \to Platform$ in Figure 5.12 that we already presented in Example 4.1.8, where the corresponding platform is modelled by the AHL-net *Platform* in Figure 3.2 on page 35. The scenario consists of 6 transitions new_1, new_2, $insert_1$,$insert_2$, $invite_1$ and $remove_1$, and for each of these transitions there is a family of injections $d_t = (d_{t,s} : Var(t)_s \in \uplus_{t \in T} Var(t)_s)_{s \in S}$. For instance, for the transition new_1, we have

$$\begin{aligned} Var(new_1)_{nat} &= \{free, next\}, \\ Var(new_1)_{user} &= \{user\}, \\ Var(new_1)_{wavelet} &= \{n\} \end{aligned}$$

and empty functions for the sorts *bool*, *range* and *text* (see Table 3.1 on page 36 for the signature Σ-$Wave$). So, for example the injection $d_{new_1,nat}$ maps variables $d_{new_1,nat}(free) = (free, new_1)$ and $d_{new_1,nat}(next) = (next, new_1)$.

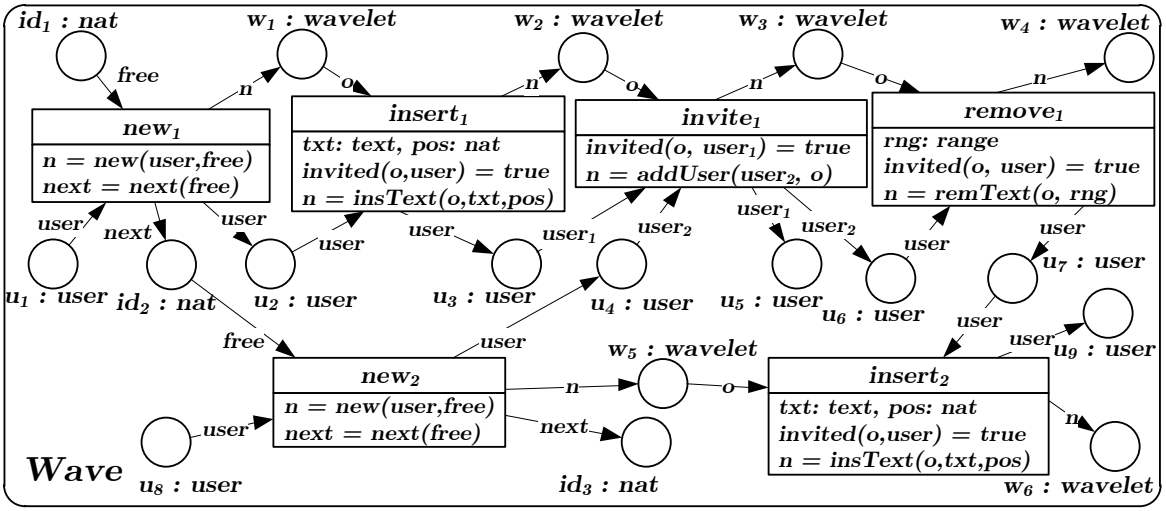


Figure 5.12: AHL-process model of an abstract scenario

The family $\hat{X} = (\hat{X}_s)_{s \in S}$ of realisation variables consists of all images of variables occurring in the environment of a transition w. r. t. to d_t, and all places, sorted by their types. Note that the set of places is disjoint from the set of renamed transition variables which means that it is not necessary to rename the places to obtain a disjoint union of the transition variables

and places:

$$\begin{aligned}
\hat{X}_{nat} &= \{(free, new_1), (next, new_1), (free, new_2), (next, new_2),\\
&\quad (pos, insert_1), (pos, insert_2), id_1, id_2, id_3\}\\
\hat{X}_{user} &= \{(user, new_1), (user, new_2), (user, insert_1), (user, insert_2),\\
&\quad (user_1, invite_1), (user_2, invite_1), (user, remove_1), u_1, \ldots, u_9\}\\
\hat{X}_{wavelet} &= \{(n, new_1), (n, new_2), (o, insert_1), (n, insert_1), (o, insert_2), (n, insert_2),\\
&\quad (o, invite_1), (n, invite_1), (o, remove_1), (n, remove_1), w_1, \ldots, w_6\}\\
\hat{X}_{text} &= \{(txt, insert_1), (txt, insert_2)\}\\
\hat{X}_{range} &= \{(rng, remove_1)\}
\end{aligned}$$

The set of realisation conditions $Real_{Wave}$ is given by the union of condition sets $Cond_{Wave}$, Pre_{Wave} and $Post_{Wave}$ with

$$\begin{aligned}
Cond_{Wave} = \{&(n, new_1) = new((user, new_1), (free, new_1)),\\
&(next, new_1) = next((free, new_1)),\\
&(n, new_2) = new((user, new_2), (free, new_2)),\\
&(next, new_2) = next((free, new_2)),\\
&invited((o, insert_1), (user, insert_2)) = true,\\
&(n, insert_1) = insText((o, insert_1), (txt, insert_1), (pos, insert_1)),\\
&invited((o, insert_2), (user, insert_2)) = true,\\
&(n, insert_2) = insText((o, insert_2), (txt, insert_2), (pos, insert_2)),\\
&invited((o, invite_1), (user_1, invite_1)) = true,\\
&(n, invite_1) = addUser((user_2, invite_1), (o, invite_1)),\\
&invited((o, remove_1), (user, remove_1)) = true,\\
&(n, remove_1) = remText((o, remove_1), (rng, remove_1))\}\\
Pre_{Wave} = \{&(free, new_1) = id_1, (user, new_1) = u_1,\\
&(free, new_2) = id_2, (user, new_2) = u_8,\\
&(o, insert_1) = w_1, (user, insert_1) = u_2,\\
&(o, insert_2) = w_5, (user, insert_2) = u_7,\\
&(o, invite_1) = w_2, (user_1, invite_1) = u_3, (user_2, invite_1) = u_4,\\
&(o, remove_1) = w_3, (user, remove_1) = u_6\}\\
Post_{Wave} = \{&(next, new_1) = id_2, (user, new_1) = u_2, (n, new_1) = w_1,\\
&(next, new_2) = id_3, (user, new_2) = u_4, (n, new_2) = w_5,\\
&(n, insert_1) = w_2, (user, insert_1) = u_3,\\
&(n, insert_2) = w_6, (user, insert_2) = u_9,\\
&(n, invite_1) = w_3, (user_1, invite_1) = u_5, (user_2, invite_1) = u_6,\\
&(n, remove_1) = w_4, (user, remove_1) = u_7\}
\end{aligned}$$

From Theorem 5.3.3 we know that for every assignment $\hat{v} : \hat{X} \to A$ that satisfies all equations in $Real_{Wave}$, called solution for the realisation conditions, there is a corresponding instantiation of the AHL-process net *Wave* and vice versa. An example of a solution for the realisation conditions that corresponds to the instantiation $Inst = (inst, Wave)$ in Figure 5.13 is given in the following:

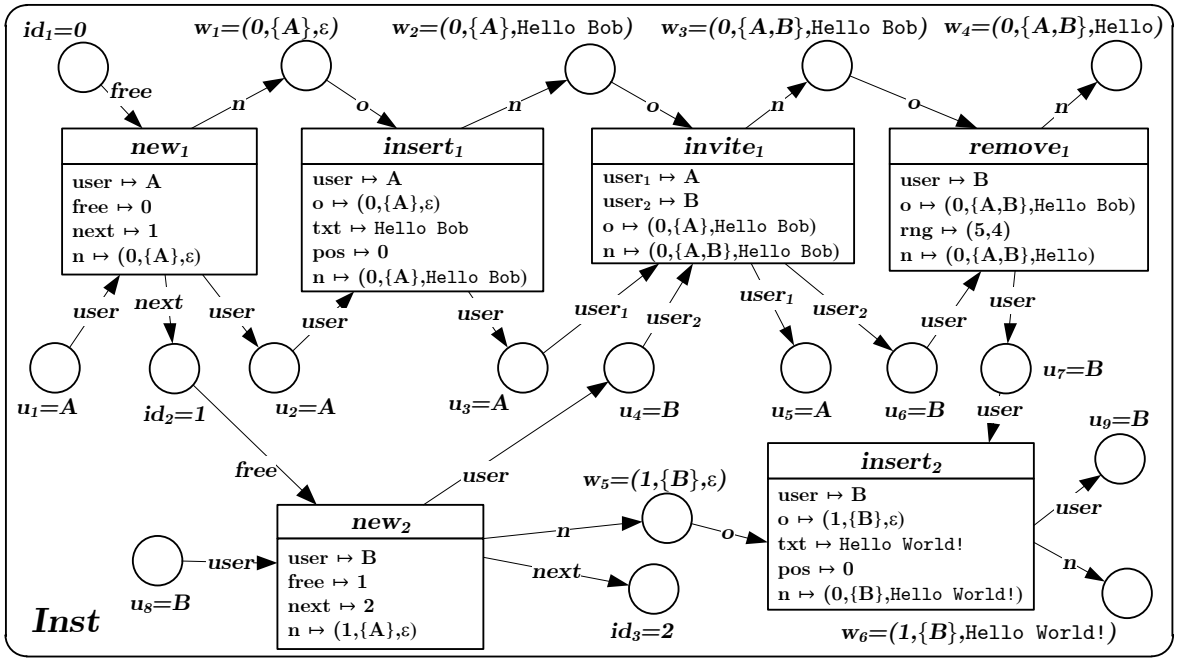


Figure 5.13: Realisation of the AHL-process net *Wave*

$$\hat{v}_{nat} : \hat{X}_{nat} \to A_{nat}$$
$$(free, new_1) \mapsto 0, (next, new_1) \mapsto 1, (free, new_2) \mapsto 1, (next, new_2) \mapsto 2,$$
$$(pos, insert_1) \mapsto 0, (pos, insert_2) \mapsto 0, id_1 \mapsto 0, id_2 \mapsto 1, id_3 \mapsto 2$$

$$\hat{v}_{user} : \hat{X}_{user} \to A_{user}$$
$$(user, new_1) \mapsto A, (user, new_2) \mapsto B, (user, insert_1) \mapsto A, (user, insert_2) \mapsto B,$$
$$(user_1, invite_1) \mapsto A, (user_2, invite_1) \mapsto B, (user, remove_1) \mapsto B$$
$$u_1 \mapsto A, u_2 \mapsto A, u_3 \mapsto A, u_4 \mapsto B, u_5 \mapsto A, u_6 \mapsto B, u_7 \mapsto B, u_8 \mapsto B, u_9 \mapsto B$$

$$\hat{v}_{wavelet} : \hat{X}_{wavelet} \to A_{wavelet}$$
$$(n, new_1) \mapsto (0, \{A\}, \epsilon), (n, new_2) \mapsto (1, \{B\}, \epsilon),$$
$$(o, insert_1) \mapsto (0, \{A\}, \epsilon), (n, insert_1) \mapsto (0, \{A\}, \texttt{Hello Bob}),$$
$$(o, insert_2) \mapsto (1, \{B\}, \epsilon), (n, insert_2) \mapsto (1, \{B\}, \texttt{Hello World!}),$$
$$(o, invite_1) \mapsto (0, \{A\}, \texttt{Hello Bob}), (n, invite_1) \mapsto (0, \{A, B\}, \texttt{Hello Bob}),$$
$$(o, remove_1) \mapsto (0, \{A, B\}, \texttt{Hello Bob}), (n, remove_1) \mapsto (0, \{A, B\}, \texttt{Hello}),$$
$$w_1 \mapsto (0, \{A\}, \epsilon), w_2 \mapsto (0, \{A\}, \texttt{Hello Bob}),$$
$$w_3 \mapsto (0, \{A, B\}, \texttt{Hello Bob}), w_4 \mapsto (0, \{A, B\}, \texttt{Hello}),$$
$$w_5 \mapsto (1, \{B\}, \epsilon), w_6 \mapsto (1, \{B\}, \texttt{Hello Bob})$$

$$\hat{v}_{text} : \hat{X}_{text} \to A_{text}$$
$$(txt, insert_1) \mapsto \texttt{Hello Bob}, (txt, insert_2) \mapsto \texttt{Hello World!}$$

$$\hat{v}_{range} : \hat{X}_{range} \to A_{range}$$
$$(rng, remove_1) \mapsto (5, 4)$$

For each assignment of a variable (x, t) with transition t and $x \in Var(t)$, in the consistent

transition assignment (t, v) in *Inst*, we have $v(x) = \hat{v}(x, t)$. Moreover, for each variable in $\hat{X}$ that corresponds to a place p in *Wave*, we have a coloured place $(\hat{v}(p), p)$ in *Inst*.

Note that the construction of the term equation system and the construction of instantiations for found solutions can be done with our tool, as described in Chapter 6. ♢

6 Tool Support

In this chapter we present the implementation of a tool support for our framework for the modelling and analysis of communication platforms using algebraic high-level nets and processes. First, in Section 6.1 we consider the requirements for the tool support. In Section 6.2 we review a number of tools that are already available for the modelling of high-level Petri nets. In Section 6.3 we present the current development status of our own tool.

6.1 Requirements for a Tool Support for the Modelling and Analysis of Communication Platforms and Scenarios

Based on the requirements for the modelling of communication platforms described in Section 2.3 and the analysis problems described in Section 2.4, we have the following requirements for a tool support, considering the corresponding realisations and results of this thesis:

Modelling of Communication Platforms As pointed out in Concept 3.1.8, communication platforms can be modelled as algebraic high-level (AHL-)nets. An AHL-net is a Petri net with a high-level data type part. Since Petri nets are graph-like structures, a suitable tool-support should be a visual editor for "drawing" the graphical components like places, transitions, and the arcs between them. Moreover, there have to be means to formally describe the data type part of the net, and to annotate the structures to corresponding visual elements in the net, i. e. to define the types of places, the firing conditions of transitions and the inscriptions on arcs.

Modelling of Scenarios According to Concept 4.1.7, abstract scenarios can be modelled using AHL-processes, and according to Concept 4.3.6, concrete scenarios can be modelled using instantiated AHL-processes. Since AHL-process nets are a special kind of AHL-nets, for the modelling of these nets it may suffice to enable the user to check whether an AHL-net satisfies the requirements of being an AHL-process net in Definition 4.1.1. For the modelling of instantiations, the user has to have the ability to annotate each place of a net with a data value of the corresponding type, and each transition with a (consistent) assignment of all variables in the environment of the transition.

Evolution of Communication Platforms and Scenarios For the structural evolution of communication platforms (Concept 3.2.16) and of scenarios (Concept 4.2.13 and Concept 4.6.9), we need an implementation of the rule-based transformation of AHL-nets. For the transformation of scenarios it is additionally required that the transformation condition for AHL-process nets in Definition 4.2.10 can be checked, and for the transformation of instantiations using abstract productions, it is required that the instantiation condition in Definition 4.6.7 can be checked.

Analysis of Communication Platforms and Scenarios For the Local Church-Rosser Theorems of AHL-nets and AHL-processes, the tool has to be able to check for two direct transformations of the same net whether these transformations are (strongly) parallel independent, and for a sequence of two subsequent direct transformation it has to be possible to check whether these transformations are (strongly) sequentially independent.

Moreover, for the Parallelism Theorems of AHL-nets and AHL-processes, the tool has to provide means for the synthesis and analysis of parallel productions, whereas for the Concurrency Theorems, the tool has to enable the synthesis and analysis of (strongly) E-related transformations and the construction of (strong) E-dependency relations.

Finally, regarding the concrete realisations of abstract scenarios in Section 5.3, there has to be a way to compute the realisation variables and conditions of an AHL-process net, and to check whether there are solutions of these conditions. If for an AHL-process net there are solutions for the realisation conditions, the tool should be able to construct the corresponding instantiation.

6.2 Available Tool Support for the Modelling and Analysis of High-Level Petri Nets

There are already several tools for the modelling and analysis of Petri nets and other graph-structures. In the following we review some of these tools that are possible candidates for our modelling approach.

CPN Tools The CPN Tools[24] is a visual editor for the modelling of Coloured Petri nets (CPNs) [Jen91, JR91]. CPNs are a high-level Petri net variant that uses data types in the programming language CPN ML which is an extension of the well-known functional programming language Standard ML. The CPN Tools offer a variety of techniques for the simulation, verification and analysis of CPNs [Jen97b].

The functional programming paradigm fits in with the concept of data types based on signatures and algebras [EM85], however, the main difference between algebraic high-level and coloured Petri nets is the fact that CPNs do not allow to specify firing conditions for transitions, but instead, the data type part is only used for arc inscriptions.

In Theorem 4.3 in [PER95] it is shown that each AHL-net AN also defines a corresponding CPN CN. However, the resulting CPN CN corresponds merely to the flattening $Flat(AN)$ of the AHL-net AN. Thus, on the transfer from the AHL-net to the CPN modelling technique, we completely lose the high-level structure of the AHL-net which is vital for the modelling of communication platforms. Therefore, the CPN Tools are not a suitable tool support for the modelling and analysis of AHL-nets in the application domain of communication platforms.

RONEditor The RONEditor[25] is a visual editor for the modelling of reconfigurable object nets (RONs) and is described in [BEHM07]. RONs are based on the idea of algebraic higher-order (AHO-)nets that are algebraic high-level nets, where the tokens are (low-level) Petri nets (called object nets) and productions for the rule-based transformation of the nets (called rules). Instead of firing conditions for transitions, the RON-approach offers four kinds of transitions with fixed semantics, like the firing of an object net or

[24] CPN Tools – http://cpntools.org/
[25] RONEditor – http://www.user.tu-berlin.de/o.runge/tfs/projekte/roneditor/

the application of a rule on an object net. For simulation of the behaviour of RONs, the RONEditor converts the net models to attributed typed graphs [EEPT06b] and uses the AGG graph transformation engine (see below).

AHLI-Net Editor and CPEditor The RONEditor described above evolved into the MuvitorKit [MBE09], an abstract framework for the rapid implementation of visual editors. Based on the MuvitorKit, the Eclipse Modeling Framework (EMF)[26] and the Graphical Editing Framework (GEF)[27], an editor for AHL-nets with individual tokens (AHLI-nets, see Remark 3.1.11) was implemented in the diploma thesis [Fis10]. The AHLI-net editor consists of an editor for the definition of signatures that is based on the Eclipse Xtext framework, and corresponding algebras for signatures can be implemented as Java classes. Moreover, the editor allows the visual modelling of AHLI nets that use the previously specified data type part, and also the simulation of the firing behaviour using the AGG graph transformation engine on attributed typed graphs constructed from the AHLI-nets.

In the diploma thesis [Pas12] the AHLI-net editor has been extended to support also the rule-based transformation of AHLI-nets. The editor supports also nested application conditions [HP05] and multi-amalgamated rules [BFH85, GEH10, Gol11]. Similar to the simulation of the firing behaviour, the transformation of AHLI-nets is based on the conversion to graphs and graph-transformation rules and using the AGG graph transformation engine.

Based on the implementations of the AHLI-net editor, in [Mod12] the CPEditor, a visual tool for the design, simulation and validation of communication platforms was developed. The editor supports the usage of signatures in the same format as in [Fis10], and correspondingly algebras can defined as Java classes that have to be registered in the plugin information of the CPEditor. Apart from the possibility to design and transform AHLI-net models, the CPEditor provides verification techniques to investigate the independence of two single firing steps, transformation rules or a combination of a firing step and a transformation rule (see [Mod12]). Analogously to the simulation of the firing behaviour and the transformation steps, the verification of independence is based on a conversion to typed attributed graphs, using the API functions of the AGG tool for independence of graph transformations.

AGG Graph Transformation Engine The Attributed Graph Grammar (AGG)[28] system is a development environment for transformation systems of (typed) attributed graphs [Tae04, EEPT06b]. The tool consists of a graphical user interface for the modelling of graphs and graph transformation rules. Moreover, it supports a variety of analysis techniques for graph transformation systems presented in [EEPT06b], like the analysis of critical pairs, graph parsing or the investigation of termination criteria for layered graph transformation systems.

The graph transformation engine is the internal model of the AGG system, and it can be used without the graphical user interface by a Java API. The AGG API is used in the AHL-net editor and the CPEditor for the simulation of firing behaviour and rule-based reconfiguration, as mentioned above. The formal basis for the usage of the AGG graph transformation engine is a result in [MEE10], where a functor $\mathcal{F} : \mathbf{PTINets}|_{\mathcal{M}_{inj}} \to \mathbf{AGraphs}_{PNTG}$ from P/T nets with individual tokens (PTI nets) restricted to injective

[26] EMF project – http://www.eclipse.org/emf
[27] GEF project – http://www.eclipse.org/gef
[28] Attributed Graph Grammar System – http://user.cs.tu-berlin.de/ gragra/agg/

morphisms to attributed graphs typed over a specific type graph is investigated. It is shown that the functor $\mathcal{F}$ is an $\mathcal{M}$-functor. This implies that the functor preserves and creates productions, matches and direct transformations, i. e. they can be transferred along the functor in each direction. As pointed out in [MEE10, MEE12], the functor $\mathcal{F}$ cannot be extended to a functor that maps also non-injective morphisms, and up to now it is unknown whether there is a suitable $\mathcal{M}$-functor for the conversion between Petri net and typed attributed graph transformation systems with arbitrary morphisms. As a result, the RONEditor, AHLI-net editor and CPEditor only support rule-based transformation with injective matching.

6.3 Algebraic High-Level Net and Process Editor APE

In order to support the modelling and analysis of communication platforms and scenarios using the integrated framework presented in this thesis, we implement a tool called **A**lgebraic High-Level Net and **P**rocess **E**ditor (APE). In the following we describe the architecture of the tool and its currently implemented functions. Subsection 6.3.5 gives an overview over the current status of the implementation that is still in progress.

The tool is not based on one of the available tools presented in Section 6.2. As already mentioned, the CPN Tools are not suitable for the editing of algebraic high-level nets, since the conversion of AHL-nets into coloured Petri nets destroys the high-level nature of the nets. The AHLI-net editor and the CPEditor are visual editors for the modelling of a kind of AHL-nets, but there are two main reasons why we do not build our tool as an extension of these tools.

The first reason is the fact that the tools do not support non-injective morphisms, due to the fact that they rely on the conversion of nets into attributed typed graphs. Considering the importance of AHL-processes in this thesis which are AHL-morphisms that are in general non-injective, this is an unacceptable restriction for our framework for the modelling of communication platforms and scenarios.

The second reason is the fact that in the AHLI-net editor and CPEditor the semantics of algebras is implemented in the programming language Java. The way the Java expressions are used as firing conditions in these tools follows the paradigm of imperative programming, effectively imposing a strict order of computation for the single expressions. Considering the firing behaviour of AHL-nets, this is not a problem as long as it is ensured that the pre conditions of transitions are inscribed with all necessary input parameters, which is a restriction for the modelling in the aforementioned tools. In this thesis, however, we consider not only the firing semantics but also the process semantics of AHL-nets, and we consider instantiations of AHL-processes. In Section 5.3 we have shown that all instantiations of an AHL-process net can be computed as solutions for a system of term equations. The strength of this result relies on the fact that the computation does not depend on the causal order of places or transitions inside the net, allowing to use constraint solving strategies for the solution of the equation system. Therefore, we use the logic programming language Prolog to implement the semantics of the data type part of the AHL-nets, as described in full detail in Subsection 6.3.2.

6.3.1 Tool Architecture

The APE tool mainly consists of two parts: A visual editor that is being implemented in the object-oriented programming language Java[29], and a back-end for the modelling and

[29] Oracle Java – http://www.java.com/

computations of the data type part in the logic programming language Prolog. As interpreter for the Prolog language we use the open-source implementation SWI-Prolog[30] which is widely used in reseach, education and commercial applications. The communication between the two parts is based on JPL[31], a bidirectional interface between Java and SWI-Prolog.

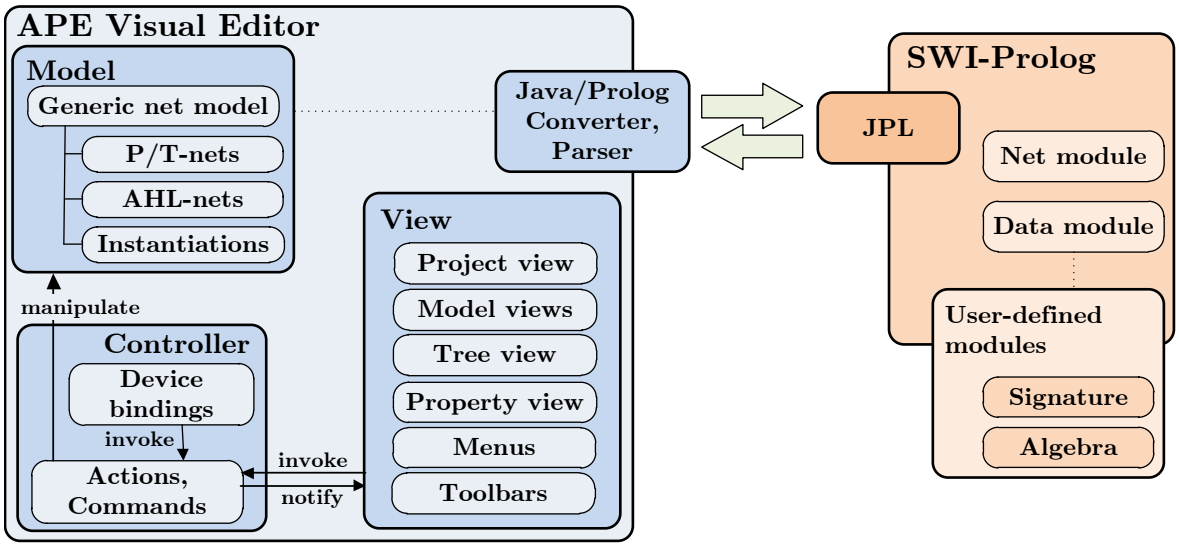


Figure 6.1: Architecture of the APE tool

The implementation follows the Model-View-Controller (MVC) pattern, where the Controller is mediating between the Model and the View, as shown in Figure 6.1.

Model The Model part of the editor consists exactly only of classes for the models such as AHL-nets or instantiations as described in the previous sections of this thesis. Supported are classes for low-level P/T-nets, algebraic high-level nets and instantiations of AHL-nets. The fact that all these Petri net classes basically follow the same concept of "Petri nets are monoids" [MM90] allows the definition of an abstract generic class `Net` that is the superclass of all concrete Petri net classes, and all manipulations of nets like the creation or deletion of net elements such as places, transitions and arcs is already abstractly implemented in the class `Net`.

The abstract model is basically a bipartite directed simple graph (a graph with at most one edge between a source and a target node) between sets of places and transitions. Moreover, each edge of the simple graph, called `ArcCollection` consists of a bundle of single arcs, called `ArcElements` with the same source and target. This definition is owed to the dichotomy between mathematical definition and visual representation of Petri nets: While the entirety of all `ArcElements` with the same transition as target or source node correspond to the monoid sum that is the pre or post domain, respectively, of that transition, each `ArcCollection` corresponds to an arrow in the visual representation of the net. Figure 6.2 shows an example of an AHL-net, where we have $pre(t) = (a,p) \oplus (b,p)$ and $post(t) = (x,p) \oplus (y,p)$. In the graphical representation that is used throughout this thesis and other literature, depicted in the left of Figure 6.2 for our example, there is a single arrow from p to t and from t to p, respectively. Each of these arrows corresponds to an `ArcCollection`. Alternatively, it is possible to draw

[30] SWI-Prolog – http://www.swi-prolog.org
[31] JPL interface – http://www.swi-prolog.org/packages/jpl

an arrow for each element in the pre respectively post domain of t, like depicted in the right of Figure 6.2, where each of the arrows corresponds to an `ArcElement`. Therefore, since it is possible to manipulate both, the `ArcCollections` and the `ArcElements` with implications to the respective other component, the `Net` class provides a good support to work with the nets from the mathematical as well as from a – for the user more convenient – visual perspective.

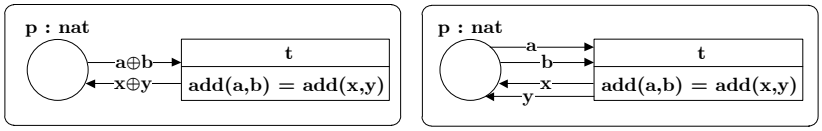


Figure 6.2: `ArcElements` and `ArcCollections`

The implementation of the concrete net classes as subclasses of the abstract `Net` class resembles the idea of parametrised net classes in [Pad96, PE01]. For subclasses of the generic net model it suffices to define the additional data of places, transitions and arc elements. In the case of P/T-nets, the additional data is for all three element types empty. In the case of AHL-nets, each place consists of a type, each transition consists of a set of equations, and each arc element consists of a term. Moreover, an instantiation is an extension of an AHL-net, where additionally each place is equipped with a data value, and each transition has a data assignment for all variables in its environment. For a better usability of the tool, it allows instantiations to be partial, i. e. it is possible to omit data values of places and the transition assignments can be partial functions, but it also offers the possibility to check if an instantiation is complete and consistent.

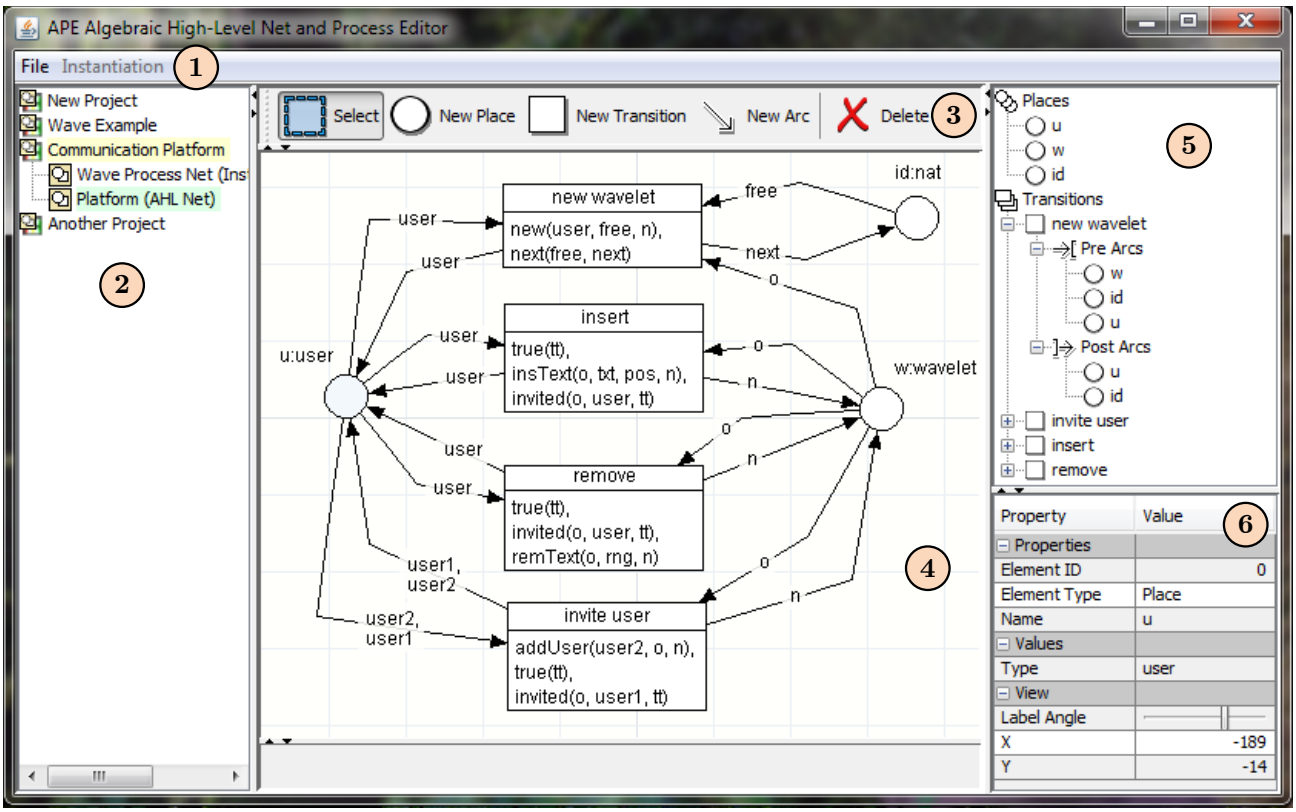


Figure 6.3: User interface of the APE tool

View The view of the tool defines the visible user interface, as shown in Figure 6.3. The view contains a menu (1) in the top, that is used for instance to create new projects or models, and to save or load them. Moreover, it consists of a project view (2) in the left, that shows all loaded models organised into projects. At the top-center of the view is a toolbar (3) that has buttons for the selection and manipulation of the models. The model view (4) in the center shows the currently selected model that can be edited by the user. Behind the model view there is a separate view model for each of the (domain) models that contain all the visual informations for those models, like the positions of elements. These informations are strictly separated from the domain model, because they are only present for the convenience of the user, but they do not belong to the underlying mathematical model. The tree view (5) at the top-right organises all elements of the currently edited model in a tree. This is to help finding elements in larger models, and gives an overview over the elements and structure that is independent from the graphical representation in the model view. Finally, the property view (6) at the bottom-right shows the properties of the currently selected model, project or model element. It can also be used to change properties, if they are editable, like the name of an element or its position.

Controller The controller is responsible to handle all user input of the tool. This can be for instance a mouse click on one of the visual components of the visual editor or a device binding like pressing a key on the keyboard or using the mouse wheel. Interactions with visual components invoke actions, while device bindings invoke commands. The main difference between actions and commands is that an action is always active in the same context, while a command can be registered for different device bindings. Actions and commands can manipulate the models and/or notify the view of changes that have to be made in response to the user input, like updating the visual representation of a changed model, or change the view because of a new selection.

6.3.2 Signatures and Algebras in Prolog

The idea of using Prolog for the implementation of tools for high-level Petri nets is not new. In the late 1980s a compiler and simulator for Open Petri Nets (OPNs) was implemented using the Prolog solving engine [Dib88]. The aim of the work was the analysis of the functional behaviour of telecommunication systems. In order to avoid the development of a graphical user interface for the editor that was only text-based, the developers changed the used Petri net class towards Hierarchical Coloured Petri nets [Jen97a], allowing them to use the CPN Tools editor, as pointed out in [CDH99]. As already mentioned above, our graphical user interface is implemented in the object-oriented programming language Java, and we use Prolog only as a text-based back-end for the data types.

Prolog is a logic programming language with only a single data type – the *term* that can be of one of the following forms:

Atom An atom is a single word that starts with a small letter, like `x`, or an arbitrary text string in single quotes, like `'an Atom'`. An atom is an entity without any inherent meaning.

Numbers A number can be an integer, like `1`, or a floating point number, like `1.0`.

Variables A variable is a single word starting with a capital letter or an underscore that contains only letters, digits and underscores, like `Var_1`. Variables have the same meaning as variables in mathematics, i. e. they serve as place-holders for arbitrary terms.

Compound Terms A compound term consists of an atom called *functor*, and a number of terms that are the *arguments* of the functor, like `functor(arg1,Arg2)`. The number of the arguments is called *arity* of the term. A special case of compound terms are lists that are denoted with square brackets, like `[]` for the empty list, and elements of the list are separated by commas, like in `[1,2,3]`.

The functors represent predicates or relations and their meaning is specified by *facts* or *rules*. A fact has the form

```
functor(Arguments).
```

and means that the the property represented by the functor is true for the arguments. Rules are of the form

```
functor(Arguments) :- Body.
```

where the `:-` resembles an implication arrow to the left (←), and the property represented by the functor is true for the arguments if the body of the rule is true. The body can consist of terms, concatenated by commas, denoting conjunction, or by semicolons, denoting disjunction.

We implemented a data module (see Figure 6.1) that allows to define signatures and algebras in Prolog. The signature Σ-*Wave* in Table 3.1 can be specified as follows:

Listing 1: Definition of signature in Prolog (excerpt)

```
% Sorts:
:- type(bool).
:- type(nat).
:- type(range).

% Operations:
:- op(true, [bool]).
:- op(false, [bool]).
:- op(start, [range,nat]).
:- op(toRange, [nat,nat,range]).
:- op(next, [nat,nat]).
```

Note the special use of the rule syntax without a functor left to the `:-`. This kind of rule does not depend on a functor, and the body of the rule is automatically evaluated when the corresponding module is loaded by the interpreter. A new sort can be defined using the functor `type`, resulting in a new functor with the name of the defined type and an arity of 1. This functor can then be used for the definition of the corresponding carrier sets in an algebra, as shown in the code listing below. For instance, the functor `bool` is defined to be true for elements within the finite domain $\{0, 1\}$, representing the Boolean values false and true. The functor `nat` is defined to be true for natural numbers within the finite domain $\{0, ..., 16\}$. This is not exactly the same as in our algebra A in Table 3.2, where the carrier set A_{nat} contains all natural numbers. However, it is advised to restrict the number of elements in a carrier set, if possible, in order to increase the performance of calculations.

Moreover, for an operation $f : s_1 \dots s_n \to s$ of a signature, the corresponding operation is specified in our Prolog implementation with `op(f, [s1,...,sn,s])`. This results in the declaration of a new functor `f` with arity $n + 1$. The corresponding function in an algebra can be defined by specifying the semantics of an automatically constructed functor `f_A`, as demonstrated in the code listing below. A call of the functor `f` has the effect that it is checked

whether the arguments adhere to the types specified in the signature, and if this is the case, then the function `f_A` is called with the given arguments. Our automatic type check of the variables is done by calling the corresponding functor, i. e. for an argument `N` that has to be a natural number, `nat(N)` is called. This also has the effect that if `N` is an unbound variable it is set or constrained (see Remark 6.3.1) to the corresponding type.

Note that the logic programming language Prolog does not provide a concept of functions, and therefore it is important to consider all functions

$$f_A : A_{s_1} \times ... \times A_{s_n} \to A_s$$

in an algebra as a relation

$$f_A \subseteq A_{s_1} \times ... \times A_{s_n} \times A_s$$

corresponding to a functor `f_A`, where the first n arguments are the arguments of the function, and the last argument is the result. Further note that this has the effect that a constant is not a single value, but instead it is a unary relation that is true for exactly one value. Accordingly, for instance the functor `true_A` that corresponds to the constant $true_A = T$ in the algebra A, is defined as a functor that is true for the single element `1`.

Listing 2: Definition of algebra in Prolog (excerpt)

```
% Carrier sets:
bool(B) :- B in 0..1.

nat(N) :- nat_upper_bound(B), N in 0..B.
nat_upper_bound(16).

range([R1,R2]) :- nat(R1), nat(R2).

% Operations:
true_A(1).

false_A(0).

start_A([N,_],N).

toRange_A(N1,N2,[N1,N2]).

next_A(N1,N2) :- N2 #= N1 + 1.
```

Remark 6.3.1 (Finite Domain Constraint Programming). The type definitions that we use are not pure Prolog definitions, but we make use of the Constraint Logic Programming over Finite Domain (`clpfd`) library of SWI-Prolog. Alternatively to the definition of the functor `nat` above, it would be possible to define natural numbers between 0 and 16 using the built-in predicate `between(0,16,B)`. The difference would be, that if we call the functor `nat(N)` – which happens whenever an argument of a called operation has the type `nat` – then the variable `N` is set to a definite integer value between `0` and `16` until Prolog's backtracking mechanism fails and it is set to another value in the range. In contrast, using the constraint logic expression `N in 0..16` leaves the variable `N` unbound, but it sets the domain of possible values for the variable. Accordingly, also the operation `#=` in the body of `next_A` is not an assignment, but a constraint, that restricts variables `N1` and `N2` to be successors, even if they are not bound to specific values. Taking into account the domains from the `nat` type definition, this restricts the domain of `N1` to be between `0` and `15`, and the domain of `N2` to

be between `1` and `16`, and if one of the two variables is set to a definite value (which may happen later), then the respective other one is automatically set to its predecessor respectively successor.

Extensive use of constraint programming can have an immense impact on the performance of an algorithm, like the one for the computation of concrete realisations for abstract scenarios (see Subsection 6.3.4), because it may prevent a lot of unnecessary backtracking. △

The fact that we have to encode functions as functors in Prolog does also have ramifications for the modelling of arc inscriptions and firing conditions of AHL-nets. Since there are no functions but only functors (predicates) in Prolog, there is also no notion of composition, like the composition $g \circ f$ of functions f and g, and accordingly there are no complex terms of the form `g(f(...))`. Thus, transferring terms and equations from the pure algebra to Prolog, one has to consider the following:

1. For arc inscriptions we have the restriction that we only allow variables as arc inscriptions. This does not limit the modelling possibilities, since each complex term inscription *term* can be replaced by a variable x, and we obtain the same semantics if we add the equation $term = x$ to the firing conditions of the corresponding transition.

2. The same holds for more complex terms in equations: If we have an equation of the form $f(x_1, ...x_n) = x$, and x_1,...,x_n or x is not a variable, then the complex sub-term is replaced by a variable and we add a corresponding equation as described in the previous item.

3. An equation $f(x_1, ...x_n) = x$, where x_1,...,x_n and x are variables, corresponds to a Prolog term `f(x1,...,xn, x)`.

The model of an AHL-net in Figure 6.3 demonstrates the modelling of the AHL-net *Platform* in Figure 3.2 on page 35 in our tool, where all arc inscriptions and firing equations are modelled as Prolog terms. Note that due to technical reasons the variables of the data type part of the AHL-net are not encoded as Prolog variables, but as Prolog atoms, and accordingly, an assignment is not a binding of variables, but a mapping of the corresponding atoms to values (see Remark 6.3.2 in Subsection 6.3.4 for more details).

6.3.3 Visual Editing of Net Models

The APE tool allows the visual modelling of P/T nets, AHL-nets and instantiations. The visual user interface of the tool is shown in Figure 6.3. A new model can be created via the menu (1). For the placement of new places, transitions or arcs, the user can select the corresponding tool in the toolbar (3) and then place the net elements in the model view (4). Newly created net elements first have default values, e. g. places are named `Place 1`, `Place 2`, and so on, and arcs of AHL-nets are initially inscribed with an `x`. In order to ensure the existence of a default type of places our data module provides a default type `void` with a single element `empty`. The default values of selected net elements can be changed using the property view (6) or keyboard shortcuts. Moreover, a set of selected net elements can be deleted with a button on the toolbar (3). If a place or a transition is deleted, then also all connected arcs are deleted, in order to avoid dangling arcs without source or target. The models (together with the corresponding visual information) can be saved alone or as the whole project via the menu (1) in the file format ApeML, a human-readable XML-variant.

6.3.4 Calculation of Realisations for Abstract Scenarios

In Section 5.3 we presented a result that describes how instantiations of an AHL-process net can be computed via solutions of a term equation system. For the realisation of this result in our tool, we implemented a net module in Prolog, that allows also the definition of AHL-net models in the Prolog language. Using the corresponding menu entry in the APE tool, for the model of an AHL-net that has been created in the visual editor, the corresponding Prolog model is constructed automatically. Moreover, outgoing from that Prolog net model, the net module provides functionality for the construction and solving of the corresponding term equations as described in Section 5.3.

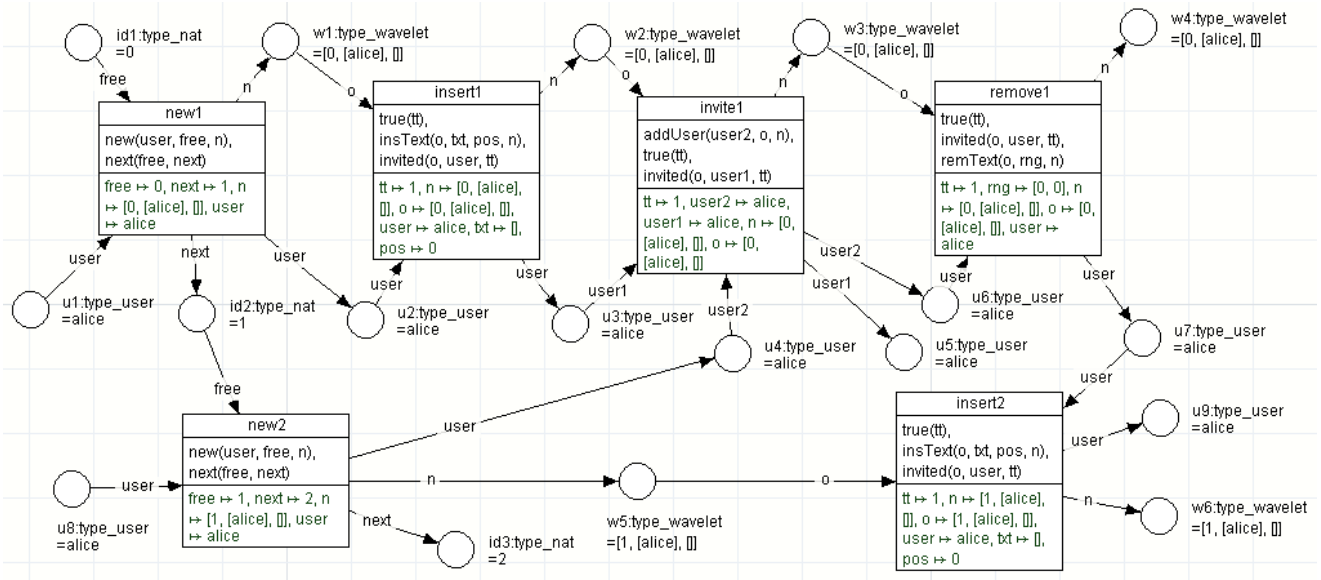


Figure 6.4: Computed realisation of an abstract scenario

An example of a computed realisation of the AHL-process net *Wave* in Figure 4.1 on page 60 is shown in Figure 6.4. Note that in our implementation of the algebra A, tuples are implemented as lists, and words (strings) are implemented as lists of integers to benefit from the constraint programming mechanisms described in Remark 6.3.1. The presented instantiation is the result of the first found solution. It is always possible to start the computation and visualisation of the next solution as long as there are still variable assignments left that are not found or already discarded by the algorithm.

It is possible to instantiate the AHL-net partially before starting to compute the (complete) realisation. This means that it is possible to define some of the data values for the places and also partial variable assignments for the transitions. The partial instantiation is taken into account by the computation of the complete instantiation, effectively constraining the solutions to assignments that comply with the predefined values. This can be used to lower processing effort of the computation, but also to restrict the results to a set of scenarios that is of particular interest, like scenarios for a given marking of the input places, or scenarios that contain a specific action with predefined values.

Remark 6.3.2 (Representation of Variables in Prolog). As already pointed out in Subsection 6.3.2, the variables occurring in an AHL-net are not encoded as Prolog variables, but as atoms. The main technical reason for this lies in the functionality of the computation of realisations for abstract scenarios. According to Definition 5.3.2 the realisation variables consist of a disjoint union of all variables occurring in the environments of transitions. In

the realisation conditions, each variable x that originally occurs in a firing condition of a transition t, is replaced by its distinct realisation variable representative that corresponds to transition t which can be expressed as a pair (x, t). If we encode the variable x by a Prolog variable `X` and the transition t by some element `t`, then (x, t) corresponds to a Prolog list `[X,t]`. But `[X,t]` is not a variable and can only be unified to terms of the form `[Y,t]`, where `Y` is an arbitrary term. So replacing `X` by `[X,t]` in a term that represents a firing condition does not have the desired effect of replacing the variable with its corresponding realisation variable. Instead, it is necessary to relate the pair `[X,t]` to another fresh variable `_X` and then use `_X` for the term replacement. When Prolog has solved the term equation system, the variable `_X` is unified with a value $[\![\texttt{\_X}]\!]$ in the (Prolog representation of the) algebra, and the relation between `[X,t]` and $[\![\texttt{\_X}]\!]$ corresponds to a part of a solution $\hat{v} : \hat{X} \to A$ with $\hat{v}(x, t) = [\![\texttt{\_X}]\!]$. So, the variable `X` is not (and never should be) replaced by any value, and therefore, it is better to represent this immutable entity as an atom. △

6.3.5 Current Development Status

The APE tool is currently still under development, and not all intended features are already implemented. In the following we give a short overview over the current status and what is still to be done, considering the requirements presented in Section 6.1:

Modelling of Communication Platforms The APE tool enables the modelling of P/T-nets, AHL-nets and instantiations, based on a generic net model that can be easily extended to other support also other types Petri nets. We intend to implement also net class transformations that allow for instance to turn an AHL-net into a P/T-net (along the skeleton functor), and to lift P/T-nets into an AHL-net corresponding to the $Data_{\mathbf{PT}}$-functor in [Tav02, Urb03]. This should support the user to concentrate on different aspects of the modelling. Signatures and algebras for the AHL-nets can be implemented in Prolog using the functors provided by our Prolog data module.

Modelling of Scenarios The modelling of AHL-nets and instantiations principally enables the user also to model (instantiated) AHL-process nets, because they are special cases of AHL-nets and instantiations. We plan to the current implementation to allow also to check whether an AHL-net is an AHL-process net. Moreover, it is planned to allow also the modelling of AHL-processes that relate a model of an AHL-process net with the model of an AHL-net in form of an AHL-morphism.

Evolution of Communication Platforms and Scenarios The evolution of communication platforms and scenarios in the sense of rule-based transformation is not yet implemented. The editor has to be extended to allow also the modelling of productions and to provide functionality for their applications.

Analysis of Communication Platforms and Scenarios Since the rule-based evolution is not yet implemented, there are also no analysis techniques for the Local Church-Rosser, Parallelism and Concurrency Theorems at the present. For the computation of realisations of abstract scenarios, the tool offers the functionality described in Subsection 6.3.4.

7 Related Work

In this chapter we give an overview on the related work of this thesis. First, in Section 7.1 we review other contributions to the formal modelling of communication platforms. Then, in Section 7.2 we give an overview on related work on the formal modelling using the modelling technique of Petri nets, as it is the basis for the modelling of communication platforms and scenarios in this thesis. Moreover, considering the fact that structural evolution of platforms and scenarios in this work are modelled using rule-based transformation in the sense of graph transformation, in Section 7.3 we review related work in the field of graph and model transformation. Finally, in Section 7.4 we review other modelling techniques that are related to the techniques used in this thesis.

7.1 Formal Modelling of Communication Platforms and Apache Wave

In this section we review some of the related work concerning the formal modelling of communication platforms. Although, there exists a lot of literature about the modelling of all kinds of communication systems, most of the literature has a different focus and different aims concerning more technical aspects of the communication, like a specific hardware or software architecture or the communication protocol. The investigation of human behaviour, on the other hand, does usually not involve the application of formal mathematical techniques. One important exception is the work of Tony Modica in [Mod12] that we discuss in the following.

Communication platforms in [Mod12] The work on the formal modelling, simulation and validation of communication platforms by Tony Modica [Mod12] is closely related to this thesis, considering the focus on communication platforms as well as the formal techniques. The main intersection between this thesis and [Mod12] is the fact that both works employ an integration of algebraic high-level (AHL-)nets and graph transformation techniques for the modelling and evolution of communication platforms. There are, however, major differences in the focus of the works.

On the conceptual layer, [Mod12] is focussed on the realisation of technical communication aspects like direct calls, multi-user chats, conferences, multicasting of data, and communication platform constraints, mainly inspired by a case study of the telephony software Skype [Sky13]. On the technical layer, the work is highly focussed on the operational firing semantics and rule-based transformation of AHL-nets. For this reason, the work is mainly based on the concept of algebraic high-level nets with individual tokens (AHLI-nets, see also [MGE+10, MGH10]).

User actions are modelled as single firing steps, or firing steps that trigger the application of a *handler*, resembling the idea of graph transformation units [Kus98, KK99] or *ActiGras* [EGLT11], extended by a concept of prioritisation of productions. It is important to note that the triggering firing step of a user action in the sense of [Mod12]

models a single atomic action like a mouse click or a keystroke. Complex user interactions have to be modelled by different user action models, and dependencies between different user interactions are difficult to verify, because the outcome of a user action depends on the result of the corresponding handler. For this reason, it is important for the results concerning independence of user actions in [Mod12] that the user actions have a functional behaviour. Moreover, the associated handler of a user action models the internal consequences of that user action on the platform, like the establishment or removal of communication channels. This means that each user action potentially changes the communication platform model that models not only the capabilities of the communication platform but also the current states of all communications on the platform.

In contrast, on the conceptual layer, this thesis is more focussed on a high-level perspective on user interactions on communication platforms, and, taking into account the potential uncertainty of human behaviour, we do not assume functional behaviour of complex user interactions. Consequently, on the technical layer, we focus mainly on the process semantics of AHL-nets that is based on the nets' firing semantics. The main advantage of this technical approach is the fact that we are not restricted to consider only single atomic user actions, but the approach of (instantiated) AHL-processes inherently supports the modelling of complex scenarios of user interaction. Accordingly, we model changes of the current state of a communication by potentially complex user interactions as applications of single productions. This allows us to analyse the independence of complex user behaviour by analysis of the corresponding direct transformations, whereas following the modelling approach in [Mod12], the consideration of complex interactions is only possible by dissection of the complex user action models into atomic pieces. This requires the simulation of each user action handler, because it is not possible to know a priori which part of a handler is actually executed in a specific situation.

Another important benefit of our modelling approach for interaction scenarios as AHL-processes is the fact that we obtain a clear distinction between the current state of the communication platform and the states of communications on the platform. While user actions in the sense of [Mod12] trigger modifications on the platform model, in our modelling approach modifications that concern only the communication can be modelled on the technical layer as transformations of AHL-processes which, in general, leave the AHL-net model of the corresponding underlying communication platform unchanged. Modifications to the capabilities of a communication platform, on the other hand, can be considered independently from the current states of communications on that platform. This distinction allows, for instance, to analyse the compatibility of platform evolutions with interactions on the platform.

Aside from the differences between the modelling approach in [Mod12] and in this thesis, the agreement of the modelling approaches in the modelling technique of algebraic high-level nets may serve as a starting point for the development of a unified integrated approach that benefits from the results of both of the works. Considering the two-layered modelling using algebraic higher-order (AHOI-)nets in [Mod12], this could be achieved in at least two different ways. Firstly, it is possible to consider an algebraic higher-order nets with AHL-nets and (instantiated) -processes together with corresponding productions for their rule-based transformations as tokens, serving as a control structure for the modelling of communication platforms and scenarios. Secondly, since an AHOI-net is a special case of an AHLI-net with a data type part that

again operates on AHLI-nets, one can think of a modelling of communication platforms and scenarios in the sense of this thesis, however, with the difference that the data type part of the AHL-net models is implemented as reconfigurable AHLI-nets in the sense of [Mod12].

Communication platforms in other literature Due to the importance and ubiquity of computer-supported communication there is a lot of different literature about models of communication platforms. However, many of the works about communication platforms are more focussed on the technical aspects of communication like the hardware architecture, the network topology or the different communication protocols. Accordingly, the notion of a platform is often used in the sense of a hardware architecture, like in [MSM97, Sie04, JLT09]. In [SV01], a platform is defined as an abstraction layer for hardware or software systems, and [LNW03] introduces relations between platforms in form of functions that resemble our approach of abstraction and realisation in Section 3.3. The Open Data platform in [LTB+12] encompasses architecture as well as software aspects of a point for publishing and sharing data. In the area of multi-agent systems, [SMM+08] presents a communication platform as an organisational structure of agents that provides trust and anonymity for mobile agents, and [Nov08] discusses requirements for suitable communication platforms supporting multi-agent and multi-robot systems. The aim of the framework of I-centric communication in [AvdMSPZ01, Arb03, Ste03] is the modelling of the entire communication behaviour of humans via I-centric services and service platforms.

Due to the fact that Apache Wave is an open-source project, there are various resources available, including for instance specifications of the underlying Google Wave Federation Protocol [BBB+09] for the server-to-server communication and the Google Wave Conversation Model [GN09] that specifies the internal structure of waves and their blips. An overview on the resources can be found at [Pro13]. [Yon10] presents the formal modelling and analysis of parts of the former communication platform Google Wave that preceded Apache Wave, using the π-calculus modelling framework. The main focus of that work lies on technical aspects concerning the underlying server-to-server and client-to-server protocols.

7.2 Petri Nets

The modelling technique of algebraic high-level nets used in this thesis is based on the general notion of Petri nets that was first introduced by Carl Adam Petri in [Pet62] and has been further researched in numerous publications, like e. g. [Roz87, Rei85, MM90].

Open Petri Nets and Processes The composition of AHL-processes presented in Chapter 4 is mainly inspired by the compositional modelling of open net processes [BCEH01, BCE+07]. Open nets are place/transition (P/T) nets with a distinguished set of places, representing the interface of the net towards the external world. The idea of open nets is partly inspired by the Petri Box Calculus (PBC) [BDH92, KEB94], where a set of process algebra-like operators allows to build complex nets from basic net components. Another inspiration for the open Petri net approach are Petri nets with interface [NPS95, PW98]. The interface consists of an input part (places) and an output part (transitions), and is used to compose different nets. A high-level variant that combines the idea of open Petri nets and algebraic specification [EM85] is presented in [UP08].

High-Level Petri Nets and Processes There are several extensions of Petri nets with

data type parts such as Predicate/Transition nets [GL81, Gen87], and Coloured Petri nets (CPNs) [Jen91, JR91, Jen97a] with data types in the programming language ML.

The combination of Petri nets and algebraic specifications [EM85] was mainly initiated by Krämer [Krä87, Krä89] and extended in [Hum89, Rei90, DHP91, EPR92, ER97] leading to the notion of algebraic high-level (AHL-)nets.

Based on the notion of low-level processes of place/transition nets, algebraic high-level (AHL-)processes were defined in [EHP+02] and further investigated in [Ehr05, EHGP09, Gab09].

Moreover, the idea of Petri nets as token objects [Val98, Val01] led to the modelling technique of algebraic higher-order (AHO-)nets [HM03, HEM05, Hof06] that are AHL-nets with nets and rules as tokens.

Petri Nets with Dynamically Changing Behaviour Other extensions of the Petri net approach concern dynamic changes of the firing behaviour of the nets. In dynamic Petri nets [Val78], arc inscriptions in the predomain of a transition may denote places that specify the number of consumed tokens. Communicative Nets and Cooperative Nets [SB91, SB94] allow to model systems as a collection of nets that can interact using message sending. In recursive Petri nets [HP99, HP00, HB08] some transitions can initiate a new token game within the same net. The approach of nets-within-nets [KR07] can be used as a mechanism for dynamic transition refinement. Moreover, in mobile Petri nets[AB09], the tokens are names for places and the name of a consumed token on firing can be used to specify a destination for the production of new tokens.

Timed Petri Nets There are many different approaches for extending Petri nets by time. In Time Petri nets (TPNs) [BD91], two labels are assigned to each transition, denoting the time that has to pass before the transition can fire after being enabled (earliest firing time), and the maximal time it may be enabled until it has to fire (latest firing time). The time nets can be analysed using a state space approach [GLMR05].

Deterministic timed Petri nets [BH07] introduce a delay between removal and creation of tokens. In addition, each place has a designated delay, denoting the time before a created token can be consumed from that place. Deterministic timed Petri nets are based on timed marked graphs [CCCS92], a subclass of Petri nets, where each place of a marked graph has exactly one input edge and one output edge.

In time environment-relationship (TER) nets [GMMP91], time is considered as token attribute (*chronos*) with special behaviour. Token timestamps are set by specific action relations associated to transitions, which also require time monotonicity of firing sequences. This approach has inspired the definition of graph transformation systems with time [GVH04], since considering time as attribute one may re-use the formal framework of attributed graph transformation.

The modelling approach of timed Coloured Petri nets (timed CPNs) [JK09] extends the high-level Petri net variant CPNs [Jen97a] by assigning a *duration* to a transition and so-called *timestamps* to each token, indicating the earliest point in time when a token can be used by a transition. A transition that fires adds the duration time of the transition to each created token's timestamp, so that in general, these tokens cannot be used immediately, but only after the time the transition takes has passed.

Finally, the algebraic approach of timed place/transition (P/T) nets [GLE12] is based on the approach of timed CPNs, but it is not an extension of CPNs, but instead it extends an algebraic approach of classical P/T nets in the sense of [MM90]. An extension of

timed P/T nets to timed AHL-nets in analogy to the extension of classical P/T nets to AHL-nets is outlined in [LGM12].

Parametrised Petri Net Classes Based on the fact that many different Petri net classes share some fundamental similarities, in [Pad96, PE01], a unified approach for Petri nets is introduced, that allows the formalisation of many different Petri net classes by actualisation of net structure and data type parameters. This leads to common abstraction of a wide range of Petri net classes, like e. g. elementary nets, low-level P/T nets, AHL-nets and CPNs, as parametrised net classes. Considering the fact that there are also different net classes relevant for this thesis, this inspired the implementation of a generic net model for our tool support that is used as superclass for the actual concrete net class models (see Subsection 6.3.1). The extended notion of universal parametrised net classes offers an abstract categorical theory of rule-based structuring of different Petri net classes.

7.3 Graph and Model Transformation

The theory of rule-based graph transformation has been introduced in the seventies [EPS73, Ehr79], resulting in a large variety of different approaches [Roz97]. A break-through for the double-pushouts (DPO) approach used in this thesis was the introduction of general categorical frameworks like adhesive [LS04], adhesive high-level replacement (HLR), and weak adhesive HLR [EHKPP91a, EEPT06b] categories, requiring the existence of pushouts and pullbacks along (a suitable class of) monomorphisms, and the satisfaction of a so-called (weak) Van Kampen property which describes a form of compatibility between pushouts and pullbacks. Almost all main results have been formulated in these abstract frameworks, including the Local Church-Rosser, Parallelism, and Concurrency Theorems, the Embedding and Extension Theorems, Completeness of Critical Pairs, and the Local Confluence Theorem [EEPT06b].

Prominent examples of adhesive categories are the categories of sets and graphs, but it turned out that for instance the category of P/T nets is not an adhesive category which led to the generalisations of adhesive HLR and weak adhesive HLR categories in [EEPT06b]. Moreover, in [EGH10] it is shown that all the main results from [EEPT06b] are also valid in the more general framework of $\mathcal{M}$-adhesive categories. The category of P/T nets as well as the categories of AHL-nets and instantiations together with suitable classes of monomorphisms are $\mathcal{M}$-adhesive categories (see Subsection A.1.3). Another generalisation of adhesive categories is given by the framework of partial VK square adhesive categories in [Hei10], and a comparison of the different categorical frameworks can be found in [EGH10].

Transformation of Petri Nets The rule-based transformation of Petri nets have been introduced for low-level P/T nets in [Pad92] and for AHL-nets in [EPR92, PER95]. Note that the rule-based transformation in the DPO-approach depends highly on the definition of the morphisms in the corresponding category. Therefore, transformation of AHL-nets presented in this thesis is slightly different from the one in previous works, due to an alteration of the definition of AHL-morphisms as pointed out in Remark 3.1.6.

$\mathcal{M}$-adhesive transformation systems of low-level P/T have been introduced in [EHPP07, EHP+07] and extended to $\mathcal{M}$-adhesive transformation systems algebraic high-level nets in [Pra07, PEHP08, Pra08]. The rule-based transformation of P/T- and AHL-systems (i. e. nets together with markings) does not allow to change the markings without replacement of the corresponding places with new ones. To overcome this restrictions, the approaches of low- and high-level Petri nets with individual tokens were introduced

in [MGE+10, MGH10, Mod12], leading to an $\mathcal{M}$-adhesive transformation systems of marked Petri nets, where it is also possible to modify the markings by transformations (see also Remark 3.1.11).

Application Conditions The application of a production usually depends on the left-hand side of the production and a match morphism from the left-hand side into the corresponding object that is to be transformed. A way to further restrict the applicability of productions are application conditions that are introduced for graph grammars in [EH86, HHT95], and extended to weak adhesive HLR systems in [EEHP04]. Moreover, the notion of application conditions has been extended to nested application conditions in [HP05] that allow to specify more complex conditions using Boolean expressions, leading to expressions that are equivalent to first order logic on graphs (or the objects of the corresponding $\mathcal{M}$-adhesive category, respectively).

Application conditions for the rule-based transformation of AHL-nets and instantiated AHL-processes are not considered in this thesis, but their inclusion in our framework is an interesting aspect for future research (see Subsection 8.4.2).

7.4 Other Modelling Techniques

The Unified Modelling Language is a large framework of different techniques for the modelling of object-oriented systems [BRJ98, BJR99] that are mainly defined informal or semi-formal. There have been several efforts for the formalisation of single parts of the UML [BHH+97, GPP98, BCD+06], but there does not exist a consistent formal semantics for the whole framework.

Another prominent technique for the formal modelling of concurrent communicating systems are process calculi or process algebras [Mil73, Hen88] like the Calculus of Communication Systems (CCS) [Mil80], the Algebra of Communicating Processes (ACP) [BK84, BK85], Communicating Sequential Processes (CSP) [Hoa85], and the π-calculus [Mil99]. In contrast to the integrated modelling framework of algebraic high-level nets and -processes used in this work, in the process calculi there is usually no explicit distinction between the modelling of a process and the system, where the process is running.

8 Conclusion

In this chapter we summarise the results of this thesis, and we give an outlook on future work. The results encompass realisations of the modelling requirements from Section 2.3 using our integrated modelling framework of algebraic high-level (AHL-)nets and processes, as described in Section 8.1. Moreover, in Section 8.2 we review our results for the analysis problems in Section 2.4. In Section 8.3 we review some general categorical results that are used for the modelling and analysis in this thesis, and that can also be used for different modelling techniques and application domains. Finally, in Section 8.4 we give an overview over the future work.

8.1 Realisation of Modelling Requirements

In this section we review the realisation of the requirements for the modelling of communication platforms and scenarios that we discussed in Section 2.3 using our integrated framework of algebraic high-level (AHL-)nets and processes. Note that each realisation of a modelling concept is accompanied by a corresponding example concerning our Apache Wave case study in Section 2.2. An overview over the realisations is given in Table 8.1. The realisation of modelling requirements includes adequate modelling techniques for communication platforms and scenarios, as well as techniques for the evolution of these models. Moreover, our concept for the modelling of scenarios is also suitable for the modelling of histories.

Modelling of Platforms As pointed out in Section 2.3, for the modelling of the platforms, it is important to allow the modelling of the users as well as all sorts of data that is product of the communication between these users. Moreover, the model of a has to provide actions that can be used by the users to communicate with each other. In Concept 3.1.8 we describe that AHL-nets are a suitable modelling technique for the modelling of communication platforms. This is exemplified in Example 3.1.9, where an Apache Wave platform is modelled as an AHL-net.

In some cases such as the different manifestations of Apache Wave, there may be different platforms with a different set of actions or data, but users on different platforms may still communicate with each other due to a common communication protocol. This means that the modelling of communication platforms should be compositional, allowing to view a combination of platforms as a platform. As described in Concept 3.2.8, the composition of communication platforms can be modelled by the gluing of the corresponding AHL-net models. Moreover, in Theorem 4.5.14 we have shown that the process semantics of AHL-nets is compositional. Examples of the composition of different Apache Wave platforms are given in Example 3.2.9.

Evolution of Platforms Another requirement for the modelling of communication platforms is the possibility to model also their evolution, allowing to change resources, data

Table 8.1: Realisation of modelling requirements in this thesis

Concept	**Realisation**
Modelling of platforms	Concept 3.1.8: Algebraic high-level nets
Composition of platforms	Concept 3.2.8, Fact 3.2.7: Gluing of AHL-nets Theorem 4.5.14: Compositional process semantics of AHL-nets
Evolution of platforms	Concept 3.2.16, Fact 3.2.14: Rule-based transformation of AHL-nets, Concept 3.3.2: Data-evolution of AHL-nets
Abstract and semi-abstract scenarios	Concept 4.1.7: Algebraic high-level processes, Concept 4.3.6: Abstract instantiations of AHL-processes
Concrete scenarios	Concept 4.3.6: Concrete instantiations of AHL-processes
Levels of abstraction	Concept 3.3.8: Levels of abstraction of communication platforms, Concept 4.4.5: Levels of abstraction of scenarios, Theorem 3.3.10: Compatibility of abstraction and structuring
Composition of scenarios	Concept 4.2.7, Fact 4.2.6: Gluing of AHL-processes, Concept 4.5.9, Fact 4.5.8: Gluing of instantiated AHL-processes
Evolution of scenarios	Concept 4.2.13, Theorem 4.2.11: Transformation of AHL-processes Concept 4.6.9, Theorem 4.6.8: Transformation of instantiated AHL-processes, Concept 4.4.4, Fact 4.4.3: Data evolution of scenarios
Views on scenarios	Concept 4.5.15, Theorem 4.5.13: Restriction and Amalgamation of AHL-processes and instantiations
Modelling of histories	Concept 4.7.3: Finite and strict instantiated AHL-processes
Progression of histories	Concept 4.7.11, Fact 4.7.10: Sequential transformation of instantiated AHL-processes
Behaviour of robots	Concept 4.7.11: Sets of sequential productions for instantiated AHL-processes

or actions of the platform. In Concept 3.2.16 we describe how the structural evolution of communication platforms can be modelled by rule-based transformation of the corresponding AHL-nets. This allows to change the resources and actions of the platform. In Example 3.2.17 we consider an example of the structural evolution of an Apache Wave platform.

Moreover, in Concept 3.3.2 we present the modelling of the evolution of the data types of a communication platform by data-images of the corresponding AHL-nets along generalised algebra homomorphisms. Example 3.3.3 contains an example of the data-evolution of an Apache Wave platform.

Modelling of Scenarios Concerning the interactions of human-centric communication platforms, a modelling technique for different scenarios of interactions on a platform is

required. A scenario contains a number of actions performed by users, and these actions have to be in compliance with the actions and data that can be used on the corresponding platform. The model of a scenario should reflect the causal relation of subsequent actions, but it should also be possible to model the concurrent occurrence of actions, since usually it is possible that different users perform actions independently at the same time (see also the discussion on page 23 about a tolerant interpretation of simultaneity in near-real time communication).

These requirements are fulfilled by the modelling technique of AHL-processes that can be used for the modelling of abstract scenarios as described in Concept 4.1.7. An abstract scenarios captures the causal and logical relations between actions and resources, but it does not contain any definite data values that are used or present throughout the scenario. An example of an abstract scenario in an Apache Wave platform is given in Example 4.1.8.

Further, concrete scenarios that consist also of specific data values that are used or present throughout the scenario can be modelled as instantiated AHL-process as described in Concept 4.3.6. This includes also the modelling of semi-concrete scenarios, i. e. it is possible to specify the data values by terms that may be instantiated to different concrete values. Examples of concrete and semi-concrete scenarios in an Apache Wave platform are given in Example 4.3.7 and Example 4.4.6.

Modelling of Histories As pointed out in Concept 4.7.3, histories as a finite special case of concrete scenarios can also be modelled by finite instantiated AHL-processes.

Evolution of Scenarios Analogously to the evolution of communication platforms it is possible to model also the structural evolution of scenarios by rule-based transformation of the corresponding models. The evolution of abstract scenarios is modelled as rule-based transformation of AHL-processes, as described in Concept 4.2.13. The evolution of concrete and semi-concrete scenarios can be modelled as rule-based transformation of instantiated AHL-processes, as described in Concept 4.6.9. Examples of the structural evolution of scenarios of an Apache Wave platform are given in Example 4.2.14 and Example 4.6.10. Moreover, not only the data-evolution of communication platforms, but also the data-evolution of scenarios can be modelled by data-images, as pointed out in Concept 4.4.4.

Moreover, the progression of histories by user interactions can be modelled as rule-based transformation of instantiated AHL-processes using sequential productions as described in Concept 4.7.11. An example of the progression of the history of a wave by interactions of a robot is considered in Example 4.7.12.

Levels of Abstraction The possibility to model abstract, semi-concrete as well as concrete scenarios allows the modelling of different levels of abstractions for scenarios. Moreover, the different possible abstraction levels of a scenario can be formally related as described in Concept 4.4.5. Levels of abstraction for scenarios of an Apache Wave platform are considered in Example 4.4.6.

This can not only be done for single scenarios, but according to Concept 3.3.8 it is also possible to consider different abstraction levels of whole communication platforms. Different abstraction levels of an Apache Wave platform are considered in Example 3.3.9. Moreover, in Theorem 3.3.10 it is shown that our abstraction mechanism is compatible with structuring of the models.

Views on Scenarios Regarding the compositionality of platforms described above, it should also be possible to restrict scenarios to the current view on the platform. This is particularly interesting in the context of cross-platform communication, where different platforms together form a common larger platform. The views on scenarios can be modelled by restriction and amalgamation of (instantiated) AHL-processes as described in Concept 4.5.15. An example of different views on models of scenarios of Apache Wave platforms is given in Example 4.5.16.

Behaviour of Users and Robots Analysing the interactions in human-centric communication platforms, we require not only a formal model of the platform, but also a suitable model of the behaviour of the users. This includes the behaviour of humans, and, considering robots as a special case of automated users, also the behaviour of robots. The behaviour of users and robots can be modelled as sets of sequential productions for instantiated AHL-processes as described in Concept 4.7.11. In Example 4.7.12 we describe the modelling of the behaviour of a robot that detects and corrects typographical errors.

8.2 Results for Analysis Problems

Based on our modelling approach for communication platforms and scenarios, we developed techniques that allow the analysis of different problems. Considering the analysis problems discussed in Section 2.4, this thesis provides several results, as described in the following. An overview over the analysis results can be seen in Table 8.2. The analysis results concern for instance the independence of platform as well as scenario evolutions, the compatibility between platform evolutions and existing scenarios, and the set of all concrete realisations that exist for a given abstract scenarios.

Compatibility and Composability of Platform Evolutions Especially in the case of open-source platforms, it can happen that outgoing from one platform there are different evolutions of that platform, introducing or removing features or resources, respectively. Considering two of these evolutions, it is an interesting question if the evolutions exclude or depend on each other. If this is not the case, it may be possible that the evolutions are compatible with each other, in the sense that one evolution can also be applied to the resulting platform of the other one and vice versa. This problem can be analysed by investigating the independence of the corresponding rule-based transformations of AHL-nets, as stated in Concept 5.1.5, and shown in the the Local Church-Rosser Theorem for AHL-nets (Theorem 5.1.4). Note that the Local Church-Rosser Theorem for AHL-nets is based on the more general Local Church-Rosser Theorem for weak adhesive high-level replacement (HLR) systems in [EEPT06b] that we instantiated to the special case of AHL-nets. An example of independent evolutions of Apache Wave platforms is given in Example 5.1.1.

In the case that we have independent platform evolutions, there should be a way to unite the different evolutions and obtain one evolution, consisting of the benefits of either of the single evolutions, but with the difference that the platform evolves in a single step. This can be achieved using the Parallelism Theorem for AHL-nets (Theorem 5.1.7) that is an instantiation of the Parallelism Theorem for weak adhesive HLR systems in [EEPT06b] to AHL-nets. An example of the parallel evolution of Apache Wave platforms is given in Example 5.1.8.

In the case of dependencies on the other hand, the preferred result would be a composition of the evolutions in the right order, assuring that no conflicts occur during the

Table 8.2: Analysis of problems in this thesis

Concept	**Result**
Independence of platform evolutions	Concept 5.1.5, Theorem 5.1.4: Local Church-Rosser Theorem [EEPT06b] for AHL-nets
Parallel platform evolutions	Theorem 5.1.7: Parallelism Theorem [EEPT06b] for AHL-nets
Concurrent platform evolutions	Theorem 5.1.12: Concurrency Theorem [EEPT06b] for AHL-nets
Cross-platform communication	Theorem 4.5.13 Restriction and Amalgamation of AHL-processes
Independence of scenario evolutions	Concept 5.2.6, Theorem 5.2.4: Local Church-Rosser Theorem for AHL-process nets
Parallel scenario evolutions	Theorem 5.2.8: Parallelism Theorem for AHL-process nets
Concurrent scenario evolutions	Theorem 5.2.14: Concurrency Theorem for AHL-process nets
Compatibility between scenarios and platform evolutions	Concept 4.8.5, Theorem 4.8.4: Extension of AHL-processes Concept 4.8.16, Theorem 4.8.14: AHL-process evolution based on action evolution, Concept 4.8.26, Theorem 4.8.25: Evolution of instantiated AHL-processes based on action evolution
Concrete realisations of abstract scenarios	Theorem 5.3.3: Realisation conditions for AHL-process nets

cumulative evolution of the platform. For this problem, we provide the Concurrency Theorem for AHL-nets (Theorem 5.1.12) that is as well an instantiation of the corresponding result for weak adhesive HLR systems in [EEPT06b]. An example of the concurrent evolution of Apache Wave platforms is shown in Example 5.1.14.

Compatibility between Scenarios and Platform Evolutions Moreover, due to the fact that platforms can evolve, we need a way to check if scenarios of the original platform are still in compliance with the result of the modification. Otherwise, it is an interesting question, if scenarios can be adapted appropriately, in order to obtain a consistent scenario corresponding to the new platform.

The Extension Theorem for AHL-processes (Theorem 4.8.4) provides a sufficient and necessary condition that allows to analyse whether an AHL-process can be extended along a transformation of the corresponding system net. As described in Concept 4.8.5 this allows the analysis of the compatibility of scenarios with platform evolutions, and it provides a mechanism to obtain a valid scenario for the evolved platform if they are compatible. Note that despite the fact that the Extension Theorem is only formulated for AHL-processes, it is also applicable to instantiated AHL-processes, as pointed out in Remark 4.8.2, and thus, the result can be applied not only to abstract but also to concrete scenarios. An example of the extension of a scenario based on the evolution of an Apache Wave platform is shown in Example 4.8.6.

Moreover, for cases of platform evolutions, where it is necessary to adapt the scenario in order to obtain compliance with the result of the evolution, we introduced the concept of action evolution patterns. In Theorem 4.8.14 and Theorem 4.8.25 we show under which conditions we obtain an evolution of (instantiated) AHL-processes that is based on an action evolution with a given pattern. In Concept 4.8.16 we describe how this can be used for the evolution of abstract scenarios based on platform evolutions, and similar, Concept 4.8.26 describes the evolution of concrete scenarios based on platform evolution. The results are exemplified for an abstract scenario of an Apache Wave platform in Example 4.8.17, and for a concrete scenario of an Apache Wave platform in Example 4.8.27.

Concrete Realisations of Abstract Scenarios Considering abstract scenarios without any specific data or users, the question arises if there are at all concrete values that can be used to obtain a concrete scenario that is a realization of the abstract scenario. According to Concept 4.4.5, the model of the realisation of an abstract scenario modelled by an AHL-process mp is a corresponding instantiation $(inst, mp)$. In Theorem 5.3.3 we show how all concrete realisations of an abstract scenario can be computed as solutions of a term equation system. This result applied to an abstract scenario of an Apache Wave platform in Example 5.3.4.

Conflicts between Interactions Considering interactive (near-)real-time communication using modern platforms it is possible that many users perform different action at the same time. In the case that they are using the same resources this may lead to conflicts. This can especially be a problem when a variety of different robots is involved in the communication. It would be helpful to detect these conflicts at design-time to prevent problems that may occur during the communication.

As described in Concept 4.7.11, user interactions can be modelled via sequential transformations of instantiated AHL-processes, a special case of the modelling technique for the evolution of concrete scenarios. In Section 5.2 we transfer the results concerning the analysis of platform productions to corresponding results for the analysis of scenarios. Note that the general Local Church-Rosser, Parallelism and Concurrency Theorems for weak adhesive HRL systems in [EEPT06b] are also valid for the more general categorical framework of $\mathcal{M}$-adhesive categories (see [EGH10]), and therefore, they are also applicable to instantiations and typed AHL-nets, as pointed out in Remark 5.2.1. Due to the fact that (instantiated) AHL-processes are a special case of typed AHL-nets, where the domain is an (instantiated) AHL-process net, in Section 5.2 we concentrate only on the analysis of AHL-process nets. The application of the analysis results to instantiated AHL-processes is only a matter of trivially combining all results for AHL-process nets, instantiations and typed AHL-nets.

For the Local Church-Rosser Theorem for AHL-process nets (Theorem 5.2.4, we introduced the notion of strong independence of AHL-process net transformations that can be used to describe the independence of scenario evolutions, as stated in Concept 5.2.6. An example of independent scenario evolutions on an Apache Wave platform is shown in Example 5.2.7. Given independent scenario evolutions, it is also possible to synthesise and analyse a parallel production that performs the same scenario evolutions in a single step, using the Parallelism Theorem for AHL-process nets in Theorem 5.2.8. An example of the parallel evolution of scenarios in an Apache Wave platform is given in Example 5.2.10. The synthesis and analysis of direct transformations that correspond to non-independent scenario evolutions is shown in the Concurrency Theorem for AHL-

process nets (Theorem 5.2.14). Example 5.2.17 shows an example of concurrent scenario evolutions in an Apache Wave platform.

Note that considering the modelling of interactions as instantiated AHL-processes, the independence of scenario evolution is a quite strict criterium. It remains an interesting aspect for future work to consider weaker notions of the independence of scenarios, as we discuss in Subsection 8.4.1.

Cross-Platform Communication Another interesting aspect for the analysis is the possibility of communication participants on different platforms to communicate with each other. Considering the communication interface it may be necessary to exchange specific resources in order to ensure that the communication works correctly.

In our modelling framework, the communication over different platforms can be expressed by interactions on the composition of the different platform models. This corresponds to an amalgamated scenario in the sense of Concept 4.5.15. Accordingly the ability to join different scenarios on different platforms for a cross-platform communication can be checked via agreement of the corresponding AHL-processes, because as shown in Theorem 4.5.13, every amalgamated AHL-process bijectively corresponds to an agreement over the interface. It remains open to investigate possibilities for the case that we do not have an agreement on the interface.

8.3 Categorical Results

In this section we review some general categorical results that have been researched in this thesis. These results are used for some of the main results, and they may serve as a basis for interesting future extension, like the introduction of application conditions for AHL-net and -process transformations (see Subsection 8.4.2), or a generalised categorical theory of non-$\mathcal{M}$-adhesive transformation systems of AHL-process nets and -processes with and without (weak) instantiations.

8.3.1 Functor Creations and Cocreations of Petri Net Categories

A functor creation, or F-creation, as defined in Definition A.1.12 is a construction that for a functor $F : \mathbf{C} \to \mathbf{D}$ and a morphism $f : A \to F(B)$ in $\mathbf{D}$ yields an object $\bar{A}$ and morphism $\bar{f} : \bar{A} \to B$ in $\mathbf{C}$ such that $F(\bar{A}) = A$ and $F(\bar{f}) = f$, and the morphism satisfies a universal property (see Definition A.1.12 for details).

The notion of F-creations resembles the idea of cofree constructions, with the difference that a cofree construction $f^* : A^* \to B$ of a $\mathbf{D}$-morphism $f : A \to F(B)$ depends only on f, while the F-creation $\bar{f} : \bar{A} \to B$ of f via $\mathbf{C}$-object B depends on both, f and B. In fact, as shown in Fact A.1.18, an F-creation via an object B is a cofree construction, although not with respect to functor F, but with respect to the so-called slice functor $F \setminus B$.

Moreover, F-creations can be characterised by cartesian (or prone) morphisms, as shown in Fact A.1.15. The notion of cartesian morphisms is usually used in a rather pure mathematical context. Therefore, it is often formulated in higher category theory, and exemplified by mathematical examples like groups, rings and topological spaces, making it difficult to access the theory from a computer scientific perspective. For this reason, we define F-creations as a general construction rather than an abstract mathematical property.

In Section A.6 we show that there are several functor creations along functors between different Petri net classes that are relevant for this thesis, like the functor $Sys : \mathbf{Procs} \to \mathbf{AHLNets}$ that maps a process $mp : K \to AN$ to the system net AN, or the functor

Net : **Inst** $\to$ **AHLNet** that maps an instantiation $(inst, AN)$ to the corresponding AHL-net AN. The abstract categorical functor creations also have concrete counterparts that are relevant in the conceptual main part of the thesis. For instance, a Sys-creation of an AHL-morphism f via an AHL-process mp is exactly the restriction of the AHL-process mp along f, and a Net-creation of an AHL-morphism f via an instantiation $Inst$ is exactly the restriction of $Inst$ along f.

Further, in Subsection A.1.4 we show some general results for functor creations of which the most important one is that they create pullbacks (see Fact A.1.27). This is for example used in the proof of Theorem 4.8.14, where we use the fact that the horizontal AHL-morphisms of the construction are T-creations with T : **AHLNets** $\to$ **Sets** being the functor the projects an AHL-net to its set of transitions. Creations along the injection W : **Inst** $\to$ **wInst** that maps concrete instantiations to their representation as weak instantiation are used to show the reflection of concrete instantiations along AHL-morphisms with isomorphic data type part in Lemma A.5.5. This in turn is used for the proof of the equivalence of the consistent creation condition and instantiation condition in Fact 4.6.13. Moreover, the existence of Net- and $wNet$-creations allows to easily extend the construction of $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisations of AHL-nets to a corresponding construction for (weak) instantiations in Fact A.4.5. This is the basis for the construction of $\mathcal{E}'$-$\mathcal{M}'_{AHL}$ factorisations of instantiations that is necessary for applying the Concurrency Theorem also to the categories of (weak) instantiations (see Remark 5.2.1).

In Definition A.1.28 we define the dual notion of F-cocreations, and for each of the results for F-creations, we obtain a dual result for F-cocreations, like the fact that F-cocreations create pushouts (see Corollary A.1.33). The most important examples of functor cocreations in this thesis are the $Data$-cocreations with $Data$: **AHLNets** $\to$ **Algs** being the projection from AHL-nets and -morphisms to their data type part, and the $InstData$-cocreations with $InstData = Data \circ Net$ being the corresponding projection for instantiations. The $Data$- and $InstData$-cocreations correspond exactly to the data-images of AHL-nets and instantiations, respectively. This is for instance useful for the proof of Theorem 4.6.8, because the data-shifting of a production for instantiations is based on pushouts created by functor creations. Moreover, $Data$- and $InstData$-cocreations are categorical generalisations of our abstraction and data evolution techniques for AHL-nets and instantiations, respectively. Thus, the creation of pushouts by $Data$- and $InstData$-cocreations is a key factor in the proofs for the compatibility of structural and data evolution (Fact 3.3.4), of abstraction and structuring (Theorem 3.3.10), and also the compatibility of AHL-processes with data evolution and abstraction (Fact 4.1.9). Additionally, the universal property of cocreations allows to easily extend to data-image construction of instantiations also to a corresponding construction for instantiated AHL-processes (Fact 4.4.3).

Despite the fact that the general theory of functor creations and cocreations has already been useful in the course of this thesis, it may also help for future research. One possible application area are application conditions for the rule-based transformation of instantiations, as discussed further in Subsection 8.4.2.

8.3.2 Pullbacks of Petri Net Categories

The categorical notion of pullbacks is used in this thesis mainly for the mathematical formalisation of restriction constructions for AHL-processes and instantiations in Section 4.5. Since it is possible to define a quasi-category **Cat** of all categories, where the objects are all

categories and the morphisms are all functors,[32] it is also possible to apply the idea of the pullback construction also to pullbacks over co-spans of functors.

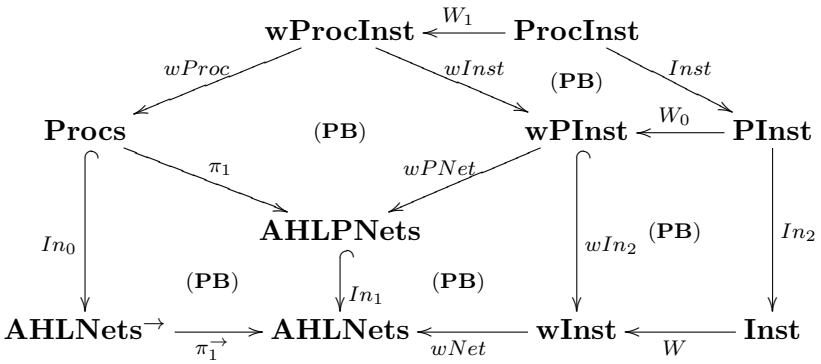


Figure 8.1: Pullback diagrams of Petri net categories

In Subsection A.1.5 we show that almost all Petri net and process categories that are used throughout this thesis together with suitable functors between them form pullback diagrams of categories, as depicted in Figure 8.1. This gives us an interesting picture of how the Petri net and process categories are connected to each other.

An important strength of the general categorical framework of $\mathcal{M}$-adhesive categories is the fact that they are closed under several categorical constructions like product, comma or functor categories. This allows to show the required properties of an $\mathcal{M}$-adhesive category only for some basic categories and then obtain the same results also for the large variety of their more complex compositions. For instance, the proof of the fact that the category $(\mathbf{Inst}, \mathcal{M}_{AHL})$ is in $\mathcal{M}$-adhesive (Fact A.5.9) is mainly based on the fact that the category $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ is $\mathcal{M}$-adhesive, and the category **Inst** is a full subcategory of the comma category $Skel \downarrow Flat$.

Unfortunately, there is no suitable class $\mathcal{M}$ of monomorphisms such that the category **AHLPNets** of AHL-process nets is an $\mathcal{M}$-adhesive category. However, in Section 4.2 we still obtain a useful transformation system of AHL-process nets by considering only those gluings of AHL-process nets that are also pushouts in the category **AHLNets**, and accordingly, we consider only direct transformations of AHL-process nets that are also direct transformations of AHL-nets.

One can say that we consider only those direct transformations that can be *translated* and *created* by the functor $In_1 : \mathbf{AHLPNets} \to \mathbf{AHLNets}$ in the sense of [MEE12], where in contrast to this work, $\mathcal{M}$-functors between $\mathcal{M}$-adhesive categories are considered that translate and create all direct transformations. So, in order to obtain a general categorical framework for transformation systems of process categories, it may be worth investigating the translation of transformations along functors with weaker assumptions.

A first step in this direction is the fact that functor creations and cocreations are closed under pullback categories, as shown in Fact A.1.39 and Corollary A.1.40, respectively, leading to the fact that all functors in Figure 8.1 have creations for certain classes of morphisms, as shown in Section A.6. It remains an open task for future research to use this knowledge for the definition of a general categorical framework of transformation systems for processes that rely on the transformation systems of underlying net categories. Such a framework may allow the formalisation of one general mechanism for transferring results for the transformation of

[32] A category usually has a class of objects. The collection of all categories, however, is no proper class in the sense of axiomatic set theory, and therefore, **Cat** is not a proper category.

basic categories to the transformation of more complex categories constructed as a pullback of categories. Up to now, this is not done in a unified way, but separately for each single case.

8.4 Future Work

In this section we present some aspects for future research that remains to be done outgoing from the results of this thesis. In Subsection 8.4.1, we give an outlook on additional techniques for the analysis of the independence of user interactions. Moreover, in Subsection 8.4.2, we discuss the extension of the evolution of AHL-nets, -processes and instantiations by a control mechanism called application conditions. In Subsection 8.4.3, we briefly discuss the modelling and analysis of safety and security properties, using the techniques presented in this thesis. Finally, in Subsection 8.4.4, we discuss the application of our framework for the modelling and analysis of communication platforms to other case studies and application domains.

8.4.1 Conflicts and Independence of User Interactions

In Section 4.7 we presented the modelling of user behaviour as sequential transformations of finite AHL-process nets. Since this is a special kind of direct transformations of AHL-process nets, it allows the analysis of the independence of user interactions, using the corresponding results for strong independence of direct transformations of AHL-process nets in Section 5.2. However, as the name suggests, the strong independence of AHL-process net transformations is a quite strong condition, restricting the analysis of independence to a subset of all cases that might be of interest. Interpreting the output places of an instantiated AHL-process net that models a history of a wave as the resources that are available in the current state of the wave, strong parallel independence of two sequential transformations requires that none of the transformations consumes resources that are needed by the respective other one. Strong sequential independence, on the other hand, requires that the second sequential transformation does not require resources that are created by the first one. Given strong parallel or sequential independence, the Local Church-Rosser Theorem for AHL-process nets (Theorem 5.2.4) implies that the user interactions that correspond to the sequential transformations can be applied in different order, leading to the exact same history and state of the wave. A weaker notion of independence may allow that one of the sequential transformations to uses or creates resources that are needed for the other one, and vice versa, but only if after (or before) the first transformation there are still suitable other resources that can be used by the other transformation instead.

Considering a case of a weaker parallel independence, for sequential transformations $K \overset{\varrho_1,m_1}{\Rightarrow} K_1$ and $K \overset{\varrho_2,m_2}{\Rightarrow} K_2$ this can be expressed by AHL-morphisms $a_1 : D \to R_1$ and $a_2 : D \to R_2$ in Figure 8.2, where the dependency relation $L_1 \leftarrow D \to L_2$ is constructed as pullback (PB), and we require that $a_1(P_D) \subseteq OUT(R_1)$ and $a_2(P_D) \subseteq OUT(R_2)$.[33]

The AHL-net D contains a representative for each pair of places $p_1 \in L_1$ and $p_2 \in L_2$ that are matched to the exact output place in K, indicating a conflict over a resource that is used by both of the transformations.[34] Now, the AHL-morphisms $a_1 : D \to R_1$ and $a_2 : D \to R_2$ express alternative matchings for the conflicting elements, allowing us to bypass the conflict

[33] The dependency relation $L_1 \leftarrow D \to L_2$ must not be confused with an E-dependency relation $L_1 \to E \leftarrow L_2$ of ϱ_1^{-1} and ϱ_2 as used in the Concurrency Theorems. In fact, this kind of dependency relation was used in an early version of a Concurrency Theorem for high-level replacement systems in [EHKPP91b, EHKPP91c].

[34] Note that this resource-conflict is different from a delete-use conflict in the sense of graph transformation. The fact that a resource is used does not mean that there is anything deleted in the net, but it only means that the matched place will probably not remain an output place.

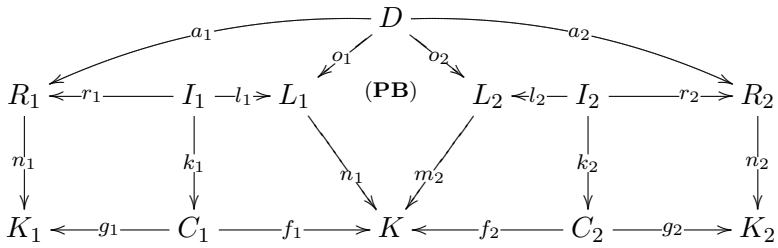


Figure 8.2: Weak parallel independence of sequential transformations

by definition of matches $m'_1 : L_1 \to K_2$ and $m'_2 : L_2 \to K_1$ with

$$m'_1(x) := \begin{cases} g_2 \circ f_2^{-1} \circ m_1(x) & \text{, if } x \notin o_1(D); \\ n_2 \circ a(x) & \text{, if } x \in o_1(D). \end{cases}$$

and an analogous definition of m'_2. Note that the matches are well-defined, because the sequential productions are non-deleting, and thus, f_1 and f_2 are isomorphisms. So instead of requiring that the same resources are still available in the result of the respective other transformation, we require that there is at least some resource that is either retained or produced by the other production.

Of course, using these constructed matches, it cannot be expected that we obtain the same result as with the original match. However, it would be desirable that applying the transformations in different order, we still obtain equal states of the wave, although it cannot be expected that the resulting histories are equal. Instead, we may require that there is some sort of equivalence between the resulting histories, requiring that it contains similar occurrences of actions, but possibly in a different order.

The sequential transformation of instantiated AHL-processes is inspired by the sequential composition of instantiated AHL-processes (based on AHL-occurrence nets instead of AHL-process nets) in [EHGP09, Gab10], where we already defined notions of the equivalence of instantiated AHL-occurrence nets and AHL-processes, and independence of their sequential composition. An equivalence e between two instantiated AHL-processes $(inst_1, mp_1)$ and $(inst_2, mp_2)$ is a pair (e_P, e_T) of bijections between the sets of places and transitions, respectively, that does not necessarily constitute an AHL-morphism, but satisfies the following conditions:[35]

1. The functions e_P and e_T are compatible with the AHL-processes, i. e. it is required that $mp_{2,P} \circ e_P = mp_{1,P}$ and $m_{2,T} \circ e_T = mp_{1,T}$.

2. A place p is an input or output place of the first AHL-process with $inst_{1,P}(p) = (a, p)$, if and only if $e_P(p)$ is an input respectively output place of the second AHL-process with $inst_{2,P}(e_P(p)) = (a, e_P(p))$.

Independence, on the other hand is a property of sequential compositions of instantiations $(Inst_1, Inst_2)$ w. r. t. $(Inst_0, i_1, i_2)$ and $(Inst_2, Inst_1)$ w. r. t. $(Inst_0, i_3, i_4)$. For the sake of brevity, we do not recall the definition of independence, but we refer to [EHGP09, Gab10] for the full details. In [EHGP09, Gab10] it is shown that independent sequential transformations lead to equivalent results.

[35] In [EHGP09, Gab10] only AHL-nets and -morphisms with a fixed data type part (Σ, A) are considered. A generalisation of the notion of equivalence to AHL-process nets with different data type part may require an extension with the involvement of components (e_Σ, e_A) for the data type part.

It is an interesting task to transfer this idea of independence from compositions to sequential transformations of AHL-processes, in order verify if we obtain equivalent transformation results. The question if different orders of transformations with alternative matches, as defined above, lead to equivalent results, according to the Concurrency Theorem is equivalent to the question if applications of the E-concurrent production $\varrho_1 *_{E_1} \varrho_2$ of ϱ_1 and ϱ_2 via m_1 and m_2', and the E-concurrent production $\varrho_2 *_{E_2} \varrho_1$ of ϱ_2 and ϱ_1 via m_2 and m_1' lead to equivalent results. Since the sequential productions ϱ_1 and ϱ_2 are non-deleting, $\varrho_1 *_{E_1} \varrho_2$ and $\varrho_2 *_{E_2} \varrho_1$ should have isomorphic left-hand sides. So, taking into account the construction of E-related AHL-net transformations in Fact 5.1.13, we conjecture that we obtain equivalent transformation results if we have independent compositions of instantiations, corresponding to compositions of AHL-process nets (R_1, R_2) w. r. t. $(D, r_1 \circ l_1^{-1} \circ o_1, a_2)$ and (R_2, R_1) w. r. t. $(D, a_1, r_2 \circ l_2^{-1} \circ o_2)$. A detailed formalisation of all the required theory and a formal proof of the conjecture is an open task for future research.

8.4.2 Application Conditions

Application conditions, that are introduced for graph grammars in [EH86, HHT95], and extended to weak adhesive HLR systems in [EEHP04], are a mechanism that allow to control the applicability of productions for rule-based transformation. The notion of application conditions has been extended to nested application conditions in [HP05] that allow to specify more complex conditions using Boolean expressions, leading to expressions that are equivalent to first order logic on graphs (or the objects of the corresponding $\mathcal{M}$-adhesive category, respectively). Nested conditions ac_P over an object P are inductively defined as follows:

- $true$ is a nested condition over P.
- For every morphism $a : P \to C$ and nested condition ac_C over C, $\exists(a, ac_C)$ is a nested condition over P.
- A nested condition can also be a Boolean formula over nested conditions, i. e. for nested conditions ac_P and $(ac_{P,i})_{i\in\mathcal{I}}$ also $\neg ac_P$, $\vee_{i\in\mathcal{I}} ac_{P,i}$ and $\wedge_{i\in\mathcal{I}} ac_{P,i}$ are nested conditions over P.

An application condition for a production $\varrho : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$ is a nested condition ac_L over L. The application of a production usually depends on the left-hand side of the production and a match morphism from the left-hand side into the corresponding object that is to be transformed. If a production has a nested application condition ac_L, the production is only applicable at a given match m, if m also satisfies the application condition. Satisfaction of a nested condition ac_P over P by a morphism $m : P \to G$, written $m \vDash ac_P$, is given if:

- $ac_P = true$,
- $ac_P = \exists(a, ac_C)$ with $a : P \to C$ and there exists $q : C \to G \in \mathcal{M}$ such that $q \circ a = m$, and $q \vDash ac_C$ (see Figure 8.3),
- $ac_P = \neg ac_P'$ and $m \nvDash ac_P'$,
- $ac_P = \vee_{i\in\mathcal{I}} ac_{P,i}$ and $m \vDash ac_{P,i}$ for some $i \in \mathcal{I}$, or
- $ac_P = \wedge_{i\in\mathcal{I}} ac_{P,i}$ and $m \vDash ac_{P,i}$ for all $i \in \mathcal{I}$.

Nested application conditions are well-researched in the context of $\mathcal{M}$-adhesive transformation systems [EGH+12a, EGH+12b], including analysis results for parallelism, concurrency, embedding, critical pairs and local confluence for transformation systems with nested

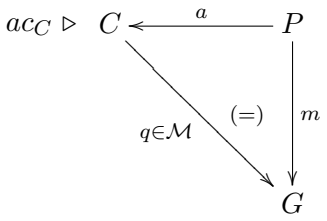


Figure 8.3: Satisfaction of nested condition

application conditions. The category **AHLNets** of AHL-nets that is used for the modelling of communication platforms in this thesis, together with the class $\mathcal{M}_{AHL}$ of monomorphisms with isomorphic data type part forms an $\mathcal{M}$-adhesive category. This means that there is already a large variety of techniques for the modelling and analysis of AHL-net transformations with nested application conditions available. The same holds for the categories of instantiations and weak instantiations, since we show in Fact A.5.9 that they form also $\mathcal{M}$-adhesive categories. Unfortunately, the general theory for $\mathcal{M}$-adhesive transformation systems cannot be applied directly to transformation systems of AHL-processes.

In the case of AHL-process nets and AHL-processes we do not have suitable $\mathcal{M}$-adhesive categories that would allow to use the general results for $\mathcal{M}$-adhesive transformation systems with nested application conditions. However, the category **AHLPNets** of AHL-process nets is a full subcategory of the category **AHLNets** of AHL-nets, and direct transformations of AHL-process nets are defined as a special case of direct transformations of AHL-nets. Accordingly, in principal it is possible to use a theory for nested application conditions for AHL-net transformations also for nested application conditions for AHL-process net transformations. This would mean that we equip productions for AHL-process nets with nested application conditions for AHL-nets. Considering the fact that injective AHL-morphisms reflect the property of being an AHL-process net (Lemma A.7.1), this means that an application condition of the form $\exists(a, ac_C)$ with $a : L \to C$ is only satisfied by a match $m : L \to K$ into an AHL-process net K, if also C is an AHL-process net. Effectively, we obtain that an AHL-net that is not an AHL-process net that occurs somewhere in the nesting of a condition, implies that the corresponding component of the condition is never satisfied. Hence, it would make sense to restrict application conditions for AHL-process nets to be built up only over AHL-process nets. Considering the fact that the general framework for the analysis of $\mathcal{M}$-adhesive transformation systems with nested application conditions involves structuring techniques like the shifting of conditions over morphisms or productions, a satisfactory approach for the analysis of AHL-process net transformations with nested application conditions should provide corresponding results for these structuring techniques with the requirement that they result only in proper application conditions that contain only AHL-process nets.

Another more subtle problem concerning also nested application conditions of AHL-nets and instantiations is caused by the fact that all morphisms in the class $\mathcal{M}_{AHL}$ have an isomorphic data type part. Considering the morphism q in Figure 8.3 that has to be in $\mathcal{M}_{AHL}$ in the case of AHL-nets or instantiations, we obtain that all nets that occur in the nesting of an application condition must have the same data type part as a matched net, in order to satisfy the application condition. Especially in the case of productions for instantiations this would mean a serious restriction, because it means that effectively we cannot specify satisfiable nested application conditions for abstract productions which, in general, have a data type part that is different from the data type part of the concrete instantiations the productions are applied to (see Definition 4.6.1).

A similar problem occurs in the context of AHL-nets with individual tokens (AHLI-nets), as pointed out in [Mod12], leading to the notions of variable application conditions (VACs) and structural satisfaction. A variable application condition contains only AHLI-nets with a term algebra as data type part, similar to abstract instantiations which also are required to have a term algebra as data type part. Then, structural satisfaction of a VAC is defined analogously to the classical satisfaction, but with the difference that for conditions of the form $\exists(a, ac_C)$ the required morphism q as in Figure 8.3 does not have to be an $\mathcal{M}$-morphism, but can be *almost injective*, meaning that only the structural part needs to be injective, and it can have an arbitrary data type part. Using a mechanism called *data shifting*, the structural satisfaction of variable applications is successfully related in [Mod12] to the classical satisfaction of nested application conditions, which allows to transfer results for the classical satisfaction also to the special case of structural satisfaction.

The data shifting construction is based on a *data-structure factorisation* construction $f = f^S \circ f^D$ of AHL-morphisms f, and a *data extension* f^{+h} of AHL-morphisms f along Σ-homomorphisms h. It can be shown that the data extension f^{+h} of an AHL-morphism $f : ANI_1 \to ANI_2$ along homomorphism h is a $Data_{AHLI}$-cocreation of h via ANI_1 along functor $Data_{AHLI} : \mathbf{AHLINets} \to \mathbf{Alg}(\mathbf{\Sigma})$ that projects AHLI-nets and -morphisms to their data type part. Moreover, it can be shown that for a data-structure factorisation $f = f^S \circ f^D$ of an AHLI-morphism $f : ANI_1 \to ANI_2$, we have that f^D is the $Data_{AHLI}$-cocreation of $Data_{AHLI}(f)$ via ANI_1 and f^S is induced by the universal cocreation-property of f^D. Accordingly, the data shifting of an AHLI-morphism $a : P \to C$ is a pushout created by $Data_{AHLI}$-cocreations (Corollary A.1.33).

In fact, the introduction of functor cocreations in this thesis is strongly inspired by the idea of generalising this approach. So, it should be possible to lift the theory of variable application conditions and their structural satisfaction from the concrete case of AHLI-nets to a general theory based on functor cocreations. Considering the fact that we have $Data$-, $InstData$- and $wInstData$-cocreations, we already have everything that we need to instantiate this general approach again for nested application conditions for AHL-nets and (weak) instantiations.

An alternative approach in [HCE12] instantiates rules and negative application conditions by componentwise extremal $\mathcal{E}$-$\mathcal{M}$ factorisations. This may also be a promising approach for a theory of application conditions with abstract data type, since the category **AHLNets** has extremal $\mathcal{E}$-$\mathcal{M}_{AHL}$ factorisations, as we have shown in Corollary A.4.9. Constructions for extremal $\mathcal{E}$-$\mathcal{M}_{AHL}$ factorisations in the categories of (weak) instantiations should work analogously. Otherwise, we can obtain constructions for finite (weak) instantiations, using the general results for finitary $\mathcal{M}$-adhesive categories in [BEGG10, GBEG12].

8.4.3 Analysis of Safety and Security Problems

In this thesis we concentrate mostly on conflicts and compatibility of evolutions of platforms as well as scenarios. For these problems, it is sufficient that we have a quite high-level perspective on the platforms, where we do not concentrate on the low-level technical details. Other interesting problems concerning for instance safety properties, like the persistence of important resources, or security properties, like compliance to access restrictions, should require to move the focus away from the high-level perspective to the actual technical circumstances, like properties of the used hardware, or the communication protocol.

This seems to be particularly important in the course of current events, regarding the news about ubiquitous surveillance activities by intelligence services, like the US-American NSA and British GCHQ. Considering the possibility to compromise the security of proprietary encryption techniques by secret built-in back doors, it can be expected that in the

near future there will emerge an increasing demand for open software that supports secure communication. Due to the increasing complexity of communication platforms as well the corresponding security measures, the openness of the software alone may not be sufficient to fully comprehend its functionality, and accordingly, it is not sufficient to verify whether it is really secure. It remains an open question for future investigation, to which extent our modelling framework can support the analysis of safety and security properties of communication platforms and the underlying techniques.

8.4.4 Other Case Studies and Application Domains

In this thesis we developed a comprehensive framework for the modelling and analysis of communication platforms and scenarios, and we successfully applied it to our Apache Wave case study (see Section 2.2). Indeed, in order to avoid unnecessary complications that could impair the overview and understanding, we restricted the case study to cases of quite basic platforms, like the one modelled in Example 3.1.9. For instance, an actual wave has a tree-structure and may contain complex data elements like images, tables, maps, and so on, while in our modelling examples, we restricted waves to contain only plain text. So, for a more realistic model, it would be necessary to extend our platform models by a more complex structure of waves, and more sophisticated resources and actions that are provided by the platform.

It is an interesting question which additional modelling requirements may emerge that are necessary in order to support the comprehensive modelling of all features. This is also a good motivation for the investigation of other case studies in the future. In principal, it should be possible to model also the other examples of communication platforms discussed in Section 2.1, since our modelling approach is not in any way restricted by assumptions that are based on our case study: Our concept for the modelling of communication platforms (see Concept 3.1.8) is only based on the assumption that a communication platform consists of resources, data types, and actions that use these resources and data types, but we do not require that the resources, data types, or actions are of any peculiar form. Thus, the modelling of other communication platforms can be done by modelling an AHL-net that consists of the resources, data types, and actions that are provided by that platform. Moreover, our concepts for the modelling of scenarios (see Concept 4.1.7 for abstract scenarios, and Concept 4.3.6 for the semi-concrete and concrete scenarios) is only based on the assumption that a scenario contains occurrences of resources and actions that are used throughout the scenario, and in in the concrete case the resources have specific data values. In our Apache Wave case study, the scenarios model interactions of users and robots in waves. In the case of Wiki-systems, the scenarios may model the interactions of editors on wiki pages, and in the case of Facebook, the scenarios may model the sharing and commenting of posts on the user's timelines. A suitable modelling of the news feed in Facebook, which aggregates posts of different timelines, may require an extension of the concept of views, considering the fact that the Facebook news feed usually does not contain the entirety of current posts, but instead it is based on algorithms that supposedly try to filter out posts that are less interesting or not important for the user. However, following the modelling concepts presented in this thesis, it should in principal be possible to model the main aspects of most communication platforms and their scenarios without the necessity to extend our modelling approach.

A
Appendix

A.1 Category Theoretical Basics

In this section we give an overview on some of the category theoretical concepts used in this thesis. Category theory is a relatively young branch of mathematics that was initiated in the 1940s [EM45]. It is an abstract framework that considers objects and relations (morphisms) between them without regard to their internal details. In Subsection A.1.1 we recall some of the definitions from category theory that are of importance for this thesis, but for a comprehensive introduction to the category theory, we refer to [AM75, AHS90, BW90].

A.1.1 Basics

In this subsection, we review the definitions of some of the category theoretical concepts that are used in this thesis. The following notion of monomorphisms corresponds to that of injective functions in the category **Sets** of sets, and to injective morphisms in the Petri net categories **PTNets** and **AHLNets**. By reversing the direction of all arrows (morphisms) in the definition, we obtain the dual notion of epimorphisms.

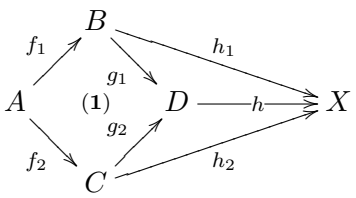
Definition A.1.1 (Monomorphism)**.** Given a morphism $f : A \to B$ in a category $\mathbf{C}$. Then f is called a *monomorphism*, if for all morphisms $g, h : C \to A$ in $\mathbf{C}$ with $f \circ g = f \circ h$ we have $g = h$. △

$$C \underset{h}{\overset{g}{\rightrightarrows}} A \xrightarrow{f} B$$

The following definition of pushouts is used for gluing constructions in the categories of sets and AHL-nets in the main part of this thesis.

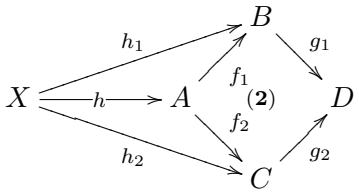
Definition A.1.2 (Pushout)**.** Given diagram (1) below in a category $\mathbf{C}$, then (1) is a *pushout* if (1) commutes and has the following universal property: For all objects X and morphisms $h_1 : B \to X$, $h_2 : C \to X$ with $h_1 \circ f_1 = h_2 \circ f_2$ in $\mathbf{C}$, there exists a unique $h : D \to X$ with $h \circ g_1 = h_1$ and $h \circ g_2 = h_2$.

If (1) is a pushout, we also say that D is a pushout of B and C over the interface A. Also, we call C a *pushout complement* of f_1 and g_1, and B a pushout complement of f_2 and g_2. △



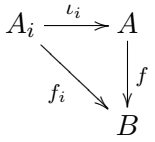
The definition of pullbacks is dual to the definition of pushouts. In the main part of this thesis, pullback constructions correspond mainly to constructions of restrictions along morphisms.

Definition A.1.3 (Pullback)**.** Given diagram (2) below in a category $\mathbf{C}$, then (2) is a *pullback* if (2) commutes and has the following universal property: For all objects X and morphisms $h_1 : X \to B$, $h_2 : X \to C$ with $g_1 \circ h_1 = g_2 \circ h_2$ in $\mathbf{C}$, there exists a unique $h : X \to D$ with $f_1 \circ h = h_1$ and $f_2 \circ h = h_2$. △



The following definition has the meaning of a disjoint union in the category **Sets** of sets. In the main part of the thesis, coproduct constructions are mainly used for the definition of parallel productions, that can be seen as a disjoint union of productions.

Definition A.1.4 (Coproduct)**.** Given objects $(A_i)_{i\in\mathcal{I}}$ for some index set $\mathcal{I}$ in a category $\mathbf{C}$. For an object A and morphisms $(\iota_i : A_i \to A)_{i\in\mathcal{I}}$, we say that $(A, (\iota_i)_{i\in\mathcal{I}})$ is a *coproduct* of $(A_i)_{i\in\mathcal{I}}$, written $A = \coprod_{i\in\mathcal{I}} A_i$, if for all $(f_i : A_i \to B)_{i\in\mathcal{I}}$ there exists a unique $f : A \to B$ such that $f \circ \iota_i = f_i$ for all $i \in \mathcal{I}$. △



A.1.2 Indexed Categories and Grothendieck Categories

In this subsection we review the notions of indexed categories and Grothendieck categories [Gro57, Joh02], and we introduce the category of signature sorted sets as an example of a Grothendieck category that is used for the families of variables in AHL-nets (see Definition 3.1.4 and Definition 3.1.5).

Definition A.1.5 (Indexed Category)**.** Given a category $\mathbf{I}$, called index category, an *indexed category* is a functor $F : \mathbf{I}^{\mathbf{op}} \to \mathbf{Cat}$, where $\mathbf{Cat}$ denotes the category of all categories. △

Definition A.1.6 (Grothendieck Category)**.** The *Grothendieck category* $\mathbf{Gr}(\mathbf{F})$ of an indexed category $F : \mathbf{I}^{\mathbf{op}} \to \mathbf{Cat}$ has as objects pairs (i, A) with $i \in \mathbf{I}$ and $A \in F(i)$. A morphism $(i_1, A_1) \to (i_2, A_2)$ is a pair (f, g) with $f : i_1 \to i_2$ in $\mathbf{I}$ and $g : A_1 \to F(f)(A_2)$ in $F(i)$.

Given morphisms $(f, g) : (i_1, A_1) \to (i_2, A_2)$ and $(f', g') : (i_2, A_2) \to (i_3, A_3)$, the composition is defined by

$$(f', g') \circ (f, g) = (f' \circ f, F(f)(g') \circ g)$$

For an object (i, A), the identity $id_{(i,A)}$ is given by (id_i, id_A). △

Example A.1.7 (Grothendieck Categories)**.** One example of a Grothendieck category is given by the category **Algs** of algebras and generalised algebra homomorphisms, where the indexed category is the contravariant functor $V : \mathbf{Sigs}^{op} \to \mathbf{Cat}$ with $V(\Sigma) = \mathbf{Alg}(\mathbf{\Sigma})$ and $V(f : \Sigma_1 \to \Sigma_2) = V_f : \mathbf{Alg}(\mathbf{\Sigma_2}) \to \mathbf{Alg}(\mathbf{\Sigma_1})$ being the forgetful functor that provides a Σ_1-reduct of a Σ_2-algebra (see [EM85]).

The objects of the category are pairs (Σ, A) of a signature Σ together with a Σ-algebra A. The morphisms of the category are generalised algebra homomorphisms $(f_\Sigma, f_A) : (\Sigma_1, A_1) \to (\Sigma_2, A_2)$, where $f_\Sigma : \Sigma_1 \to \Sigma_2$ is a signature morphism and $f_A : A_1 \to V_{f_\Sigma}(A_2)$ is a Σ_1-homomorphism. For more details we refer to [EM85].

Another example is given by the category **SSets** of many-sorted sets (S, X) where S is an (index) set and $X = (X_s)_{s \in S}$ is a family of sets indexed over S. The index category is $SSet : \mathbf{Sets^{op}} \to \mathbf{Cat}$ with $SSet(S) = \mathbf{Sets}^S$ (the category of S-indexed sets), and $SSet(f : S_1 \to S_2) = V_f : \mathbf{Sets}^{S_2} \to \mathbf{Sets}^{S_1}$ is the forgetful functor for families of sets, working analogously to the reduct of algebras on the respective carrier sets. That is, for $\mathbf{Sets}^{S_2}$-objects $X = (X_s)_{s \in S_2}$ we have $V_f(X) = (X_{f(s)})_{s \in S_1}$. For $\mathbf{Sets}^{S_2}$-morphisms $g : X_1 \to X_2$ with $(g_s : X_{1,s} \to X_{2,s})_{s \in S_2}$ we have $V_f(g) = (g_{f(s)} : X_{1,f(s)} \to X_{2,f(s)})_{s \in S_1}$. For more details on many-sorted sets, we refer to [TBG91]. ♢

Since every signature consists of a set of sorts, and every signature morphism contains a sort component function, it is possible to define analogously to the category **SSets** of many-sorted sets indexed over sets, a category **SigSets** of many-sorted sets indexed over signatures.

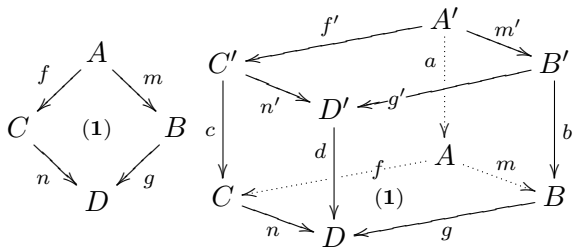
Definition A.1.8 (Signature-Sorted Sets)**.** The category **SigSets** of signature-sorted sets is defined as Grothendieck category over the index category $SigSet : \mathbf{Sigs^{op}} \to \mathbf{Cat}$ with $SigSet(\Sigma) = SSet(S)$ for objects $\Sigma = (S, OP)$, and $SigSet(f_\Sigma) = SSet(f_S)$ for morphisms $f_\Sigma = (f_S, f_{OP}) : \Sigma_1 \to \Sigma_2$ (see Example A.1.7 above for the definition of $SSet : \mathbf{Sets^{op}} \to \mathbf{Cat}$). △

A.1.3 $\mathcal{M}$-Adhesive Categories

An $\mathcal{M}$-adhesive category [EGH10] consists of a category $\mathbf{C}$ together with a class $\mathcal{M}$ of monomorphisms stable under pushouts and pullbacks as defined below. The concept of $\mathcal{M}$-adhesive categories generalizes that of adhesive [LS04], adhesive HLR [EHPP06], and weak adhesive HLR categories [EEPT06b].

Definition A.1.9 ($\mathcal{M}$-Adhesive Category)**.** An *$\mathcal{M}$-adhesive category* is a pair $(\mathbf{C}, \mathcal{M})$, where $\mathbf{C}$ is a category and $\mathcal{M}$ a class of monomorphisms with

1. $\mathcal{M}$ is closed under composition, decomposition and isomorphisms,
2. $\mathbf{C}$ has pushouts and pullbacks along $\mathcal{M}$-morphisms,
3. $\mathcal{M}$-morphisms are closed under pushouts and pullbacks, and
4. pushouts in $\mathbf{C}$ along $\mathcal{M}$ are $\mathcal{M}$-Van Kampen (VK) squares. A pushout (1) along $m \in \mathcal{M}$ is an $\mathcal{M}$-VK square if for any commutative cube (2) with (1) in the bottom and all vertical morphisms in $\mathcal{M}$, and where the back faces are pullbacks, the following statement holds: The top face is a pushout if and only if the front faces are pullbacks.



(2)

△

Remark A.1.10 (Adhesive and (Weak) Adhesive HLR Categories). The framework of $\mathcal{M}$-adhesive categories is a generalization of the framework of weak adhesive high-level replacement (HLR) categories [EEPT06b]. For weak adhesive HLR category $(\mathbf{C}, \mathcal{M})$, the Van Kampen (VK) square property has to hold for all pushouts along $m \in \mathcal{M}$ where either all vertical or all horizontal morphisms of the cube are in $\mathcal{M}$. In contrast, in an adhesive HLR category, the property has to hold for all cubes where only m is required to be an $\mathcal{M}$-morphism. All these definitions are based on the notion of adhesive categories [LS04] which is an adhesive HLR category where the class $\mathcal{M}$ contains all monomorphisms in $\mathbf{C}$. △

The most prominent examples of $\mathcal{M}$-adhesive categories are the category $(\mathbf{Sets}, \mathcal{M}_S)$ of sets and functions where the class $\mathcal{M}_S$ consists of all injective functions, and the category $(\mathbf{Graphs}, \mathcal{M}_G)$ of graphs and graph monomorphisms where the class $\mathcal{M}_G$ contains all injective graph morphisms. There are also suitable classes $\mathcal{M}$ for the Petri net categories used in this paper such that these categories are also $\mathcal{M}$-adhesive, as stated in the following fact:

Fact A.1.11 ($\mathcal{M}$-Adhesive Petri Net Categories)**.** *The categories* $(\mathbf{PTNets}, \mathcal{M}_{PT})$ *with* $\mathcal{M}_{PT}$ *being the class of all injective P/T morphisms, as well as* $(\mathbf{AHLNets}(\mathbf{\Sigma}, \mathbf{A}), \mathcal{M}_{AHL})$ *and* $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ *with the class* $\mathcal{M}_{AHL}$ *of injective AHL-morphisms with isomorphic data type part are* $\mathcal{M}$*-adhesive categories.*

Proof. In [Pra08] it is shown that the categories are weak adhesive HLR categories which implies that they are also $\mathcal{M}$-adhesive categories. □

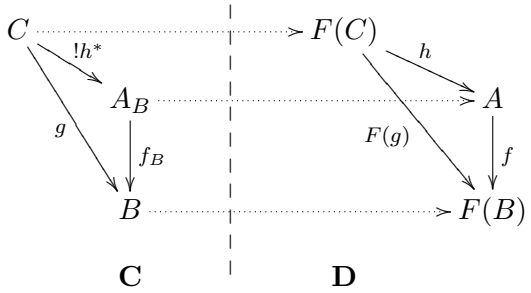
A.1.4 Functor Creations and Cocreations

In this subsection we introduce the notion of functor creations, and we show some very useful properties that we can use for various functors between categories of AHL-processes and instantiations. Note that our notion of functor creations is equivalent to the notion of cartesian (or prone) morphisms [Gro71, Gir71, Joh02], and thus, readers familiar with cartesian morphisms and their properties can skip most of this subsection. However, the notion of cartesian morphisms is mostly used in a purely mathematical context. Therefore, the theory of cartesian morphisms is often formulated in higher category theory and rarely motivated by examples from a computer scientific context. Moreover, as computer scientists we are interested rather in a constructive than a descriptive perspective. The notion of cartesian morphisms is rather a property of a morphism, whereas a functor creation is the result of a construction, yielding a cartesian morphism.

Definition A.1.12 (F-Creations)**.** Given a functor $F : \mathbf{C} \to \mathbf{D}$, objects B in $\mathbf{C}$ and A in $\mathbf{D}$, and a morphism $f : A \to F(B)$ in $\mathbf{D}$. A morphism $f_B : A_B \to B$ in $\mathbf{C}$ is called F*-creation* of f via B, if

- $F(A_B) = A$,

- $F(f_B) = f$, and
- *(universal property)* for all objects C with morphisms $g : C \to B$ in $\mathbf{C}$ and $h : F(C) \to A$ in $\mathbf{D}$ with $F(g) = f \circ h$ there exists a unique morphism $h^* : D \to A_B$ such that $F(h^*) = h$ and $f_B \circ h^* = g$.



We say that F *has creations* (or $\mathbf{C}$ *has* F*-creations*), if for every B in $\mathbf{C}$ and $f : A \to F(B)$ in $\mathbf{D}$ there is an F-creation of f via B. Given a class $\mathcal{D}$ of $\mathbf{D}$-morphism, we say that F has creations along $\mathcal{D}$, if for all $f : A \to F(B) \in \mathcal{D}$ there is an F-creation of f via B. △

Remark A.1.13. The notations A_B and f_B for the created object and morphism, respectively, is not mandatory, but we use it only to underline the fact that the construction depends on the object B. In the following we shall often use the notations $\bar{A}$ and $\bar{f}$ instead, because the object B is implicitly given by the codomain of the F-creation. Moreover, for a $\mathbf{C}$-morphism $f : A \to B$, if we just say that f is an F-creation then it implicitly means that f is an F-creation of its image $F(f)$ via its codomain B. △

The definition of F-creations is closely related to the definition of cartesian morphisms (also sometimes called prone morphisms) and Grothendieck fibrations [Gro71, Gir71, Joh02]. Therefore we recall the following definition of cartesian morphisms:

Definition A.1.14 (Cartesian Morphism)**.** Given a functor $F : \mathbf{C} \to \mathbf{D}$. A morphism $f : A \to B$ in $\mathbf{C}$ is *cartesian* (w. r. t. F) if for any morphisms $g : C \to B$ in $\mathbf{C}$ and $h : F(C) \to F(A)$ in $\mathbf{D}$ such that $F(f) \circ h = F(g)$, there exists a unique morphism $h^* : C \to A$ such that $f \circ h^* = g$ and $F(h^*) = h$. △

Fact A.1.15 (Characterization by Cartesian Morphism)**.** *Given a functor functor* $F : \mathbf{C} \to \mathbf{D}$*. A morphism* $f : A \to B$ *is an* F*-creation of* $F(f)$ *via* B *iff* f *is cartesian w. r. t.* F*.*

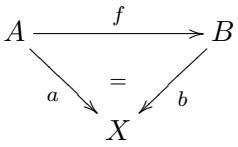
Proof. This follows directly from the definitions of F-creations (Definition A.1.12) and cartesian morphisms (Definition A.1.14). □

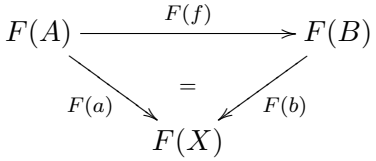
The definition of F-creations can also be reduced to the definition of cofree constructions, although the construction is not completely "free" as it depends on the $\mathbf{C}$-object B which is a necessary information for the "typing" of the construction. A categorical way to describe typed objects is the use of a slice category $\mathbf{C} \setminus X$ where $\mathbf{C}$ is a category, X is an object in $\mathbf{C}$ and $\mathbf{C} \setminus X$ consists of all $\mathbf{C}$-objects typed over X, and morphisms compatible with the typing. That is, the objects in $\mathbf{C} \setminus X$ are $\mathbf{C}$-morphisms $a : A \to X$, and a $\mathbf{C} \setminus X$-morphism f between objects $a : A \to X$ and $b : B \to X$ is a $\mathbf{C}$-morphism $f : A \to B$ such that $f \circ b = a$.

For every functor $F : \mathbf{C} \to \mathbf{D}$ and objects X in $\mathbf{C}$ we obtain a corresponding functor, mapping $\mathbf{C}$-objects typed over X to $\mathbf{D}$-objects typed over $F(X)$, called slice functor of F over X.

Definition A.1.16 (Slice Functor)**.** Given a functor $F : \mathbf{C} \to \mathbf{D}$ and an object X in $\mathbf{C}$. The *slice functor* $F \setminus X : \mathbf{C} \setminus X \to \mathbf{D} \setminus F(X)$ of F over X is defined by

- $F(a : A \to X) = F(a) : F(A) \to F(X)$ for objects $a : A \to X$, and
- $F(f : A \to B) = F(f) : F(A) \to F(B)$ for morphisms $f : A \to B$ in $\mathbf{C} \setminus X$.





△

Lemma A.1.17 (Composition of Slice Functors)**.** *Given functors $F : \mathbf{B} \to \mathbf{C}$ and $G : \mathbf{C} \to \mathbf{D}$. Then for every object X in $\mathbf{B}$ we have $(G \setminus F(X)) \circ (F \setminus X) = (G \circ F) \setminus X$.*

Proof. Note that F and G have types $F \setminus X : \mathbf{B} \setminus X \to \mathbf{C} \setminus F(X)$ and $G \setminus F(X) : \mathbf{C} \setminus F(X) \to \mathbf{D} \setminus G(F(X))$, and thus we have compatible types for the compositions $(G \setminus F(X)) \circ (F \setminus X), (G \circ F) \setminus X : \mathbf{B} \setminus X \to \mathbf{D} \setminus G(F(X))$.

Moreover, given an object $a : A \to X$ in $\mathbf{B}$, then by Definition A.1.16 we have

$$(G \setminus F(X)) \circ (F \setminus X)(a) = (G \setminus F(X)) \circ F(a) = G(a) = ((G \circ F) \setminus X)(a)$$

Analogously, given a morphism $f : A \to B$, we have

$$(G \setminus F(X)) \circ (F \setminus X)(f) = (G \setminus F(X)) \circ F(f) = G(f) = ((G \circ F) \setminus X)(f)$$

Hence, we have $(G \setminus F(X)) \circ (F \setminus X) = (G \circ F) \setminus X$. □

Fact A.1.18 (Characterization of F-Creations by Cofree Constructions)**.** *Given a functor $F : \mathbf{C} \to \mathbf{D}$, an object B in $\mathbf{C}$ and a morphism $f : A \to F(B)$ in $\mathbf{D}$. Then $\bar{f} : \bar{A} \to B$ is an F-creation of f via B if and only if $\bar{f}$ is a cofree construction with respect to functor $F \setminus B$ and counit id_A.*

Proof. **If.** Let $\bar{f} : \bar{A} \to B$ be a cofree construction of $f : A \to F(B)$ with respect to functor $F \setminus B$ and counit $id_A : A \to A$. We have to show that $\bar{f}$ is an F-creation of f via B. Let $g : C \to B$ in $\mathbf{C}$ and $h : F(C) \to A$ in $\mathbf{D}$ such that $f \circ h = F(g)$. Then we have that $g : C \to B$ is a $\mathbf{C} \setminus B$-object and that h is a $\mathbf{D}$-morphism $h : F(g) \to f$. Thus, by the fact that $\bar{f}$ is a cofree construction of f, there exists a unique $\mathbf{C} \setminus B$-morphism $h^* : g \to \bar{f}$ such that $id_A \circ F \setminus B(h^*) = h$. This means that we have a unique $\mathbf{C}$-morphism $h^* : C \to \bar{A}$ with $\bar{f} \circ h^* = g$ and $F(h^*) = F \setminus B(h^*) = id_A \circ F \setminus B(h^*) = h$. Hence, $\bar{f}$ is an F-creation of f via B.

Only If. Let $\bar{f} : \bar{A} \to B$ be an F-creation of f via B. We have to show that $\bar{f}$ is a cofree construction of f w. r. t. $F \setminus B$ and counit $id_A : A \to A$. Let $g : C \to B$ be an object in $\mathbf{C} \setminus B$ and $h : F(g) \to f$ a morphism in $\mathbf{D} \setminus F(B)$. Then we have that h is a $\mathbf{D}$-morphism $h : F(C) \to A$ with $f \circ h = F(g)$ which by F-creation $\bar{f}$ of f via B implies that there is a unique $\mathbf{C}$-morphism $h^* : C \to \bar{A}$ with $F(h^*) = h$ and $\bar{f} \circ h^* = g$. This means that h^* is a unique $\mathbf{C} \setminus B$-morphism $h^* : g \to \bar{f}$ with $id_A \circ F \setminus B(h^*) = F \setminus B(h^*) = F(h^*) = h$. Hence, $\bar{f}$ is a cofree construction of f w. r. t. functor $F \setminus B$ and counit id_A.

□

Corollary A.1.19 (Uniqueness of F-Creations)**.** *F-creations are unique up to isomorphism. That is, for a functor $F : \mathbf{C} \to \mathbf{D}$, an object B in $\mathbf{C}$ and a morphism $f : A \to F(B)$ in $\mathbf{D}$ with F-creation $\bar{f} : \bar{A} \to B$ of f via B, we have the following:*

1. *If there is a morphism $\bar{f}' : \bar{A}' \to B$ and an isomorphism $i : \bar{A} \to \bar{A}'$ such that $\bar{f}' \circ i = \bar{f}$, then also $\bar{f}'$ is an F-creation of f via B.*

2. *If there is an F-creation $\bar{f}' : \bar{A}' \to B$ of f via B, then there is an isomorphism $i : \bar{A} \to \bar{A}'$ such that $\bar{f}' \circ i = \bar{f}$.*

Proof. This follows directly from Fact A.1.18 and the uniqueness of cofree constructions. □

Corollary A.1.20 (Creations and Functor Composition)**.** *Given a functor $F : \mathbf{B} \to \mathbf{C}$ that has creations along a class $\mathcal{C}$ of $\mathbf{C}$-morphisms, and a functor $G : \mathbf{C} \to \mathbf{D}$ that has creations along $\mathcal{D}$ of $\mathbf{D}$-morphisms, where G-creations of $\mathcal{D}$-morphisms are $\mathcal{C}$-morphisms. Then also $G \circ F : \mathbf{B} \to \mathbf{D}$ has creations along $\mathcal{D}$.*

The $G \circ F$-creation $\bar{f}$ of a $\mathbf{D}$-morphism $f : A \to G \circ F(B) \in \mathcal{D}$ via $\mathbf{B}$-object B is given by F-creation of $\hat{f} : \hat{A} \to F(B)$ via B, where $\hat{f}$ is the G-creation of f via $F(B)$.

Proof. The construction of $\bar{f}$ given above is well-defined because the G-creation $\hat{f}$ of $f \in \mathcal{D}$ is a $\mathcal{C}$-morphism and F has creations along $\mathcal{C}$. Moreover, due to Fact A.1.18 we know that $\hat{f}$ is cofree construction w. r. t. slice functor $G \setminus F(B) : \mathbf{C} \setminus F(B) \to \mathbf{D} \setminus G(F(B))$, and that $\bar{f}$ is cofree construction w. r. t. slice functor $F \setminus B : \mathbf{B} \setminus B \to \mathbf{C} \setminus F(B)$. Thus, by composition of cofree constructions, we obtain that $\bar{f}$ is cofree construction w. r. t. slice functor $(G \circ F) \setminus B$ (see Lemma A.1.17). Hence, by Fact A.1.18 we have that $\bar{f}$ is $G \circ F$-creation. □

In the following definition we consider functors of the form $H : \mathbf{C}^{op} \times \mathbf{D}$, called *correspondence* from $\mathbf{C}$ to $\mathbf{D}$ in [Lur09]. The definition of a general connection category corresponds to the definition of the definition of the category $C *^H \mathbf{D}$ in [Lur09] (Section 2.3.1). The name of the category is in analogy to [Ehr71], where it is called a *Verbindungskategorie* (German for connection cateogory). The objects and morphisms of a general connection category of a functor $H : \mathbf{C}^{op} \times \mathbf{D}$ are the disjoint unions of objects and morphisms, respectively, in $\mathbf{C}$ and $\mathbf{D}$, but additionally the general connection category consists of morphisms between $\mathbf{C}$- and $\mathbf{D}$-objects given by the image of H.

Definition A.1.21 (General Connection Catgory)**.** Given a functor $H : \mathbf{C}^{op} \times \mathbf{D} \to \mathbf{Sets}$, the *connection category* $\mathbf{CC}(H)$ over H is defined by

- $Ob_{\mathbf{CC}(H)} = Ob_{\mathbf{C}} \uplus Ob_{\mathbf{D}}$ as objects,
- $Mor_{\mathbf{CC}(H)}(A, B) = Mor_{\mathbf{C}}(A, B)$ for objects A, B in $\mathbf{C}$,
- $Mor_{\mathbf{CC}(H)}(A, B) = Mor_{\mathbf{D}}(A, B)$ for objects A, B in $\mathbf{D}$,
- $Mor_{\mathbf{CC}(H)}(A, B) = H(A, B)$ for objects A in $\mathbf{C}$ and B in $\mathbf{D}$,
- $Mor_{\mathbf{CC}(H)}(A, B) = \emptyset$ for objects A in $\mathbf{D}$ and B in $\mathbf{C}$, and
- the composition is defined by composition in $\mathbf{C}$ and $\mathbf{D}$ for objects in $\mathbf{C}$ and $\mathbf{D}$, respectively, and given morphisms $f \in Mor_{\mathbf{C}}(A, B)$, $g \in H(B, C)$ and $h \in Mor_{\mathbf{D}}(C, D)$ the composition $h \circ g \circ f : A \to D$ is defined by

$$h \circ g \circ f = H(f, h)(g)$$

△

Given a functor $F : \mathbf{C} \to \mathbf{D}$, the Mor-functor (sometimes also called hom-functor) $Mor(F\_, \_)$ is a correspondence from $\mathbf{C}$ to $\mathbf{D}$ in the sense of [Lur09]. Accordingly, we define the connection category of F as the general connection category of $Mor(F\_, \_)$.

Definition A.1.22 (Connection Category)**.** Given a functor $F : \mathbf{C} \to \mathbf{D}$, we define

$$\mathbf{ConCat}(F) = \mathbf{CC}(Mor(F_{\_}, {}_{\_}))$$

where $Mor(F_{\_}, {}_{\_})$ is the hom-set functor $Mor(F_{\_}, {}_{\_}) : \mathbf{C}^{op} \to \mathbf{D}$, leading to the following choice of objects and morphisms:

- $Ob_{\mathbf{ConCat}(F)} = Ob_{\mathbf{C}} \uplus Ob_{\mathbf{D}}$ as objects,
- $Mor_{\mathbf{ConCat}(F)}(A, B) = Mor_{\mathbf{C}}(A, B)$ for objects A, B in $\mathbf{C}$,
- $Mor_{\mathbf{ConCat}(F)}(A, B) = Mor_{\mathbf{D}}(A, B)$ for objects A, B in $\mathbf{D}$,
- $Mor_{\mathbf{ConCat}(F)}(A, B) = Mor_{\mathbf{D}}(F(A), B)$ for objects A in $\mathbf{C}$ and B in $\mathbf{D}$,
- $Mor_{\mathbf{ConCat}(F)}(A, B) = \emptyset$ for objects A in $\mathbf{D}$ and B in $\mathbf{C}$, and
- the composition is defined by composition in $\mathbf{C}$ and $\mathbf{D}$ for objects in $\mathbf{C}$ and $\mathbf{D}$, respectively, and given morphisms $f \in Mor_{\mathbf{C}}(A, B)$, $g \in Mor(F_{\_}, {}_{\_})(B, C)$ and $h \in Mor_{\mathbf{D}}(C, D)$ the composition $h \circ g \circ f : A \to D$ is defined by

$$h \circ g \circ f = Mor(F_{\_}, {}_{\_})(f, h)(g) = Mor(F(f), h)(g) = h \circ g \circ F(f)$$

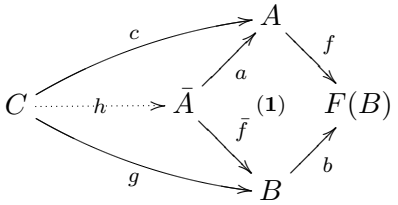
We also call $\mathbf{ConCat}(F)$ the *connection category* of F. △

Definition A.1.23 (F-Identity)**.** Given functor $F : \mathbf{C} \to \mathbf{D}$ and object C in $\mathbf{C}$. A morphism $f : C \to F(C)$ is called an F-identity if there is $f = id_{F(C)} \in Mor(F(C), F(C))$. Note that an F-identity is not an identity in the category $\mathbf{ConCat}(F)$. △

Fact A.1.24 (Characterization of F-Creations by Pullbacks)**.** *Given functor $F : \mathbf{C} \to \mathbf{D}$, an object B in $\mathbf{C}$ and morphism $f : A \to F(B)$ in $\mathbf{D}$. Then $\bar{f} : \bar{A} \to B$ is an F-creation of f via B if and only if diagram (1) below is a pullback in $\mathbf{ConCat}(F)$ where a and b are F-identities.*

$$\begin{array}{ccc} \bar{A} & \xrightarrow{a} & A \\ {\scriptstyle \bar{f}}\downarrow & (\mathbf{1}) & \downarrow{\scriptstyle f} \\ B & \xrightarrow[b]{} & F(B) \end{array}$$

Proof. **If.** Let $\bar{f} : \bar{A} \to B$ be an F-creation of f via B. Then we have that $F(\bar{A}) = A$ and $F(\bar{f}) = f$. So there are morphisms $a : \bar{A} \to A$ and $b : B \to F(B)$ being F-identities and we have $b \circ \bar{f} = id_{F(B)} \circ F(\bar{f}) = f = f \circ id_A = f \circ a$ which means that the diagram (1) above exists in $\mathbf{ConCat}(F)$ and it commutes. We have to show that it also satisfies the universal property of pullbacks.



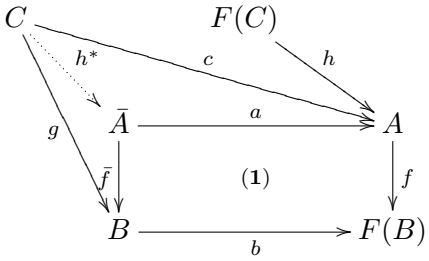
So let C in $\mathbf{ConCat}(F)$ together with morphisms $c : C \to A$ and $g : C \to B$ such that $f \circ c = b \circ g$. Then from the fact that B is in $\mathbf{C}$ and $Mor_{\mathbf{ConCat}(F)}(D, B) = \emptyset$ for all D

in **D**, it follows that C is a **C**-object, and thus g is a **C**-morphism. Moreover, morphism $c : C \to A$ with C in **C** and A in **D** corresponds to a morphism $c^* : F(C) \to A$ with

$$f \circ c^* = f \circ c = b \circ g = id_{F(B)} \circ F(g) = F(g)$$

Hence, the universal property of F-creation $\bar{f}$ implies a unique morphism $h : C \to \bar{A}$ such that $F(h) = c^*$ and $\bar{f} \circ h = g$. Then $F(h)$ is the required unique morphism because $a \circ h = id_A \circ F(h) = id_A \circ c^* = c$.

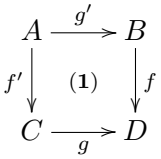
Only If. Let (1) be a pullback in $\mathbf{ConCat}(F)$ with a and b being F-identities, meaning that $F(\bar{A}) = A$ and $F(\bar{f}) = id_{F(B)} \circ F(\bar{f}) = b \circ \bar{f} = f \circ a = f \circ id_A = f$.



Now, let C in **C** together with morphisms $g : C \to B$ in **C** and $h : F(C) \to A$ in **D** such that $f \circ h = F(g)$. Then given by $h : F(C) \to A$ we also have a morphism $c : C \to A$ in $\mathbf{ConCat}(F)$ with $c = h$. Thus, we have $f \circ c = f \circ h = F(g) = id_{F(B)} \circ F(g) = b \circ g$ which by pullback (1) implies a unique $h^* : C \to \bar{A}$ in $\mathbf{ConCat}(F)$ such that $a \circ h^* = c$ and $\bar{f} \circ h^* = g$. Due to the fact that the codomain $\bar{A}$ of h^* is a **C**-object, we obtain that h^* is a **C**-morphism. Hence, $h^* : C \to \bar{A}$ is the required unique morphism because $a \circ h^* = c$ means $F(h^*) = id_A \circ F(h^*) = a \circ h^* = c = h$.

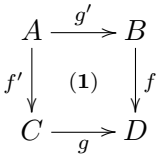
□

Lemma A.1.25 (Target Pullbacks are Pullbacks in Connection Category)**.** *Given a functor $F : \mathbf{C} \to \mathbf{D}$ and pullback (1) below in* **D***. Then (1) is also a pullback in* $\mathbf{ConCat}(F)$*.*



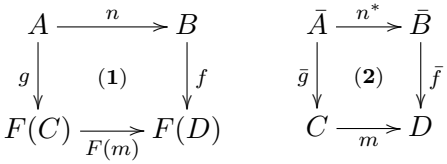
Proof. The universal pullback property naturally holds for all **D**-objects, but it remains to show that it holds also for **C**-objects. So let E in **C** with $k : E \to B$, $h : E \to C$ such that $f \circ k = g \circ h$. This means that we have **D**-morphisms $k : F(E) \to B$, $h : F(E) \to C$ with $f \circ k = g \circ h$, implying a unique **D**-morphism $m : F(E) \to A$ such that $g' \circ m = k$ and $f' \circ m = h$ that can be interpreted as unique $\mathbf{ConCat}(F)$-morphism $m : E \to A$ with this property. Hence, (1) is a pullback in $\mathbf{ConCat}(F)$. □

Lemma A.1.26 (Pullbacks in Connection Category are Source Pullbacks)**.** *Given a functor $F : \mathbf{C} \to \mathbf{D}$ and pullback (1) below in* $\mathbf{ConCat}(F)$ *with all objects and morphisms in* **C***. Then (1) is also a pullback in* **C***.*

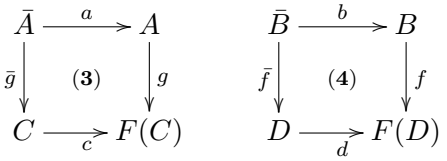


Proof. This follows immediately from the fact that (1) is a diagram in $\mathbf{C}$ and the unique induced morphism for all objects E and morphisms $k : E \to B$, $h : E \to C$ in $\mathbf{C}$ also is in $\mathbf{C}$, since $Mor_{\mathbf{ConCat}(F)}(E, A) = Mor_{\mathbf{C}}(E, A)$ for all E and A in $\mathbf{C}$. □

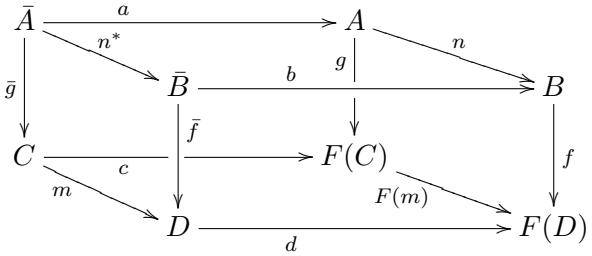
Fact A.1.27 (Creation of Pullbacks)**.** *Given a functor $F : \mathbf{C} \to \mathbf{D}$ that has creations along $\mathcal{D}$, morphism $m : C \to D$ in $\mathbf{C}$ and pullback (1) below in $\mathbf{D}$ with $f, g \in \mathcal{D}$. Then there exist F-creations $\bar{f} : \bar{B} \to D$ and $\bar{g} : \bar{A} \to C$ of f via D and g via C, respectively, and a morphism $n^* : \bar{A} \to \bar{B}$ with $F(n^*) = n$ in $\mathbf{C}$, such that (2) is a pullback in $\mathbf{C}$.*



Proof. Since F has creations along $\mathcal{D}$, and we have $f, g \in \mathcal{D}$, we can construct F-creations $\bar{f}$ and $\bar{g}$, leading to pullbacks (3) and (4) below in $\mathbf{ConCat}(F)$ by Fact A.1.24.



Then, morphisms $m \circ \bar{g} : \bar{A} \to D$ in $\mathbf{C}$ and $n : A \to B$ in $\mathbf{D}$ with $f \circ n = F(m) \circ g = F(m) \circ F(g) = F(m \circ g)$ due to F-creation $\bar{f}$ implies a unique morphism $n^* : \bar{A} \to \bar{B}$ such that $F(n^*) = n$ and diagram (2) above commutes. So we have the commuting cube in $\mathbf{ConCat}(F)$ below, where the right face is pullback (1) in $\mathbf{D}$, the left face is diagram (2) in $\mathbf{C}$, the back and front faces are pullbacks (3) and (4) in $\mathbf{ConCat}(F)$, respectively. Note that due to Lemma A.1.25 the pullback (1) in $\mathbf{D}$ is also a pullback in $\mathbf{ConCat}(F)$. So, by pullback composition we also have pullback (3)+(1) in the right back, and thus also pullback (2)+(4) in the front left of the commuting cube. Hence, by pullback decomposition of pullbacks (2)+(4) and (4), we obtain that (2) is a pullback in $\mathbf{ConCat}(F)$, and using the fact that all objects and morphisms in (2) are in $\mathbf{C}$, by Lemma A.1.26 we obtain that (2) is a pullback in $\mathbf{C}$.



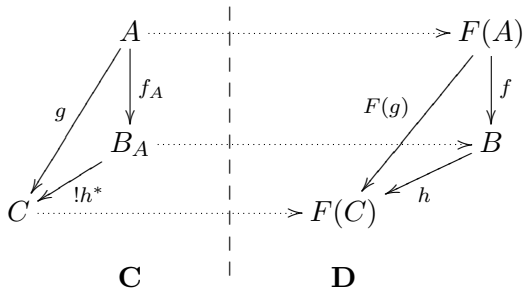
□

The creation along a functor $F : \mathbf{C} \to \mathbf{D}$ requires that there is a morphism $f : A \to F(B)$ in $\mathbf{D}$. Dually, we define the *cocreation* along functors that works in similar fashion for morphisms $f : F(B) \to A$.

Definition A.1.28 (F-Cocreations)**.** Given a functor $F : \mathbf{C} \to \mathbf{D}$, objects A in $\mathbf{C}$ and B in $\mathbf{D}$, and a morphism $f : F(A) \to B$ in $\mathbf{D}$. A morphism $f_A : A \to B_A$ in $\mathbf{C}$ is called *F-cocreation* of f via A, if

- $F(B_A) = B$,
- $F(f_A) = f$, and

- *(universal property)* for all objects C with morphisms $g : A \to C$ in $\mathbf{C}$ and $h : B \to F(C)$ in $\mathbf{D}$ with $F(g) = h \circ f$ there exists a unique morphism $h^* : B_A \to D$ such that $F(h^*) = h$ and $h^* \circ f_A = g$.



We say that F *has cocreations* (or $\mathbf{C}$ *has* F*-cocreations*), if for every A in $\mathbf{C}$ and $f : F(A) \to B$ in $\mathbf{D}$ there is an F-cocreation of f via A. Given a class $\mathcal{D}$ of $\mathbf{D}$-morphism, we say that F has cocreations along $\mathcal{D}$, if for all $f : F(A) \to B \in \mathcal{D}$ there is an F-cocreation of f via A. △

Fact A.1.29 (Duality of Creations and Cocreations)**.** *Given a functor $F : \mathbf{C} \to \mathbf{D}$, a $\mathbf{C}$-object A and a $\mathbf{D}$-morphism $f : F(A) \to B$. A morphism $\bar{f} : A \to \bar{B}$ is an F-cocreation of f via A iff $\bar{f}$ is an F^{op}-creation of $\mathbf{D}^{op}$-morphism $f : B \to F(A)$ via $\mathbf{D}^{op}$-object A, where $F^{op} : \mathbf{C}^{op} \to \mathbf{D}^{op}$ is the dual functor that maps objects and morphisms exactly as F does.*

Proof. It is easy to see that the definition of F-cocreations can be obtained from the definition of F-creations by inverting all arrows in the categories $\mathbf{C}$ and $\mathbf{D}$, and vice versa. □

Remark A.1.30 (Duality of Creations and Cocreations). Note that for every functor $F : \mathbf{C} \to \mathbf{D}$ we have $(F^{op})^{op} = F$, and thus, using Fact A.1.29 we can freely switch between the representation as F-creations and F-cocreations by inversion of all arrows in the categories $\mathbf{C}$ and $\mathbf{D}$. Due to this fact, all the results for F-creations above can be transferred to corresponding results for F-cocreations by dualisation of all used notions in both of the categories. △

In particular, this means that F-cocreations are unique up to isomorphism in the sense of Corollary A.1.19, since the notion of isomorphism is dual to itself.

Corollary A.1.31 (Uniqueness of F-Cocreations)**.** *F-cocreations are unique up to isomorphism. That is, for a functor $F : \mathbf{C} \to \mathbf{D}$, an object A in $\mathbf{C}$ and a morphism $f : F(A) \to B$ in $\mathbf{D}$ with F-cocreation $\bar{f} : A \to \bar{B}$ of f via A, we have the following:*

1. *If there is a morphism $\bar{f}' : A \to \bar{B}'$ and an isomorphism $i : \bar{B} \to \bar{B}'$ such that $\bar{f}' = i \circ \bar{f}$, then also $\bar{f}'$ is an F-cocreation of f via A.*
2. *If there is an F-cocreation $\bar{f}' : A \to \bar{B}'$ of f via A, then there is an isomorphism $i : \bar{B} \to \bar{B}'$ such that $\bar{f}' = i \circ \bar{f}$.*

Further, F-cocreations are closed under functor composition as shown for F-creations in Corollary A.1.20.

Corollary A.1.32 (Cocreations and Functor Composition)**.** *Given a functor $F : \mathbf{B} \to \mathbf{C}$ that has cocreations along a class $\mathcal{C}$ of $\mathbf{C}$-morphisms, and a functor $G : \mathbf{C} \to \mathbf{D}$ that has cocreations along $\mathcal{D}$ of $\mathbf{D}$-morphisms, where G-cocreations of $\mathcal{D}$-morphisms are $\mathcal{C}$-morphisms. Then also $G \circ F : \mathbf{B} \to \mathbf{D}$ has cocreations along $\mathcal{D}$.*

The $G \circ F$-cocreation $\bar{f}$ of a $\mathbf{D}$-morphism $f : G \circ F(A) \to B \in \mathcal{D}$ via $\mathbf{B}$-object A is given by F-cocreation of $\hat{f} : F(A) \to \hat{B}$ via A, where $\hat{f}$ is the G-cocreation of f via $F(A)$.

Moreover, due to duality of pushouts and pullbacks, we obtain the following fact for the creation of pushouts by dualisation of Fact A.1.27.

Corollary A.1.33 (Creation of Pushouts by F-Cocreations)**.** *Given a functor $F : \mathbf{C} \to \mathbf{D}$ that has cocreations along $\mathcal{D}$, morphism $m : C \to D$ in $\mathbf{C}$ and pushout (1) below in $\mathbf{D}$ with $f, g \in \mathcal{D}$. Then there exist F-cocreations $\bar{f} : B \to \bar{C}$ and $\bar{g} : A \to \bar{D}$ of f via A and g via B, respectively, and a morphism $n^* : \bar{C} \to \bar{D}$ with $F(n^*) = n$ in $\mathbf{C}$, such that (2) is a pushout in $\mathbf{C}$.*

$$\begin{array}{ccc} F(A) & \xrightarrow{F(m)} & F(B) \\ {\scriptstyle f}\downarrow & (1) & \downarrow{\scriptstyle g} \\ C & \xrightarrow[n]{} & D \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{m} & B \\ {\scriptstyle \bar{f}}\downarrow & (2) & \downarrow{\scriptstyle \bar{g}} \\ \bar{C} & \xrightarrow[n^*]{} & \bar{D} \end{array}$$

Definition A.1.34 (F-Shifting of Spans by Cocreation)**.** Given a functor $F : \mathbf{C} \to \mathbf{D}$ that has cocreations. Further, let be a span $L \xleftarrow{l} I \xrightarrow{r} R$ of $\mathbf{C}$-morphisms such that $F(l)$ and $F(r)$ are isomorphisms, and $m : L \to G$ an arbitrary $\mathbf{C}$-morphism. A span $L' \xleftarrow{l'} I' \xrightarrow{r'} R'$ of $\mathbf{C}$-morphisms together with a morphism $m' : L' \to G$ is called F-shifting of $L \xleftarrow{l} I \xrightarrow{r} R$ along m if

1. s_L, s_I and s_R are F-cocreations,
2. diagrams (1) and (2) in Figure A.1 are pushouts in $\mathbf{C}$,
3. diagram (3) in Figure A.1 commutes, and
4. $F(m')$ is an isomorphism

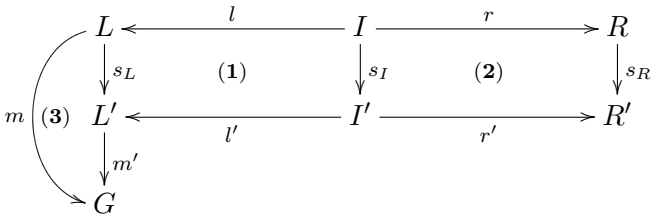


Figure A.1: F-shifting of span by cocreation

△

Fact A.1.35 (Existence of F-Shifting of Spans by Cocreation)**.** *Given a functor $F : \mathbf{C} \to \mathbf{D}$ that has cocreations. Further, let be a span $L \xleftarrow{l} I \xrightarrow{r} R$ of $\mathbf{C}$-morphisms such that $F(l)$ and $F(r)$ are isomorphisms, and $m : L \to G$ an arbitrary $\mathbf{C}$-morphism. Then the F-shifting $L' \xleftarrow{l'} I' \xrightarrow{r'} R'$ of $L \xleftarrow{l} I \xrightarrow{r} R$ along m exists.*

Proof. Due to the fact that $F(l)$ and $F(r)$ are isomorphisms, there exist inverse isomorphisms $F(l)^{-1}$ and $F(r)^{-1}$, and we have the following trivial pushouts (4) and (5) in $\mathbf{D}$:

$$\begin{array}{ccccc} F(L) & \xleftarrow{F(l)} & F(I) & \xrightarrow{F(r)} & F(R) \\ {\scriptstyle F(m)}\downarrow & (4) & \downarrow{\scriptstyle F(l)\circ F(m)} & (5) & \downarrow{\scriptstyle F(m)\circ F(l)\circ F(r)^{-1}} \\ F(G) & \xleftarrow[id]{} & F(G) & \xrightarrow[id]{} & F(G) \end{array}$$

So, given the fact that F has cocreations, by Corollary A.1.33 there exist pushouts (1) and (2) in $\mathbf{C}$ as shown in Figure A.1, where s_L, s_I and s_R are F-cocreations. Moreover, due to morphism $m : L \to G$ in $\mathbf{C}$ and $id_{F(G)}$ in $\mathbf{D}$, the F-cocreation s_L of $F(m)$ via L implies a unique morphism $m' : L' \to G$ such that diagram (3) in Figure A.1 commutes and $F(m') = id_{F(G)}$ is an isomorphism. Hence, $L' \overset{l'}{\leftarrow} I' \overset{r'}{\to} R'$ together with m' is an F-shifting of $L \overset{l}{\leftarrow} I \overset{r}{\to} R$ along m. □

A.1.5 Pullbacks of Categories

The categorical notion of pullbacks is used in this thesis mainly for the mathematical formalisation of restriction constructions for AHL-processes and instantiations in Section 4.5. Since it is possible to define a quasi-category **Cat** of all categories, where the objects are all categories and the morphisms are all functors,[36] it is also possible to apply the idea of the pullback construction also to pullbacks over co-spans of functors.

Definition A.1.36 (Pullback of Categories)**.** Given categories $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ and functors $F : \mathbf{B} \to \mathbf{D}$, $G : \mathbf{C} \to \mathbf{D}$. A category $\mathbf{A}$ together with functors $F' : \mathbf{A} \to \mathbf{C}$ and $G' : \mathbf{A} \to \mathbf{B}$ is a *pullback* of categories $\mathbf{B}$ and $\mathbf{C}$ along functors F and G, written $\mathbf{A} = \mathbf{B} \times_{F,G} \mathbf{C}$ or short $\mathbf{A} = \mathbf{B} \times_{\mathbf{D}} \mathbf{C}$, if it satisfies the universal property of pullbacks in the quasi-category **Cat** of categories: The diagram (PB) in Figure A.2 commutes, and for all categories $\mathbf{A}'$ and functors $H : \mathbf{A}' \to \mathbf{B}$, $K : \mathbf{A}' \to C$ with $F \circ H = G \circ K$ there exists a unique functor $M : \mathbf{A}' \to \mathbf{A}$ such that $G' \circ M = H$ and $F' \circ M = K$. △

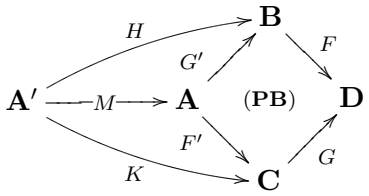


Figure A.2: Pullback of Categories

Remark A.1.37 (Strict Pullback of Categories). The definition of pullbacks of categories defined above can be seen as a definition for *strict pullbacks*, whereas non-strict pullbacks are defined using isomorphism instead of equalities, i. e. for the diagram (PB) in Figure A.2 it is only required that $G \circ F' \cong F \circ G'$ and analogously for a comparison object it is only required that $F \circ H \cong G \circ K$.

Obviously, the (strict) pullback of categories in the quasi-category **Cat** has the same properties as pullbacks in any other category, such as uniqueness up to isomorphism or composition and decomposition properties. △

Fact A.1.38 (Construction of Pullback of Categories)**.** *Given categories* $\mathbf{B}$*,* $\mathbf{C}$ *and* $\mathbf{D}$ *and functors* $F : \mathbf{B} \to \mathbf{D}$*,* $G : \mathbf{C} \to \mathbf{D}$*. The pullback* $\mathbf{A} = \mathbf{B} \times_{F,G} \mathbf{C}$ *can be constructed as subcategory* $\mathbf{A} \subseteq \mathbf{B} \times \mathbf{C}$ *such that* $\mathbf{A}$ *contains all objects* (B, C) *in* $\mathbf{B} \times \mathbf{C}$ *with* $F(B) = G(C)$*, and* $\mathbf{A}$ *contains all pairs of morphisms* (f, g) *with* $F(f) = G(g)$*. The functors* $G' : \mathbf{A} \to \mathbf{B}$ *and* $F' : \mathbf{A} \to \mathbf{C}$ *are the obvious projections.*

Proof. Let the category $\mathbf{A}$ and functors F' and G' be defined as above, for objects (A, B) in $\mathbf{A}$ we have $F(G'(A, B)) = F(A) = G(B) = G(F'(A, B))$ and analogously for morphisms.

[36] A category usually has a class of objects. The collection of all categories, however, is no proper class in the sense of axiomatic set theory, and therefore, **Cat** is not a proper category.

Further, let $\mathbf{A}'$ be a category and $H : \mathbf{A}' \to \mathbf{B}$, $K : \mathbf{A}' \to C$ functors with $F \circ H = G \circ K$. Then we define a functor $M : \mathbf{A}' \to \mathbf{A}$ with $M(X) = (H(X), K(X))$ for objects X in $\mathbf{A}'$ and $M(f) = (H(f), K(f))$ for morphisms f in $\mathbf{A}'$. This mapping is well-defined, since $F(H(X)) = G(K(X))$ implies $(H(X), K(X))$ in $\mathbf{A}$ and analogously for the morphisms. Moreover, it is a functor, because given morphisms $f : X \to Y$ and $g : Y \to Z$ in $\mathbf{A}'$, we have

$$\begin{aligned} M(g \circ f) &= (H(g \circ f), K(g \circ f)) = (H(g) \circ H(f), K(g) \circ K(f)) \\ &= (H(g), K(g)) \circ (H(f), K(f)) = M(g) \circ M(f) \end{aligned}$$

and given an object X in $\mathbf{A}'$, we have

$$M(id_X) = (H(id_X), K(id_X)) = (id_{H(X)}, id_{K(X)}) = id_{(H(X),K(X))} = id_{M(X)}$$

Furthermore, for objects X in $\mathbf{A}'$, we have

$$G' \circ M(X) = G'(H(X), K(X)) = H(X)$$

and analogously $F' \circ M(X) = K(X)$, and we obtain also the same commutativity for morphisms. It remains to show that M is unique, so let $M' : \mathbf{A}' \to \mathbf{A}$ be a functor such that $G' \circ M' = H$ and $F' \circ M' = K$. Then for an object X in $\mathbf{A}'$ and $M'(X) = (A, B)$, we have $A = G'(A, B) = G' \circ M'(X) = H(X)$ and $B = F'(A, B) = F' \circ M'(X) = K(X)$ and thus $M'(X) = (A, B) = (H(X), K(X)) = M(X)$. The proof for the morphism component of M works analogously, which means that M is unique and $\mathbf{A}'$ is pullback of $\mathbf{B}$ and $\mathbf{C}$ along F and G. □

Fact A.1.39 (Functors with Creations are Closed Under Pullbacks of Categories)**.** *Given a pullback (PB) of categories as shown in Figure A.2. If the functor F has creations along a class $\mathcal{D}$ of $\mathbf{D}$-morphisms, then F' has creations along a class $\mathcal{C}$ of $\mathbf{C}$-morphisms defined by $\mathcal{C} = G^{-1}(\mathcal{D})$.*

Assuming that $\mathbf{A}$ is constructed as in Fact A.1.38, the F'-creation $\bar{f}$ of $f : A \to F(B)$ in $\mathbf{C}$ via B in $\mathbf{A}$ can be obtained as $\bar{f} = (\hat{f}, f) : (\hat{A}, A) \to B$, where $\hat{f} : \hat{A} \to G'(B)$ is the F-creation of $G(f)$ via $G'(B)$.

Proof. Given pullback (PB) of categories as shown in Figure A.2, where F has creations along $\mathcal{D}$, we assume w. l. o. g. that the category $\mathbf{A}$ and functors F', G' are constructed as in Fact A.1.38.

We have to show that the functor F' has creations along $\mathcal{C} = G^{-1}(\mathcal{D})$. So let B be an object in $\mathbf{A}$ and $f : A \to F'(B) \in \mathcal{C}$ a morphism in $\mathbf{C}$. Then we have an object $G'(B)$ in $\mathbf{B}$ with $F(G'(B)) = G(F'(B))$ in $\mathbf{D}$, and thus $G(f) : G(A) \to F(G'(B)) \in \mathcal{C}$ which means that there is an F-creation $\hat{f} : \hat{A} \to G'(B)$ of $G(f)$ via $G'(B)$ in $\mathbf{B}$. Then we have $F(\hat{A}) = G(A)$ and $F(\hat{f}) = G(f)$. By construction of $\mathbf{A}$ this means that there is an object $(\hat{A}, A)$ and morphism $(\hat{f}, f) : (\hat{A}, A) \to B$ in $\mathbf{A}$, and we have $F'(\hat{A}, A) = A$ and $F'(\hat{f}, f) = f$.

In order to show that $\bar{f} = (\hat{f}, G(f))$ is an F'-creation, we have to show that it satisfies the universal property in Definition A.1.12. So let C be an object and $g : C \to B$ a morphism in $\mathbf{A}$, and $h : F'(C) \to A$ a morphism in $\mathbf{C}$ such that $f \circ h = F'(g)$. Then in $\mathbf{B}$, we have an object $G'(C)$ and morphism $G'(g) : G'(C) \to G'(B)$, and there is a morphism $G(h) : F(G'(C)) \to G(A)$ such that $G(f) \circ G(h) = G(f \circ h) = G(F'(g)) = F(G'(g))$. So the F-creation $\hat{f}$ of $G(f)$ via $G'(B)$ implies a unique morphism $h' : G'(C) \to \hat{A}$ such that $F(h') = G(h)$ and $\hat{f} \circ h' = G'(g)$.

Due to the construction of pullback (PB) as in see Fact A.1.38, there is $h^* = (h', h) : C \to (\hat{A}, A)$ in $\mathbf{A}$, and we have $F'(h^*) = F'(h', h) = h$ and $\bar{f} \circ h^* = (\hat{f}, f) \circ (h', h) = (\hat{f} \circ h', f \circ h) = (G'(g), F'(g)) = g$.

For the uniqueness, let $(k', k) : C \to (\hat{A}, A)$ be a morphism with $F'(k', k) = h$ and $\bar{f} \circ (k', k) = g$. Then we already have that $k = F'(k', k) = h$. Moreover, we have $F(k') = F(G'(k', k)) = G(F'(k', k)) = G(h)$ and

$$\hat{f} \circ k' = G'(\bar{f}) \circ G'(k', k) = G'(\bar{f} \circ (k', k)) = G'(g)$$

which by uniqueness of h' w. r. t. the universal property of F-creation $\hat{f}$ implies that $k' = k$ and thus $(k', k) = (h', h)$ which means that $h^* = (h', h)$ is unique. □

Corollary A.1.40 (Functors with Cocreations are Closed Under Pullbacks of Categories)**.** *Given a pullback (PB) of categories as shown in Figure A.2. If the functor F has cocreations along a class $\mathcal{D}$ of* **D***-morphisms, then F' has cocreations along a class $\mathcal{C}$ of* **C***-morphisms defined by $\mathcal{C} = G^{-1}(\mathcal{D})$.*

Assuming that **A** *is constructed as in Fact A.1.38, the F'-cocreation $\bar{f}$ of $f : F(A) \to B$ in* **C** *via A in* **A** *can be obtained as $\bar{f} = (\hat{f}, f) : A \to (\hat{B}, B)$, where $\hat{f} : G'(A) \to \hat{B}$ is the F-cocreation of $G(f)$ via $G'(A)$.*

Proof. According to Fact A.1.29 this follows from dualisation of Fact A.1.39. □

Fact A.1.41 (Pullbacks of Process and Instantiation Categories)**.** *The diagrams (1)-(3) in Figure A.3 are pullbacks of categories, where the vertical functors are inclusions, and we have the following projection functors:*

- $\pi_1^{\to} :$ **AHLNets**$^{\to} \to$ **AHLNets** *maps object $ob : AN_1 \to AN_2$ to AN_1 and morphism (f, g) to f,*
- $Net :$ **Inst** $\to$ **AHLNets** *maps object $(inst, AN)$ to AN and morphism f to f,*
- $\pi_1 :$ **Procs** $\to$ **AHLPNets** *maps object $mp : K \to AN$ to K and morphism (f, g) to f,*
- $PNet :$ **Inst** $\to$ **AHLPNets** *maps object $(inst, K)$ to K and morphism f to f,*
- $Proc :$ **ProcInst** $\to$ **Procs** *maps object $(inst, mp)$ to mp and morphism f to f.*

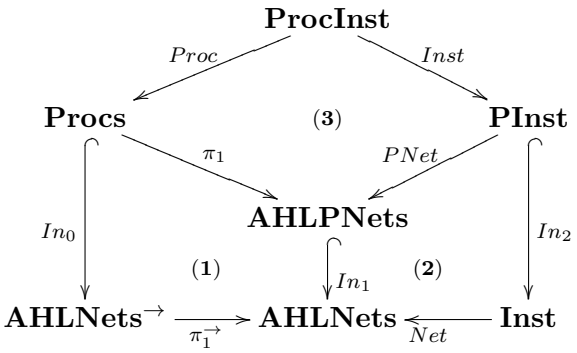


Figure A.3: Pullback diagrams of process and instantiation categories

Proof.

Pullback (1). According to Fact A.1.38, the pullback of **AHLNets**$^{\to}$ and **AHLPNets** along $\pi_1^{\to}$ and In can be constructed as subcategory $\mathbf{P_0} \subseteq$ **AHLNets**$^{\to} \times$ **AHLPNets** such that for objects $(ob : AN_1 \to AN_2, K)$ we have $AN_1 = K$ and for morphisms

$((f : AN_1 \to AN_1', g : AN_2 \to AN_2'), h : K \to K')$ we have $f = h$. So, we can define functors $I_1 : \mathbf{Procs} \to \mathbf{P_1}$ with $I_1(mp : K \to AN) = (mp, K)$ and $I_1(f, g) = ((f, g), f)$, and $J_1 : \mathbf{P_1} \to \mathbf{Procs}$ with $J_1(ob, K) = ob$ and $J_1((f, g), f) = (f, g)$. It is easy to see that I_1 and J_1 are well-defined and they establish an isomorphism of categories $\mathbf{P_1} \cong \mathbf{Procs}$. Thus, (1) is pullback.

Pullback (2). Using again Fact A.1.38, we obtain the pullback object as subcategory $\mathbf{P_2} \subseteq \mathbf{Inst} \times \mathbf{AHLPNets}$ such that for objects $((inst, AN), K)$ we have $AN = K$, and for morphisms (f, g) we have $f = g$. So obviously, we obtain inverse isomorphisms $I_2 : \mathbf{PInst} \to \mathbf{P_2}$ with $I_2(inst, K) = ((inst, K), K)$ and $I_2(f) = (f, f)$, and $J_2 : \mathbf{P_2} \to \mathbf{PInst}$ with $J_2((inst, K), K) = (inst, K)$ and $J_2(f, f) = f$. Hence, also (2) is a pullback.

Pullback (3). Again, by Fact A.1.38, we obtain the pullback object as subcategory $\mathbf{P_3} \subseteq \mathbf{PInst} \times \mathbf{Procs}$ such that for objects $((inst, K), mp : K' \to AN)$ we have $K = K'$, and for morphisms $(f, (g, h))$ we have $f = g$. Clearly, this means that there are inverse isomorphisms $I_3 : \mathbf{ProcInst} \to \mathbf{P_3}$ with $I_3(inst, mp : K \to AN) = ((inst, K), mp)$ and $I_3(f, g) = (f, (f, g))$, and $J_3 : \mathbf{P_3} \to \mathbf{ProcInst}$ with $J_3((inst, K), mp) = (inst, mp)$ and $J_3(f, (f, g)) = (f, g)$. Hence, also (3) is a pullback.

□

Corollary A.1.42 (Pullbacks of Process and Weak Instantiation Categories)**.** *The diagrams (4)-(7) in Figure A.4 are pullbacks of categories, where the functors and categories are the weak counterparts of the functors and categories, respectively, in Fact A.1.41.*

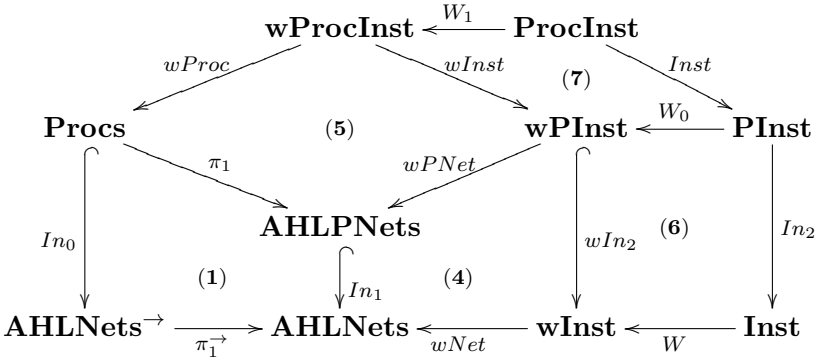


Figure A.4: Pullback diagrams of process and weak instantiation categories

Proof. Since all the properties of instantiations used in the proof of Fact A.1.41 do also hold for weak instantiations, we also have that the diagrams (4) and (5) in Figure A.4 are pullbacks of categories.

Moreover, consider the diagrams (6) and (7). Clearly, we have that the diagrams (6) and (7) commute, and thus due to pullback decomposition, they are also pullbacks. □

A.2 Initial Pushouts and Categorical Gluing Condition

Additionally to the set-theoretical gluing condition for AHL-nets in Definition 3.2.13, there is also a categorical way to characterize the existence of direct transformations, using initial pushouts as defined in [EEPT06b]:

Definition A.2.1 (Boundary, Initial Pushout)**.** Given a morphism $m : L \to G$ in an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$, a morphism $b : B \to L$ with $b \in \mathcal{M}$ is called the *boundary* over m if there is a pushout complement of m and b such that (1) is a pushout which is *initial* over m. Initiality of (1) over m means, that for every pushout (2) with $b' \in \mathcal{M}$ there exist unique morphisms $b^* : B \to D$ and $c^* : C \to E$ with $b^*, c^* \in \mathcal{M}$ such that $b' \circ b^* = b, c' \circ c^* = c$ and (3) is a pushout. B is then called the *boundary object* and c the *context* with respect to m. △

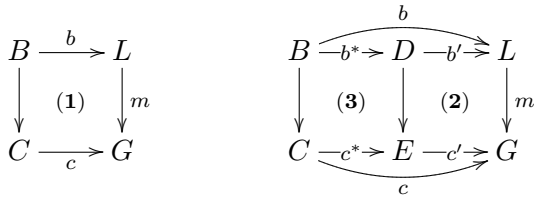


Figure A.5: Initial pushout

As shown in Fact A.1.11, the category $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ is $\mathcal{M}$-adhesive. Moreover, the category **AHLNets** has initial pushouts that can be constructed as pushout complement over so-called boundary, that is, the minimal AHL-net consisting of all identification and dangling points (see Definition 3.2.13).

Fact A.2.2 (Boundary and Initial Pushout in **AHLNets**)**.** *Given an AHL-morphism $m : L \to AN$, we define the boundary $b : B \hookrightarrow L$ of m as*

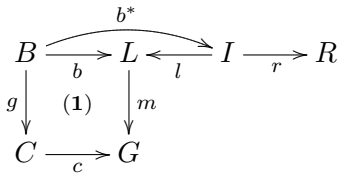
- $B = (\Sigma_B, P_B, T_B, pre_B, post_B, cond_B, type_B, A_B)$, *where*
- $(\Sigma_B, A_B) = (\Sigma_L, A_L)$,
- $P_B = DP \cup IP_P \cup \{p \in P_L \mid \exists t \in IP_T : p \in {\bullet t} \cup t^{\bullet}\}$ *with DP being the dangling points in Definition 3.2.13, and IP_P and IP_T being the sets of identification points restricted to places and transitions, respectively,*
- $T_B = IP_T$, *and*
- pre_B, $post_B$, $cond_B$ *and* $type_B$ *are restrictions of the corresponding functions in L.*

Then the context $c : C \hookrightarrow AN$ together with a morphism $B \to C$ can be obtained as pushout complement of b and m, and diagram (1) in Figure A.5 is an initial pushout.

Proof. This is shown in Fact 3.10 in [MGE+10] for AHL-nets with individual tokens (AHLI-nets), and AHL-nets can be considered as special AHLI-nets with an empty set I of individuals. □

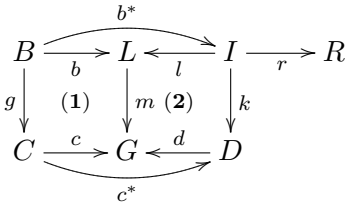
Using initial pushouts, it is possible to formulate a categorical gluing condition for $\mathcal{M}$-adhesive categories [EEPT06b], and it can be shown that this categorical gluing condition is also sufficient and necessary for the existence of pushout complement which are needed for the construction of direct transformations.

Definition A.2.3 (Categorical Gluing Condition)**.** Let $l : I \to L \in \mathcal{M}$ and $m : L \to G$ be morphisms in a given $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$ with initial pushouts.
We say that l and m satisfy the categorical gluing condition if for the initial pushout (1) over m there exists a morphism $b^* : B \to I$ such that $l \circ b^* = b$.



Given a production $\varrho = (L \xleftarrow{l} I \xrightarrow{r} R)$, we say ϱ and m satisfy the categorical gluing condition, iff l and m satisfy the categorical gluing condition. △

Fact A.2.4 (Categorical Gluing Condition)**.** *Given an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$ with initial pushouts, a match $m : L \to G$ satisfies the categorical gluing condition with respect to $l : K \to L \in \mathcal{M}$ (or a production $\varrho = (L \xleftarrow{l} I \xrightarrow{r} R)$, respectively) if and only if the context object D exists, i. e. there is a pushout complement (2) of l and m.*



If it exists, the context object D is unique up to isomorphism.

Proof. This is shown in Theorem 6.4 in [EEPT06b] for weak adhesive HLR categories, and the proof is also valid for $\mathcal{M}$-adhesive categories. □

Corollary A.2.5 (Equivalence of Gluing Conditions in **AHLNets**)**.** *Given a production $\varrho : L \xleftarrow{l} I \xrightarrow{r} R$ in* **AHLNets** *and a match $m : L \to AN$. Then ϱ and m satisfy the gluing condition for AHL-nets (Definition 3.2.13) if and only if it satisfies the categorical gluing condition for $\mathcal{M}$-adhesive categories (Definition A.2.3).*

Proof. This follows directly from the fact that both of the conditions are sufficient and necessary for the existence of a direct transformation $AN \overset{\varrho,m}{\Longrightarrow} AN'$, as shown in Fact 3.2.14 and Fact A.2.4. □

A.3 Disjoint Union and Parallel Productions

A special case of the gluing of different AHL-nets is the disjoint union, where no elements of the different nets are identified. Note that for sets $(S_i)_{i\in\mathcal{I}}$, the disjoint union S is defined as $S = \bigcup_{i\in\mathcal{I}}(S_i \times \{i\})$, which means that an element $s \in S_i$ for some $i \in \mathcal{I}$ is uniquely represented in S by an element (s, i). Furthermore, the disjoint union of sets is a coproduct in the category **Sets** . We define the disjoint union of AHL-nets not only for the binary but for the general case.

Definition A.3.1 (Disjoint Union of Algebraic High-Level Nets)**.** Given AHL-nets $(AN_i)_{i\in\mathcal{I}}$, the *disjoint union* AN of $(AN_i)_{i\in\mathcal{I}}$, written $AN = \biguplus_{i\in\mathcal{I}} AN_i$, is defined as AHL-net

$$AN = (\Sigma, P, T, pre, post, cond, type, A)$$

with

- $(\Sigma, A) = \coprod_{i\in\mathcal{I}}(\Sigma_i, A_i)$ is the coproduct of $(\Sigma_i, A_i)_{i\in\mathcal{I}}$ with injections $(\iota_{\Sigma,i}, \iota_{A,i}) : (\Sigma_i, A_i) \to (\Sigma, A)$ in **Algs**,
- $P = \coprod_{i\in\mathcal{I}} P_i$ is the coproduct of $(P_i)_{i\in\mathcal{I}}$ with injections $\iota_{P,i} : P_i \to P$ in **Sets**,

- $T = \coprod_{i\in\mathcal{I}} T_i$ is the coproduct of $(T_i)_{i\in\mathcal{I}}$ with injections $\iota_{T,i} : T_i \to T$ in **Sets**,
- $pre(t) = (\iota^{\#}_{\Sigma,i} \otimes \iota_{P,i})^{\oplus} \circ pre_i(t')$ for $t = \iota_{T,i}(t')$,
- $post(t) = (\iota^{\#}_{\Sigma,i} \otimes \iota_{P,i})^{\oplus} \circ post_i(t')$ for $t = \iota_{T,i}(t')$,
- $cond(t) = \mathcal{P}_{fin}(\iota^{\#}_{\Sigma,i}) \circ cond_i(t')$ for $t = \iota_{T,i}(t')$,
- $type(p) = \iota_{\Sigma,i} \circ type_i(p')$ for $p = \iota_{P,i}(p')$;

and inclusions $(\iota_i : AN_i \hookrightarrow AN)_{i\in\mathcal{I}}$ defined by $\iota_i = (\iota_{\Sigma,i}, \iota_{P,i}, \iota_{T,i}, \iota_{A,i})$. △

Well-definedness. The coproduct (Σ, A) with injections $(\iota_{\Sigma,i}, \iota_{A,i})_{i\in\mathcal{I}}$ in **Algs** exists because **Algs** is co-complete, as shown in [TBG91] (where the category **Algs** is called **Flat**(**ALG**)). Further, the coproducts P and T with injections $(\iota_{P,i})_{i\in\mathcal{I}}$ and $(\iota_{T,i})_{i\in\mathcal{I}}$ exist since also **Sets** has coproducts that can be constructed as disjoint union. Then by jointly surjectivity of injections $(\iota_{T,i})_{i\in\mathcal{I}}$, for every $t \in T$ there exists $i \in \mathcal{I}$ and $t' \in T_i$ such that $t = \iota_{T,i}(t')$, and t' is unique due to the fact that T is a disjoint union. Thus, the functions *pre*, *post* and *cond*, and analogously *type*, are well-defined. The well-definedness of AHL-morphisms ι_i follows directly from the definition of the *pre*, *post*, *cond* and *type* functions. □

Definition A.3.2 (Coproduct Compatible with $\mathcal{M}$)**.** Given an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$, then $\mathbf{C}$ *has coproducts compatible with* $\mathcal{M}$ if $\mathbf{C}$ has coproducts and, for all $\mathcal{M}$-morphisms $(f_i : A_i \to B_i)_{i\in\mathcal{I}}$, the coproduct morphism is also in $\mathcal{M}$, i. e. $\coprod_{i\in\mathcal{I}} f_i : \coprod_{i\in\mathcal{I}} A_i \to \coprod_{i\in\mathcal{I}} B_i \in \mathcal{M}$. △

Fact A.3.3 (Coproduct of Algebraic High-Level Nets)**.** *The category* **AHLNets** *has coproducts that can be constructed as disjoint union as defined in Definition A.3.1. Moreover, coproducts in* **AHLNets** *are compatible with the classes* $\mathcal{M}_{AHL}$ *of all monomorphisms with isomorphic data type part.*

Proof. Given AHL-nets $(AN_i)_{i\in\mathcal{I}}$, we can construct the disjoint union AN with injections $(\iota_i : AN_i \to AN)_{i\in\mathcal{I}}$ as defined in Definition A.3.1. We have to show that AN satisfies the universal property of coproducts. So let AN' be an AHL-net and $(f_i : AN_i \to AN')_{i\in\mathcal{I}}$ AHL-morphism. We define an AHL-morphism $f : AN \to AN' = (f_\Sigma, f_P, f_T, f_A)$ with (f_Σ, f_A), f_P, f_T being the unique morphisms induced by coproducts (Σ, A), P and T (see Definition A.3.1) and generalised homomorphisms $(f_{i,\Sigma}, f_{i,A})_{i\in\mathcal{I}}$, and functions $(f_{i,P})_{i\in\mathcal{I}}$, $(f_{i,T})_{i\in\mathcal{I}}$. We have to show that f is well-defined. So let $t \in T_{AN}$. As shown in the well-definedness proof of Definition A.3.1 there is an $i \in \mathcal{I}$ and $t' \in T_{AN_i}$ such that $\iota_{T,i}(t') = t$. Thus, using compositionality of $(\_ \otimes \_)^{\oplus}$ shown in the appendix of [MGE+10], we obtain

$$\begin{aligned}
(f^{\#}_{\Sigma} \otimes f_P)^{\oplus} \circ pre_{AN}(t) &= (f^{\#}_{\Sigma} \otimes f_P)^{\oplus} \circ (\iota^{\#}_{\Sigma,i} \otimes \iota_{P,i})^{\oplus} \circ pre_{AN_i}(t') \\
&= ((f_\Sigma \circ \iota_{\Sigma,i})^{\#} \otimes (f_P \circ \iota_{P,i}))^{\oplus} \circ pre_{AN_i}(t') \\
&= (f^{\#}_{i,\Sigma} \otimes f_{i,P})^{\oplus} \circ pre_{AN_i}(t') \\
&= pre_{AN'} \circ f_{i,T}(t') \\
&= pre_{AN'} \circ f_T \circ \iota_{T,i}(t') \\
&= pre_{AN'} \circ f_T(t)
\end{aligned}$$

The proof for compatibility of the *post* function works analogously. Moreover, for the *cond* function, using compositionality of $\mathcal{P}_{fin}(\_^{\#})$, shown in [MGE+10], we obtain

$$\begin{aligned}
\mathcal{P}_{fin}(f_\Sigma^\#) \circ cond_{AN}(t) &= \mathcal{P}_{fin}(f_\Sigma^\#) \circ \mathcal{P}_{fin}(\iota_{\Sigma,i}^\#) \circ cond_{AN_i}(t') \\
&= \mathcal{P}_{fin}((f_\Sigma \circ \iota_{\Sigma,i})^\#) \circ cond_{AN_i}(t') \\
&= \mathcal{P}_{fin}(f_{i,\Sigma}^\#) \circ cond_{AN_i}(t') \\
&= cond_{AN'} \circ f_{i,T}(t') \\
&= cond_{AN'} \circ f_T \circ \iota_{T,i}(t') \\
&= cond_{AN'} \circ f_T(t)
\end{aligned}$$

Finally, considering the *type* function, let $p \in P_{AN}$ and analogously to the transitions, we have $p' \in P_{AN_i}$ such that $\iota_{P,i}(p') = p$, and we have

$$\begin{aligned}
f_\Sigma \circ type_{AN}(p) &= f_\Sigma \circ \iota_{\Sigma,i} \circ type_{AN_i}(p') \\
&= f_{i,\Sigma} \circ type_{AN_i}(p') \\
&= type_{AN'} \circ f_{i,P}(p') \\
&= type_{AN'} \circ f_P \circ \iota_{i,P}(p') \\
&= type_{AN'} \circ f_P(p)
\end{aligned}$$

Hence, f is a well-defined AHL-morphism. The uniqueness of f and the required commutativity follows immediately from uniqueness and commutativity of the components.

It remains to show that coproducts in **AHLNets** are compatible with the class $\mathcal{M}_{AHL}$ of all monomorphisms with isomorphic data type part. Let $(f_i : AN_i \to AN_i')_{i\in\mathcal{I}}$ be a family of $\mathcal{M}'_{AHL}$-morphisms. The fact that the morphism $\coprod_{i\in\mathcal{I}} f_i$ is has injective P- and T-components follows from injectivity of the P- and T-components of all f_i, because due to the construction of disjoint unions we have the same injective mappings between the disjoint copies of places and transitions in the respective disjoint unions of AHL-nets. Moreover, the data type part of the disjoint union of AHL-nets is constructed as coproduct in **Algs**. Then $\mathcal{M}_{AHL}$-morphisms $f_i$imply that we have isomorphisms $f_{i,A} : A_i \xrightarrow{\sim} A_i'$ which due to the uniqueness of coproducts up to isomorphism means that we also have a corresponding isomorphism $\coprod_{i\in\mathcal{I}} f_i : \coprod_{i\in\mathcal{I}} A_i \xrightarrow{\sim} \coprod_{i\in\mathcal{I}} A_i'$. Hence, $\coprod_{i\in\mathcal{I}} f_i$ is an $\mathcal{M}_{AHL}$-morphism. □

Based on the disjoint union of AHL-nets, we define also the disjoint union of productions, called parallel production as in [EEPT06b]. As the name suggests, the application of a parallel production corresponds to parallel application of all its single parts.

Definition A.3.4 (Parallel Production for Algebraic High-Level Nets)**.** Given productions $(\varrho_i : L_i \xleftarrow{l_i} I_i \xrightarrow{r_i} R_i)_{i\in\mathcal{I}}$ for AHL-nets. The *parallel production* $\varrho^+ : L^+ \xleftarrow{l^+} I^+ \xrightarrow{r^+} R^+$ of $(\varrho_i : L_i \xleftarrow{l_i} I_i \xrightarrow{r_i} R_i)_{i\in\mathcal{I}}$, written $\varrho^+ = \biguplus_{i\in\mathcal{I}} \varrho_i$ or $\varrho^+ = \coprod_{i\in\mathcal{I}} \varrho_i$, is defined by the componentwise disjoint union $X^+ = \biguplus_{i\in\mathcal{I}} X_i$ for $X \in \{L, I, R\}$. The morphisms l^+ and r^+ are the unique morphisms induced by coproduct I^+ (see Fact A.3.3) and morphisms $\iota_i^L \circ l$ and $\iota_i^R \circ r$, respectively.

$$\begin{array}{ccccc}
L_i & \xleftarrow{\ l\ } & I_i & \xrightarrow{\ r\ } & R_i \\
{\scriptstyle \iota_i^L}\downarrow & & {\scriptstyle \iota_i^I}\downarrow & & {\scriptstyle \iota_i^R}\downarrow \\
L^+ & \xleftarrow[\ l^+\ ]{} & I^+ & \xrightarrow[\ r^+\ ]{} & R^+
\end{array}$$

△

Fact A.3.5 (Parallel Production is Production for AHL-Nets)**.** *Given productions* $(\varrho_i : L_i \overset{l_i}{\leftarrow} I_i \overset{r_i}{\rightarrow} R_i)_{i\in\mathcal{I}}$ *for AHL-nets. Then the parallel production* $\varrho^+ = \biguplus_{i\in\mathcal{I}} \varrho_i$ *is a production for AHL-nets.*

Proof. It is already shown above that the disjoint union of AHL-nets again leads to a well-defined AHL-net. The well-definedness of AHL-morphisms l^+ and r^+ follows from the fact that the disjoint union is a coproduct in **AHLNets** as shown in Fact A.3.3. It remains to show that the morphisms l^+ and r^+ are $\mathcal{M}_{AHL}$-morphisms which follows from the fact that coproducts in **AHLNets** are compatible with $\mathcal{M}_{AHL}$. □

In the following we show that the disjoint union of AHL-process nets leads to a coproduct in the category **AHLPNets**, and that the disjoint union of productions for AHL-process nets (i. e. the parallel production) again is a production for AHL-process nets.

Fact A.3.6 (Coproduct of AHL-Process Nets)**.** *The category* **AHLPNets** *has coproducts that can be constructed as disjoint union of AHL-nets as defined in Definition A.3.1.*

Proof. Given AHL-process nets $(K_i)_{i\in\mathcal{I}}$, we construct the disjoint union $K = \biguplus_{i\in\mathcal{I}} K_i$, and we have to show that K is a coproduct in **AHLPNets**. For this, it is necessary to show that K is an AHL-process net. Due to construction of the disjoint union in Definition A.3.1, the places and transitions of K are disjoint unions of the places and transitions, respectively, of the AHL-process nets K_i. Moreover, the pre and post conditions of a transition in K is directly inherited from its corresponding transition in one of the nets K_i for some $i \in \mathcal{I}$. Therefore, the net K satisfies the unarity and conflict-freeness conditions which only depend on the pre and post domain functions, since all the single nets K_i satisfy these conditions. Further, due to the disjoint construction, the causal relation $<_K$ is obtained as disjoined union of the irreflexive causal relations $<_{K_i}$, and thus, also $<_K$ is irreflexive and therefore a strict partial order. Hence, the disjoint union K is an AHL-process net.

According to Fact A.3.3, K is a coproduct of $(K_i)_{i\in\mathcal{I}}$ in **AHLNets**, and thus it is also a coproduct in the full subcategory **AHLPNets** $\subseteq$ **AHLNets**. □

Corollary A.3.7 (Parallel Production for AHL-Process nets)**.** *Given productions* $(\varrho_i : L_i \overset{l_i}{\leftarrow} I_i \overset{r_i}{\rightarrow} R_i)_{i\in\mathcal{I}}$ *for AHL-process nets. Then the parallel production* $\varrho^+ : L^+ \overset{l^+}{\leftarrow} I^+ \overset{r^+}{\rightarrow} R^+ = \uplus_{i\in\mathcal{I}}\varrho_i$ *as defined in Definition A.3.4 is a production for AHL-process nets.*

Proof. Using Fact A.3.6, we have that the componentwise coproducts L^+, I^+ and R^+ are AHL-process nets, and since **AHLPNets** is a full subcategory of **AHLNets**, $l^+ : I^+ \to L^+, r^+ : I^+ \to R^+ \in \mathcal{M}_{AHL}$ are **AHLPNets**-morphisms. □

Finally, we also show that the categories of (weak) instantiations have coproducts, allowing us to define also parallel productions for instantiations. Moreover, we show that the parallel production of an abstract production again is an abstract production, and that the satisfaction of the instantiation condition by the parallel production depends only on satisfaction by all its single components.

Fact A.3.8 (Coproduct of Instantiations)**.** *The categories* **Inst** *and* **wInst** *have coproducts that can be constructed by disjoint union of the underlying AHL-nets (see Definition A.3.1 and Fact A.3.3).*

Proof. We show the proof w. l. o. g. only for coproducts in **Inst**. Let $(inst_i, AN_i)_{i\in\mathcal{I}}$ be instantiations. We construct the coproduct $AN = \coprod_{i\in\mathcal{I}} AN_i$ according to Fact A.3.3, with injections $\iota_i : AN_i \to AN$. Then by Fact 3.4.4 we also have a corresponding coproduct

$Skel(AN) = \coprod_{i\in\mathcal{I}} Skel(AN_i)$. Thus, due to P/T-morphisms $Flat(\iota_i) \circ inst_i : Skel(AN_i) \to Flat(AN)$, the universal coproduct property implies a unique $inst : Skel(AN) \to Flat(AN)$ such that $inst \circ Skel(\iota_i) = Flat(\iota_i) \circ inst_i$.

Also, by universal coproduct property, the identity $id_{Skel(AN)}$ is the unique endomorphism $x : Skel(AN) \to Skel(AN)$ such that $x \circ Skel(\iota_i) = id_{Skel(AN)} \circ Skel(\iota_i)$ for all $i \in \mathcal{I}$. So due to the fact that $proj(AN) \circ inst \circ Skel(\iota_i) = proj(AN) \circ Flat(\iota_i) \circ inst_i = Skel(\iota_i) \circ proj(AN_i) \circ inst_i = Skel(\iota_i)$ for all $i \in \mathcal{I}$, it follows that $proj(AN) \circ inst = x = Skel(AN)$ which means that $(inst, AN)$ is an instantiation and the injections ι_i are instantiation morphisms.

It remains to show that $(inst, AN)$ satisfies the universal property of coproduct, but since all instantiation morphisms are also AHL-morphisms, the satisfaction of the universal property in **Inst** can easily be derived from the universal coproduct property of AN in **AHLNets**. □

Definition A.3.9 (Parallel Production for Instantiations)**.** Given productions $(\varrho_i : L_i \xleftarrow{l_i} I_i \xrightarrow{r_i} R_i)_{i\in\mathcal{I}}$ for instantiations. The *parallel production* $\varrho^+ : L^+ \xleftarrow{l^+} I^+ \xrightarrow{r^+} R^+$ of $(\varrho_i : L_i \xleftarrow{l_i} I_i \xrightarrow{r_i} R_i)_{i\in\mathcal{I}}$, written $\varrho^+ = \biguplus_{i\in\mathcal{I}} \varrho_i$ or $\varrho^+ = \coprod_{i\in\mathcal{I}} \varrho_i$, is defined by the componentwise disjoint union $X^+ = \biguplus_{i\in\mathcal{I}} X_i$ for $X \in \{L, I, R\}$. The morphisms l^+ and r^+ are the unique morphisms induced by coproduct I^+ (see Fact A.3.8) and morphisms $\iota_i^L \circ l$ and $\iota_i^R \circ r$, respectively.

$$\begin{array}{ccccc} L_i & \xleftarrow{l} & I_i & \xrightarrow{r} & R_i \\ {\scriptstyle \iota_i^L}\downarrow & & {\scriptstyle \iota_i^I}\downarrow & & {\scriptstyle \iota_i^R}\downarrow \\ L^+ & \xleftarrow[l^+]{} & I^+ & \xrightarrow[r^+]{} & R^+ \end{array}$$

△

Lemma A.3.10 (Parallel Production of Abstract Productions is Abstract)**.** *Given abstract productions for instantiations $(\varrho_i)_{i\in\mathcal{I}}$, the parallel production $\varrho = \coprod_{i\in\mathcal{I}} \varrho_i$ is an abstract production.*

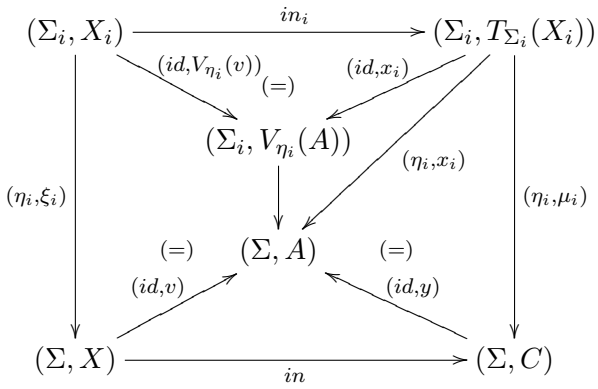
Proof. It suffices to show that the coproduct of term algebras $(\Sigma_i, T_{\Sigma_i}(X_i))$ again is a term algebra. In this proof we partly consider algebras as the corresponding signature-sorted sets (see Definition A.1.8) that are obtained by forgetting all operations and considering only the carrier sets.

We show that the coproduct $(\Sigma, C) = \coprod_{i\in\mathcal{I}}(\Sigma_i, T_{\Sigma_i}(X_i))$ together with coproduct injections $(\eta_i, \mu_i) : (\Sigma_i, T_{\Sigma_i}(X_i)) \to (\Sigma, C)$ is equal to the term algebra $(\Sigma, T_\Sigma(X))$, where X is obtained as coproduct $(\Sigma, X) = \coprod_{i\in\mathcal{I}}(\Sigma_i, X_i)$ with injections $(\eta_i, \xi_i) : (\Sigma_i, X_i) \to (\Sigma, X)$.

We do the proof by showing that algebra C is free over X in $\mathbf{Alg}(\boldsymbol{\Sigma})$.

Since all variables in X_i are also contained in $T_{\Sigma_i}(X_i)$, there are inclusions $in_i : (\Sigma_i, X_i) \to (\Sigma_i, T_\Sigma(X_i))$. Further, due to $(\eta_i, \mu_i) \circ in_i : (\Sigma_i, X_i) \to (\Sigma, C)$, the coproduct (Σ, X) implies a unique morphism $in : (\Sigma, X) \to (\Sigma, C)$ such that $(\eta_i, \mu_i) \circ in_i = in \circ (\eta_i, \xi_i)$, i. e. the outer diagram below commutes.

Now, let (Σ, A) be an arbitrary Σ-algebra, and $v : X \to A$ be an assignment. Then there are also assignments $V_{\eta_i}(v) : X_i \to V_{\eta_i}(A)$ which by freeness of $T_{\Sigma_i}(X_i)$ over X_i in $\mathbf{Alg}(\boldsymbol{\Sigma_i})$ implies unique Σ_i-homomorphisms $x_i : T_{\Sigma_i}(X_i) \to V_{\eta_i}(A)$ such that $x_i \circ in_i = V_{\eta_i}(v)$. Note that a Σ_i-homomorphism $x_i : T_{\Sigma_i}(X_i) \to V_{\eta_i}(A)$ can be interpreted as **Algs**-morphism $(\eta_i, x_i) : (\Sigma_i, T_{\Sigma_i}(X_i)) \to (\Sigma, A)$. So, the coproduct (Σ, C) implies that there is a unique **Algs**-morphism $(id, y) : (\Sigma, C) \to (\Sigma, A)$ such that $y \circ (\eta_i, \mu_i) = (\eta_i, x_i)$.

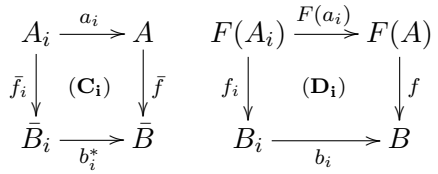


Using the commutativities in the diagram above, we obtain for every $i \in \mathcal{I}$:

$$\begin{aligned}(id, y) \circ in \circ (\eta_i, \xi_i) &= (id, y) \circ (\eta_i, \mu_i) \circ in_i \\ &= (\eta_i, x_i) \circ in_i \\ &= (id, x_i) \circ in_i \\ &= (id, V_{\eta_i}(v)) \\ &= (id, v) \circ (\eta_i, \xi_i)\end{aligned}$$

which by jointly surjective coproduct injections (η_i, ξ_i) implies that $(id, y) \circ in = (id, v)$. Hence, C is free over X in $\mathbf{Alg}(\mathbf{\Sigma})$ which by uniqueness of free constructions implies that C is the term algebra $T_\Sigma(X)$. □

Lemma A.3.11 (Cocreations and Jointly Epic Morphisms)**.** *Given a functor $F : \mathbf{C} \to \mathbf{D}$, jointly epic morphisms $(a_i : A_i \to A)_{i \in \mathcal{I}}$ in $\mathbf{C}$, and commuting diagrams $(D_i)_{i \in \mathcal{I}}$ below, where $(b_i)_{i \in \mathcal{I}}$ are jointly epic. Moreover, let $\bar{f} : A \to \bar{B}$ be the F-cocreation of f via A, $\bar{f}_i$ the F-cocreation of f_i via A_i for all $i \in \mathcal{I}$, and $b_i^* : B_i \to B$ the unique repective morphism such that diagram (C_i) below commutes and $F(b_i^*) = b_i$. Then also $(b_i^*)_{i \in \mathcal{I}}$ are jointly epic.*



Proof. We have to show that $(b_i^*)_{i \in \mathcal{I}}$ are jointly epic $\mathbf{C}$-morphisms. So, let $g, h : \bar{B} \to C$ be morphisms such that $g \circ b_i = h \circ b_i$ for all $i \in \mathcal{I}$, then we have to show that $g = h$. For all $i \in \mathcal{I}$ we have

$$F(g) \circ b_i = F(g) \circ F(b_i^*) = F(g \circ b_i^*) = F(h \circ b_i^*) = F(h) \circ F(b_i^*) = h \circ b_i$$

which implies that $F(g) = F(h)$ because $(b_i)_{i \in \mathcal{I}}$ are jointly epic.

Now, due to the fact that $\bar{f}$ is an F-cocreation of f via A, there exists a unique $x : \bar{B} \to C$ such that $x \circ \bar{f} = g \circ \bar{f}$ and $F(x) = F(g)$. Moreover, we have for all $i \in \mathcal{I}$ that

$$g \circ \bar{f} \circ a_i = g \circ b_i \circ f_i = h \circ b_i \circ f_i = h \circ \bar{f} \circ a_i$$

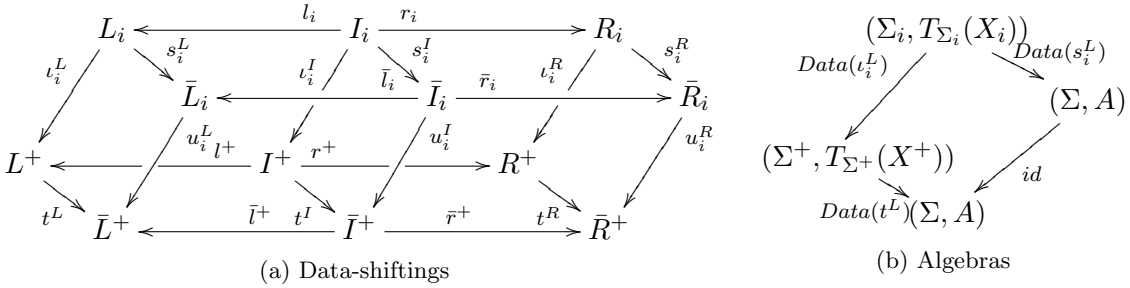
which by jointly epic $(a_i)_{i \in \mathcal{I}}$ implies that $g \circ \bar{f} = h \circ \bar{f}$. Thus, from uniqueness of x it follows that $g = x = h$. Hence, $(b_i^*)_{i \in \mathcal{I}}$ are jointly epic. □

Fact A.3.12 (Instantiation Condition and Parallel Productions)**.** *Given abstract productions* $(\varrho_i : L_i \overset{l_i}{\leftarrow} I_i \overset{r_i}{\rightarrow} R_i)_{i\in\mathcal{I}}$ *for instantiations, and let* $\varrho^+ = \biguplus_{i\in\mathcal{I}} \varrho_i : L^+ \overset{l^+}{\leftarrow} I^+ \overset{r^+}{\rightarrow} R^+$ *be a parallel production with injections* $\iota_i^C : C_i \to C^+$ *for* $C \in \{L, I, R\}$*. Moreover, let* $Inst$ *an instantiation, and* $(m_i : L_i \to Inst)_{i\in\mathcal{I}}$*,* $m^+ : L^+ \to Inst$ *matches such that* $m^+ \circ \iota_i^L = m_i$ *for all* $i \in \mathcal{I}$*. Then* ϱ^+ *and* m^+ *satisfy the instantiation condition for abstract productions if and only if for all* $i \in \mathcal{I}$ *we have that* ϱ_i *and* m_i *satisfy the instantiation condition for abstract productions.*

Proof. First, note that due to abstract productions $(\varrho_i)_{i\in\mathcal{I}}$ by Lemma A.3.10 we have that ϱ^+ also is an abstract production. Since the rule-morphisms are $\mathcal{M}_{AHL}$-morphisms that are isomorphic on the data type part, we can w. l. o. g. assume that all components of ϱ_i (for each $i \in \mathcal{I}$) have a common data type part $(\Sigma_i, T_{\Sigma_i}(X_i))$, and all components of ϱ^+ have a common data type part $(\Sigma^+, T_{\Sigma^+}(X^+))$.

Further, let $\bar{\varrho}^+$ be the data-shifting of ϱ^+ along m^+, and $(\bar{\varrho}_i)_{i\in\mathcal{I}}$ the data-shiftings of all ϱ_i along m_i. Then for (Σ, A) being the data type part of $Inst$, we have that the diagram in Figure A.6b commutes. Thus, due to $InstData$-cocreations s_i^C (for $C \in \{L, I, R\}$ and $i \in \mathcal{I}$), there exist unique morphisms $u_i^C : \bar{C}_i \to \bar{C}^+$ for every $C \in \{L, I, R\}$ and $i \in \mathcal{I}$, such that $u_i^C \circ s_i^C = t^C \circ \iota_i^C$ and $Data(u_i^C) = id_{(\Sigma,A)}$.

We have that the coproduct injections $(\iota_i^C)_{i\in\mathcal{I}}$ for every component $C \in \{L, I, R\}$ are jointly epic AHL-morphisms, since the coproducts in **wInst** are constructed via coproduct in **AHLNets** (see Fact A.3.8). Also, we have that $(id_{(\Sigma,A)})_{i\in\mathcal{I}}$ are jointly epic, because $id_{(\Sigma,A)}$ is an epimorphism. Moreover, the fact that all $\bar{\varrho}_i$ and $\bar{\varrho}^+$ are data-shiftings of ϱ_i and ϱ^+, respectively, means that they are $wInstData$-images. This in turn, according to Definition 4.4.1, means that the corresponding AHL-morphisms s_i^C and t^C are $Data$-cocreations for all $C \in \{L, I, R\}$ and $i \in \mathcal{I}$. Thus, using Lemma A.3.11 we obtain that also $(u_i^C)_{i\in\mathcal{I}}$ are jointly epic AHL-morphisms for all $C \in \{L, I, R\}$, and thus, they are jointly surjective.



(a) Data-shiftings (b) Algebras

Figure A.6: Data-shiftings of productions

Now, we consider the two directions of the proof separately.

If. Let ϱ_i and m_i satisfy the instantiation condition for all $i \in \mathcal{I}$. Then we have that the data-shiftings $\bar{\varrho}_i$ for all $i \in \mathcal{I}$ are concrete productions, meaning that their components are concrete instantiations. As shown above, there are jointly surjective morphisms $u_i^C : \bar{C}_i \to \bar{C}^+$ for every component $C \in \{L, I, R\}$ which by Lemma A.5.6 implies that also $\bar{L}^+$, $\bar{I}^+$ and $\bar{R}^+$ are concrete instantiations. Hence, $\bar{\varrho}^+$ is a concrete production, which means that ϱ^+ and m^+ satisfy the instantiations condition.

Only If. Now, let ϱ^+ and m^+ satisfy the instantiations condition. Then we have that $\bar{\varrho}^+$ is a concrete production which means that $\bar{L}^+$, $\bar{I}^+$ and $\bar{R}^+$ are concrete instantiations. As shown above, there are morphisms $u_i^C : \bar{C}_i \to \bar{C}^+$ for every component $C \in \{L, I, R\}$

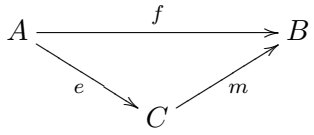
such that $Data(u_i^C) = id_{(\Sigma,A)}$, i. e. all of these morphisms have an isomorphic data type part. Thus, according to Fact A.6.9, each $\bar{C}_i$ ($i \in \mathcal{I}$) is the W-creation of $\bar{C}^+$ ($C \in \{L, I, R\}$) which means that they are concrete instantiations. Hence, $\bar{\varrho}_i$ is a concrete production for every $i \in \mathcal{I}$, implying that ϱ_i and m_i satisfy the instantiation condition for all $i \in \mathcal{I}$.

□

A.4 $\mathcal{E}$-$\mathcal{M}'$ Factorisation and $\mathcal{E}'$-$\mathcal{M}'$ Pair Factorisation

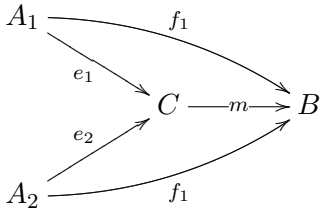
Two construction that are often needed in the context of $\mathcal{M}$-adhesive categories are $\mathcal{E}$-$\mathcal{M}'$ factorisations and $\mathcal{E}$'-$\mathcal{M}'$ pair factorisations as defined in the following. The intuitive idea of the morphism class $\mathcal{E}$ is that of epimorphic morphisms, while the idea of the morphism class $\mathcal{E}$' is that of jointly epimorphic morphism pairs. The idea of the morphism class $\mathcal{M}'$ is that of monomorphic morphisms that does not need to be included in the class $\mathcal{M}$.

Definition A.4.1 ($\mathcal{E}$-$\mathcal{M}'$ Factorisation)**.** Given an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$ and classes of morphisms $\mathcal{E}$ and $\mathcal{M}'$. We say that $\mathbf{C}$ has $\mathcal{E}$-$\mathcal{M}'$ *factorisations*, if for each morphism $f : A \to B$ there exist an object C and morphisms $e : A \to C \in \mathcal{E}$ and $m : C \to B \in \mathcal{M}'$ such that $f = m \circ e$.



△

Definition A.4.2 ($\mathcal{E}'$-$\mathcal{M}'$ Pair Factorisation)**.** Given an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$, a class $\mathcal{E}'$ of morphism pairs with the same codomain and a class $\mathcal{M}'$ of morphisms. We say that $\mathbf{C}$ has $\mathcal{E}'$-$\mathcal{M}'$ *pair factorisations*, if for each pair of morphisms $f_1 : A_1 \to B$ and $f_2 : A_2 \to B$ there exist an object C and morphisms $e_1 : A_1 \to C$, $e_2 : A_2 \to C$ and $m : C \to B$ with $(e_1, e_2) \in \mathcal{E}'$ and $m \in \mathcal{M}'$ such that $m \circ e_1 = f_1$ and $m \circ e_2 = f_2$.



△

Fact A.4.3 ($\mathcal{E}$-$\mathcal{M}'_{AHL}$-Factorisation of AHL-Nets)**.** *The $\mathcal{M}$-adhesive category of AHL-nets $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ has $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisations for the class $\mathcal{E}$ of all epimorphisms and the class $\mathcal{M}'_{AHL}$ of all monomorphisms. Given an AHL-morphism $f : AN_1 \to AN_2$, the $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisation $e : AN_1 \to AN_0$ and $m : AN_0 \to AN_2$ of f can be constructed as componentwise image factorisation with $AN_0 = f(AN_1)$.*

Proof. We have to show that there are well-defined image factorisations of AHL-nets. Let $f : AN_1 \to AN_2$, we obtain the image $AN_0 = f(AN_1)$ as AHL-net

$$AN_0 = (\Sigma_0, P_0, T_0, pre_0, post_0, cond_0, type_0, A_0)$$

constructed as componentwise image:

- $\Sigma_0 = f_\Sigma(\Sigma_1)$,

- $P_0 = f_P(P_1)$,
- $T_0 = f_T(T_1)$,
- $A_0 = f_A(A_1)$,
- $pre_0 = (f_A \otimes f_P)^{\oplus} \circ pre_1$,
- $post_0 = (f_A \otimes f_P)^{\oplus} \circ post_1$,
- $cond_0 = \mathcal{P}_{fin}(f_\Sigma^\#) \circ cond_1$, and
- $type_0 = f_\Sigma \circ type_1$.

The image construction of functions f_P and f_T is an epi-mono factorisation of sets, leading to unique functions $e_P : P_1 \to P_0$ and $m_P : P_0 \to P_2$ such that $m_P \circ e_P = f_P$, and $e_T : T_1 \to T_0$ and $m_T : T_0 \to T_2$ such that $m_T \circ e_T = f_T$, and we have that e_P and e_T are surjections and m_P and m_T are inclusions.

For the signature $\Sigma_0 = (S_0, OP_0; X_0)$, we obtain S_0 also as epi-mono factorisation of function f_S, implying a surjection $e_S : S_1 \to S_0$ and an inclusion $m_S : S_0 \hookrightarrow S_2$. Note that we have $e_S(s) = f_S(s)$.

For each $op : s_1...s_n \to s$ in OP_1 we have $f_{OP}(op) : f_S(s_1)...f_S(s_n) \to f_S(s)$ in OP_0. This means that there is also a surjection $e_{OP} : OP_1 \to OP_0$ with $e_{OP}(op) = f_{OP}(op)$, and $e_\sigma = (e_S, e_{OP})$ is a surjective signature morphism which follows directly from the construction of OP_0. Moreover, there is an inclusion $m_{OP} : OP_0 \hookrightarrow OP_2$ such that we have a signature inclusion $m_\sigma = (m_S, m_{OP})$.

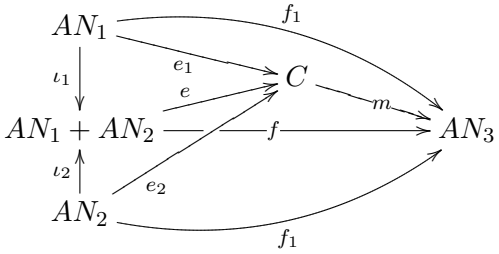
Further, for each variable $x \in X_s$ we have $f_X(x) \in X_{f_S(s)}$. This means that there is also a surjection $e_X : X_1 \to X_0$ such that $e_\Sigma = (e_\sigma, e_X)$ is a surjective **SigSets**-morphisms (see Definition A.1.8) which follows directly from the construction of X_0. Moreover, there is an inclusion $m_X : X_0 \hookrightarrow X_2$ such that we obtain a **SigSets**-inclusion $m_\Sigma = (m_\sigma, m_X)$.

The image A_0 of algebra A_1 can be constructed analogously to the homomorphic image defined in Definition 4.9 in [EM85]. For each $s_0 \in S_0$ there exists an $s \in S_1$ with $f_S(s) = s_0$. So, for the carrier sets, we have componentwise images $A_{f_S(s)} = f_{A,s}(A_{1,s})$ of functions $f_{A,s}$, implying that we have surjections $e_{A,s} : A_{1,s} \to A_{f_S(s)}$ with $e_{A,s}(a) = f_{A,s}(a)$ and inclusions $m_{A,s} : A_{f_S(s)} \hookrightarrow A_{2,f_S(s)}$ for all $s \in S_1$. For all constant symbols $c :\to s$ in OP_1 we have $f_{OP}(c)_{A_0} = c_{A_1}$, and for all operations $op : s_1...s_n \to s$ in OP_1 we have $f_{OP}(op)_{A_0} : f_{A,s_1}(A_{1,s_1}) \times ... \times f_{A,s_n}(A_{1,s_n}) \to f_{A,s}(A_{1,s})$ with $f_{OP}(op)_{A_0}(x_1, ..., x_n) = f_{OP}(op)_{A_2}(x_1, ..., x_n)$ for $x_i \in f_{A,s_i}(A_{1,s_i})$ and $i = 1, ..., n$.

Then, since for each $s \in S_1$ the functions $e_{A,s}$ map everything exactly in the same way as $f_{A,s}$, it follows from the construction of A_0 that for $e_A = (e_{A,s})_{s \in S_1}$ we have that $(e_\Sigma, e_A) : (\Sigma_1, A_1) \to (\Sigma_0, A_0)$ is a generalised homomorphism. Moreover, by definition $m_A = (m_{A,s})_{s \in S_1}$ we obtain an generalised algebra homomorphism $(m_\Sigma, m_A) : (\Sigma_0, A_0) \hookrightarrow (\Sigma_2, A_2)$.

Hence, by defining $e = (e_\Sigma, e_P, e_T, e_A)$ we obtain that $e : AN_1 \to AN_0$ is a well-defined AHL-morphism: The required compatibility of pre, post, cond and type functions follow immediately from the definition of these functions in AN_0 and the fact that the components of e map everything exactly in the same way as f does. Moreover, by definition $m = (m_\Sigma, m_P, m_T, m_A)$ we obtain a well-defined AHL-morphism $m : AN_0 \to AN_2$ which follows simply from the fact that all components of m are inclusions. Note that all components of e are surjective which means that e is an epimorphism, and all components of m are inclusions and therefore injective which means that m is a monomorphism. □

Fact A.4.4 ($\mathcal{E}'$-$\mathcal{M}'_{AHL}$ Pair Factorisation of AHL-Nets)**.** *The $\mathcal{M}$-adhesive category of AHL-nets* $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ *has $\mathcal{E}'$-$\mathcal{M}'_{AHL}$ pair factorisations for the class $\mathcal{E}'$ of all pairs of jointly epimorphic morphisms and the class $\mathcal{M}'$ of all monomorphisms.*



Construction and proof. Given two morphisms $f_1 : AN_1 \to AN_3$ and $f_2 : AN_2 \to AN_3$, then the $\mathcal{E}'$-$\mathcal{M}'_{AHL}$ pair factorisation of f_1 and f_2 can be obtained in the following way:

1. The coproduct $AN_1 + AN_2$ is constructed leading to jointly epic coproduct injections $\iota_1 : AN_1 \to AN_1 + AN_2$ and $\iota_2 : AN_2 \to AN_1 + AN_2$, and a unique induced morphism $f : AN_1 + AN_2 \to AN_3$ such that $f \circ \iota_1 = f_1$ and $f \circ \iota_2 = f_2$.
2. Using Fact A.4.3, the morphism f can be factorised into an AHL-net C, $\mathcal{E}$-morphism $e : AN_1 + AN_2 \to C$ and $\mathcal{M}'_{AHL}$-morphism $m : C \to AN_3$ such that $m \circ e = f$.
3. We define $e_1 := e \circ \iota_1$ and $e_2 := e \circ \iota_2$. Since $e \in \mathcal{E}$, it is an epimorphism, and thus, the compositions e_1 and e_2 are jointly epimorphic. Hence, $(e_1, e_2) \in \mathcal{E}'$. □

Fact A.4.5 ($\mathcal{E}$-$\mathcal{M}'_{AHL}$ Factorisation of Instantiations)**.** *The $\mathcal{M}$-adhesive categories of (weak) instantiations* $(\mathbf{wInst}, \mathcal{M}_{AHL})$ *and* $(\mathbf{Inst}, \mathcal{M}_{AHL})$ *have $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisations for the classes $\mathcal{E}$ of all epimorphisms and $\mathcal{M}'_{AHL}$ of all monomorphisms. Given a (weak) instantiation morphism $f : (inst_1, AN_1) \to (inst_2, AN_2)$, the factorisation $f = m \circ e$ with $e : (inst_1, AN_1) \to (inst_0, AN_0) \in \mathcal{E}$ and $m : (inst_0, AN_0) \to (inst_2, AN_2) \in \mathcal{M}'_{AHL}$ is obtained as follows:*

1. *The AHL-morphism $f : AN_1 \to AN_2$ is factorised into $AN_1 \xrightarrow{e} AN_0 \xrightarrow{m} AN_2$ using Fact A.4.3 with $e \in \mathcal{E}$ and $m \in \mathcal{M}'_{AHL}$.*
2. *The instantiation $(inst_0, AN_0)$ with morphism $m : (inst_0, AN_0) \to (inst_2, AN_2)$ is constructed as $wNet$- respectively Net-creation of $m : AN_0 \to AN_1$ via $(inst_1, AN_1)$, implying that also $e : (inst_1, AN_1) \to (inst_0, AN_0)$ is a (weak) instantiation morphism (see Fact A.6.5).*

Proof. Given a (weak) instantiation morphism $f : (inst_1, AN_1) \to (inst_2, AN_2)$. The existence of the $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisation $f = m \circ e$ follows from Fact A.4.3 such that $e : AN_1 \to AN_0 \in morE$ and $m : AN_0 \to AN_2 \in \mathcal{M}'_{AHL}$. According to Fact A.6.5, the functor $wNet : \mathbf{wInst} \to \mathbf{AHLNets}$ and $Net : \mathbf{Inst} \to \mathbf{AHLNets}$ have creations, and the $wNet$- respectively Net-creation of m via AN_2 can be obtained as $m : (inst_0, AN_1) \to (inst_2, AN_2)$, where $(inst_0, AN_1)$ is the restriction of $(inst_2, AN_2)$ along m (see Definition 4.5.3).

Then, by universal property of functor creation m, due to AHL-morphism $e : AN_1 \to AN_2$ and (weak) instantiation morphism $f : (inst_1, AN_1) \to (inst_2, AN_2)$ with commutativity $m \circ e = f$ in **AHLNets**, we have that $e : (inst_1, AN_1) \to (inst_0, AN_0)$ is a (weak) instantiation morphism (see Fact A.6.5).

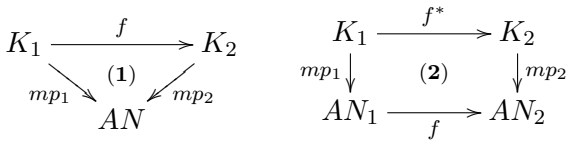
Hence, we have that $(inst_1, AN_1) \xrightarrow{e} (inst_0, AN_0) \xrightarrow{m} (inst_2, AN_2)$ is an $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisation of f in **wInst** respectively **Inst**. □

Fact A.4.6 ($\mathcal{E}'$-$\mathcal{M}'_{AHL}$ Pair Factorisation of Instantiations)**.** *The $\mathcal{M}$-adhesive categories* $(\mathbf{wInst}, \mathcal{M}_{AHL})$ *and* $(\mathbf{Inst}, \mathcal{M}_{AHL})$ *have $\mathcal{E}'$-$\mathcal{M}'_{AHL}$ pair factorisations for the class $\mathcal{E}'$ of all pairs of jointly epimorphic morphisms and the class $\mathcal{M}'$ of all monomorphisms.*

Proof. The construction and proof of this fact works completely analogously to Fact A.4.4. □

In the following we transfer the results for AHL-nets above also to typed AHL-nets. A *typed AHL-net* is an AHL-morphism $mp : K \to AN$, and we say that the AHL-net is typed over AN. Given an AHL-net AN, a morphism between typed AHL-nets $mp_1 : K_1 \to AN$ and $mp_2 : K_2 \to AN$ is given by an AHL-morphism $f : K_1 \to K_2$ such that $mp_2 \circ f = mp_1$. Typed AHL-nets over the same AHL-net AN and morphisms between them form the slice category $\mathbf{AHLNets} \setminus AN$.

Moreover, given typed AHL-nets $mp_1 : K_1 \to AN_1$ and $mp_2 : K_2 \to AN_2$, a morphism between mp_1 and mp_2 is given by a pair $(f_K : K_1 \to K_2, f_{AN} : AN_1 \to AN_2)$ of AHL-nets such that $mp_2 \circ f_K = f_{AN} \circ mp_1$. Typed AHL-nets over different AHL-nets and morphisms between them form the arrow category $\mathbf{AHLNets}^{\to}$.

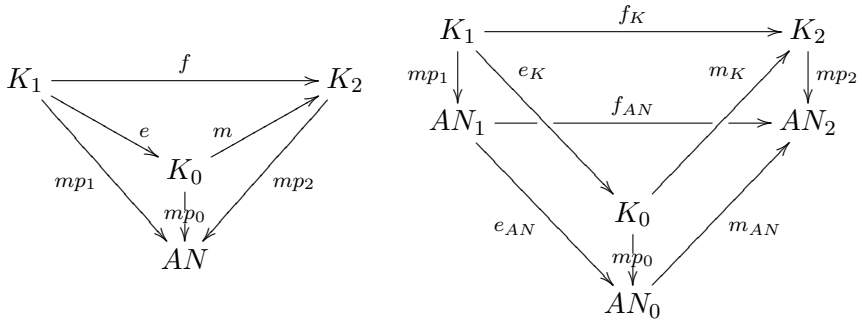


Fact A.4.7 ($\mathcal{E}$-$\mathcal{M}'_{AHL}$ Factorisation of Typed AHL-Nets)**.**

1. *For all AHL-nets AN the slice category $\mathbf{AHLNets} \setminus AN$ of AHL-nets typed over AN has $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisations that can be constructed as $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisation of the corresponding AHL-morphisms.*

2. *The category $\mathbf{AHLNets}^{\to}$ of all typed AHL-nets has $\mathcal{E}^{\to}$-$\mathcal{M}'^{\to}_{AHL}$ factorisations. The classes $\mathcal{E}^{\to}$ is the class of all epimorphisms in $\mathbf{AHLNets}^{\to}$, given by the class of $\mathbf{AHLNets}^{\to}$-morphisms (e_1, e_2) such that e_1 and e_2 are epimorphisms. Analogously, $\mathcal{M}'^{\to}_{AHL}$ of all monomorphisms in $\mathbf{AHLNets}^{\to}$, given by the class of $\mathbf{AHLNets}^{\to}$-morphisms (m_1, m_2) such that m_1 and m_2 are monomorphisms.*

   The factorisation can be constructed in the following way: Given an $\mathbf{AHLNets}^{\to}$-morphism $(f_K, f_{AN}) : mp_1 \to mp_2$, the $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisation is constructed as $e = (e_K, e_{AN}) : mp_1 \to mp_0$ and $m = (m_K, m_{AN}) : mp_0 \to mp_2$, where $f_K = m_K \circ e_K$ and $f_{AN} = m_{AN} \circ e_{AN}$ are obtained as $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisations in $\mathbf{AHLNets}$, and the typed AHL-net $mp_0 : K_0 \to AN_0$ is defined as composition $mp_0 := m_{AN}^{-1} \circ mp_2 \circ m_K$.



Proof. 1. Given an AHL-net AN $\mathbf{AHLNets} \setminus AN$-objects $mp_1 : K_1 \to AN$ and $mp_2 : K_2 \to AN$ and a morphism $f : mp_1 \to mp_2$. By definition of $\mathbf{AHLNets} \setminus AN$, we have that f is an AHL-morphism $f : K_1 \to K_2$ such that $mp_2 \circ f = mp_1$. Using Fact A.4.3, we obtain an $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisation $f = m \circ e$ with $e : K_1 \to K_0 \in \mathcal{E}$ and $m : K_0 \to K_2 \in \mathcal{M}'_{AHL}$.

Moreover, by composition $mp_0 := mp_2 \circ m$ we obtain a typed AHL-net $mp_0 : K_0 \to AN_0$, where it additionally holds that $mp_0 \circ e = mp_2 \circ \circ e = mp_2 \circ f = mp_1$. Hence, we have **AHLNets** $\setminus AN$-morphisms $e : mp_1 \to mp_0 \in \mathcal{E}$ and $m : mp_0 \to mp_2 \in \mathcal{M}'_{AHL}$ with $f = m \circ e$.

2. Given an **AHLNets**$^{\to}$-morphism $(f_K, f_{AN}) : mp_1 \to mp_2$, we define $e := (e_K, e_{AN})$ and $m := (m_K, m_{AN})$, where $f_K = m_K \circ e_K$ and $f_{AN} = m_{AN} \circ e_{AN}$ are obtained as $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisations in **AHLNets**.

   Now, we define $mp_0 = m_{AN}^{-1} \circ mp_2 \circ m_K$, and we have to show that mp_0 is well-defined. So, since m_{AN} is a monomorphism and hence injective, it suffices to show that for every element x in K_0 there exists a corresponding element y in AN_0 such that $mp_2 \circ m_K(x) = m_{AN}(y)$. Let $x \in P_{K_0}$. Then, since e_K is an epimorphism and hence surjective, there is $x' \in P_{K_1}$ with $e_K(x') = x$. Moreover, we have $y = e \circ mp_1(x') \in P_{AN_0}$ with

$$\begin{aligned} mp_2 \circ m_K(x) &= mp_2 \circ m_K \circ e_K(x') = mp_2 \circ f_K(x') \\ &= f_{AN} \circ mp_1(x') = m_{AN} \circ e_{AN} \circ mp_1(x') \\ &= m_{AN}(y) \end{aligned}$$

   which means that y is the required element such that $mp_0(x) = y$ is a well-defined mapping of places. The proof for the other components works analogously. The fact that mp_0 is a well-defined AHL-morphism can be derived from the fact that m_{AN}, mp_2 and m_K are AHL-morphisms. The fact that $m = (m_K, m_{AN})$ is a **AHLNets**$^{\to}$-morphism follows directly from definition of m_K. Moreover, for $e = (e_K, e_{AN}$, we have

$$\begin{aligned} mp_0 \circ e_K &= m_{AN}^{-1} \circ mp_2 \circ m_K \circ e_K = m_{AN}^{-1} \circ mp_2 \circ f_K \\ &= m_{AN}^{-1} \circ f_{AN} \circ mp_1 = m_{AN}^{-1} \circ m_{AN} \circ e_{AN} \circ mp_1 \\ &= e_{AN} \circ mp_1 \end{aligned}$$

   which means that also e is a **AHLNets**$^{\to}$-morphism.

   It remains to show that m is a monomorphism and e is an epimorphism in **AHLNets**$^{\to}$. The arrow category **AHLNets**$^{\to}$ is isomorphic to the functor category $[\mathbf{2}, \mathbf{AHLNets}]$, where $\mathbf{2}$ is the small category $\bullet \to \bullet$ with two objects and one non-trivial morphism between them. A natural transformation in a functor category $[\mathbf{C}, \mathbf{D}]$ is a monomorphism or epimorphism, if it is componentwise monomorphism or epimorphism, respectively, in $\mathbf{D}$. Thus, e is an epimorphism, because its components are epimorphisms in **AHLNets**, and m is a monomorphism, since its components are monomorphisms in **AHLNets**.

□

Fact A.4.8 ($\mathcal{E}_2$-$\mathcal{M}_{AHL}$-Factorisation of AHL-Nets)**.** *The $\mathcal{M}$-adhesive category* (**AHLNets**, $\mathcal{M}_{AHL}$) *has $\mathcal{E}_2$-$\mathcal{M}_{AHL}$ factorisations for the class*

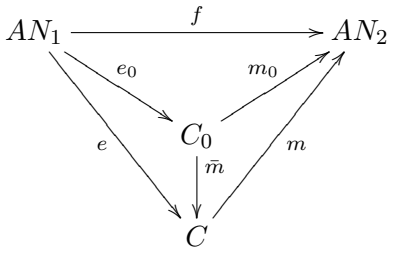
$$\mathcal{E}_2 = \{e = (e_\Sigma, e_P, e_T, e_A) \mid e_P \text{ and } e_T \text{ are surjective}\}$$

Given an AHL-morphism $f : AN_1 \to AN_2$, the $\mathcal{E}_2$-$\mathcal{M}$ factorisation is obtained in the following way:

1. *An $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisation $e_0 : AN_1 \to C_0$ and $m_0 : C_0 \to AN_2$ of f can be constructed according to Fact A.4.3.*

2. *An AHL-morphism $\bar{m} : C_0 \to C$ is obtained as data-image of C_0 along m_0, inducing a unique morphism $m : C \to AN_2$ such that $Data(m) = id_{A_{AN_2}}$ and $m \circ \bar{m} = m_0$.*

3. *The $\mathcal{E}_2$-$\mathcal{M}_{AHL}$ factorisation of f is given by AHL-net C and AHL-morphisms $e = \bar{m} \circ e_0 \in \mathcal{E}_2$ and $m \in \mathcal{M}_{AHL}$.*



Proof. We have to show that the construction above is well-defined and does in fact lead to an $\mathcal{E}$-$\mathcal{M}_{AHL}$ factorisation.

1. In Fact A.4.3 it is shown that **AHLNets** has $\mathcal{E}$-$\mathcal{M}'_{AHL}$ factorisations.

2. Using the construction of data-images in Definition 3.3.1, we obtain an AHL-net C together with an AHL-morphism $\bar{m} : C_0 \to C$. Moreover, by Fact A.6.10 we know that the data-image $\bar{m}$ is a *Data*-cocreation of m_0 via C_0. Hence, due to generalised homomorphism $id_{A_{AN_2}} : A_{AN_2} \to A_{AN_2}$ the universal property of cocreations induces a unique morphism $m : C \to AN_2$ such that $Data(m) = id_{A_{AN_2}}$ and $m \circ \bar{m} = m_0$.

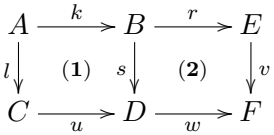
3. The morphisms $e := \bar{m} \circ e_0$ and m are a factorisation of f, because we have $f = m_0 \circ e_0 = m \circ \bar{m} \circ e_0 = m \circ e$. According to the construction of data-images, we have that the P- and T-components of $\bar{m}$ are identities. Together with the fact that e_0 is a surjective AHL-morphism, we have that the composition $e = \bar{m} \circ e_0$ also has surjective P- and T-components. Thus, we have that $e \in \mathcal{E}_2$.

   Moreover, due to the universal cocreation-property, we have that $Data(m) = id_{A_{AN_2}}$, i. e. m has an isomorphic data-type part. According to Fact A.6.10, we have that $m = (id_{\Sigma_{AN_2}}, m_{0,P}, m_{0,T}, id_{A_{AN_2}})$ which together with the fact that m_0 is a monomorphism implies that also m is a monomorphism. Hence, we have that $m \in \mathcal{M}_{AHL}$ which means that e and m are an $\mathcal{E}_2$-$\mathcal{M}_{AHL}$ factorisation of f. □

Corollary A.4.9 (Extremal $\mathcal{E}$-$\mathcal{M}_{AHL}$-Factorisation)**.** *The class $\mathcal{E}_2$ in Fact A.4.8 is the class of all* extremal morphisms w. r. t. $\mathcal{M}_{AHL}$. *An AHL-morphism e is called extremal w. r. t. $\mathcal{M}_{AHL}$, if for all AHL-morphisms m and f with $m \circ f = e$: $m \in \mathcal{M}_{AHL}$ implies that m is an isomorphism. Hence, the $\mathcal{E}_2$-$\mathcal{M}_{AHL}$ factorisation in Fact A.4.8 is an extremal $\mathcal{E}$-$\mathcal{M}_{AHL}$ factorisation.*

Proof. Let $e \in \mathcal{E}_2$ and AHL-morphisms m and f such that $m \circ f = e$ and $m \in \mathcal{M}_{AHL}$. Then by definition of morphism class $\mathcal{M}_{AHL}$ we know that the data type part (m_Σ, m_A) of m already is an isomorphism. Moreover, by definition of morphism class $\mathcal{E}_2$, we have that e_P and e_T are surjective functions, and due to componentwise composition of morphisms we have $m_P \circ f_P = e_P$ and $m_T \circ f_T = e_T$. Thus, by decomposition of surjective functions we obtain that also m_P and m_T are surjective. Since $m \in \mathcal{M}_{AHL}$ we also know that m_P and m_T are injective, and thus bijective. Hence, all components of m are isomorphisms which means that m is an isomorphism, and therefore, e is an extremal morphism w. r. t. $\mathcal{M}_{AHL}$. □

Definition A.4.10 ($\mathcal{M}$-$\mathcal{M}'$ PO-PB Decomposition Property)**.** An $\mathcal{M}$-adhesive category (**C**, $\mathcal{M}$) with a morphism class $\mathcal{M}'$ has the $\mathcal{M}$-$\mathcal{M}'$ *pushout-pullback decomposition property* if the following holds. Given the following commutative diagram with $l \in \mathcal{M}$ and $w \in \mathcal{M}'$, and where (1)+(2) is a pushout and (2) a pullback, then (1) and (2) are pushouts and also pullbacks:



△

Fact A.4.11 ($\mathcal{M}$-$\mathcal{M}'$ PO-PB Decomposition Property of AHL-Nets)**.** *The $\mathcal{M}$-adhesive category* (**AHLNets**, $\mathcal{M}_{AHL}$) *with the class $\mathcal{M}'$ of all monomorphisms has the $\mathcal{M}$-$\mathcal{M}'$ PO-PB decomposition property.*

Proof. Given the commutative diagram in Definition A.4.10 with $l \in \mathcal{M}_{AHL}$ and $w \in \mathcal{M}'_{AHL}$, where (1)+(2) is a pushout and (2) is a pullback. Due to closure of $\mathcal{M}_{AHL}$-morphisms under pushouts and pullbacks, we also have $s, v \in \mathcal{M}_{AHL}$. Then in the data type part of the diagram all vertical morphisms are isomorphisms which means that the data type parts of (1) and (2) are trivial pushouts and pullbacks in **Algs**. Moreover, the P- and T-components of l and w are injective functions, and thus $l, w \in \mathcal{M}_S$. Hence, we obtain that the P- and T-components of (1) and (2) are pushouts and pullbacks, due to $\mathcal{M}$ pushout-pullback decomposition lemma in Theorem 4.26.2 of [EEPT06b] which holds for $\mathcal{M}$-adhesive category (**Sets**, $\mathcal{M}_S$). So, by componentwise construction of pushouts and pullbacks in **AHLNets**, we also have that (1) and (2) are pushouts and pullbacks in **AHLNets**, due to corresponding pushouts and pullbacks in all underlying categories. □

A.5 Instantiations: Technical Details

The definition of instantiations in this thesis differs from the previous definition of instantiations in [Ehr05]. However, considering instantiations of AHL-process net, the definitions are equivalent in the sense that an instantiation can be uniquely transformed from one of the representations into the other one.

Fact A.5.1 (Equivalence of Definitions)**.** *Our definition of instantiations for AHL-process nets is equivalent to the definition of instantiations in [Ehr05]. In fact, there is a bijective correspondence between the two representations: For every inclusion $in : L \hookrightarrow Flat(K)$ such that $proj(K) \circ in$ is an isomorphism, there exists a unique corresponding instantiation $(inst, K)$ in* **PInst***, and vice versa.*

Proof. Consider an AHL-occurrence net K and a P/T net L together with an inclusion $in : Flat(K) \hookrightarrow Skel(K)$ such that $proj(K) \circ in$ is an isomorphism. Then we have an instantiation $Inst = (in \circ (proj(K) \circ in)^{-1}, K)$, because for $in \circ (proj(K) \circ in)^{-1} : Skel(K) \to Flat(K)$ we have

$$proj(K) \circ in \circ (proj(K) \circ in)^{-1} = id_{Skel(K)}$$

Vice versa, consider an instantiation $Inst = (inst, K)$ of an AHL-occurrence net. Performing an image factorization, we obtain a P/T net $L = inst(Skel(K))$ together with an inclusion $in : L \hookrightarrow Flat(K)$ and an epimorphism $e : Skel(K) \to L$ such that $inst = in \circ e$. Then we have

$$proj(K) \circ in \circ e = proj(K) \circ inst = id_{SkelK}$$

and

$$e \circ proj(K) \circ in \circ e = e \circ proj(K) \circ inst = e$$

which by epimorphism e implies $e \circ proj(K) \circ in$, which means that $proj(K) \circ in$ and e are inverse isomorphisms. □

Corollary A.5.2 (Instantiations are Injective)**.** *Given a (weak) instantiation $(inst, AN)$, then we have $inst \in \mathcal{M}_{PT}$ where $\mathcal{M}_{PT}$ is the class of all injective P/T-morphisms.*

Proof. Due to the requirement that $proj(AN) \circ inst = id_{Skel(AN)}$ (respectively $wproj(AN) \circ inst = id_{Skel(AN)}$), we have that $inst$ is a coretraction and thus a monomorphism in **PTNets**. Since monomorphisms in **PTNets** are injective, we have $inst \in \mathcal{M}_{PT}$. □

For every instantiation $(inst, AN)$ there is a corresponding weak instantiation $(winst, AN)$ such that the image of $inst$ is a subnet of $winst$. Therefore, the category **Inst** can be seen as a (full) subcategory of **wInst**, however, only up to isomorphism.

Fact A.5.3 (Correspondence Between Instantiations and Weak Instantiations)**.** *There is a functor W : **Inst** $\to$ **wInst**, defined by*

- $W(inst, AN) = (w(AN) \circ inst, AN)$ *for objects* $(inst, AN)$*, and*
- $W(f) = f$ *for morphisms* $f : (inst_1, AN_1) \to (inst_2, AN_2)$*,*

*and also functors W_0 : **PInst** $\to$ **wPInst** and W_1 : **ProcInst** $\to$ **wProcInst**, defined analogously. The functors W, W_0 and W_1 are fully injective.*

Proof. First we show that W is a well-defined functor. Given an instantiation $(inst, AN)$, the image $W(inst, AN) = (winst, AN)$ with $winst = w(AN) \circ inst$ is a weak instantiation, because due to the equation $proj(AN) = wproj(AN) \circ w(AN)$ (see Corollary 3.4.10), we have

$$wproj(AN) \circ winst = wproj(AN) \circ w(AN) \circ inst = proj(AN) \circ inst = id_{Skel(AN)}$$

Moreover, given an instantiation morphism $f : (inst_1, AN_1) \to (inst_2, AN_2)$, we have that f is an AHL-morphism $f : AN_1 \to AN_2$ with $inst_2 \circ Skel(f) = Flat(f) \circ inst_1$. It follows that $W(f) = f : W(inst_1, AN_1) \to W(inst_2, AN_2)$ is a weak instantiation morphism, because due to natural transformation $w : Flat \Rightarrow wFlat$, we have

$$w(AN_2) \circ inst_2 \circ Skel(f) = w(AN_2) \circ Flat(f) \circ inst_1 = wFlat(f) \circ inst_1$$

It remains to show that W is full and injective. W being full means that for every weak instantiation morphism $f : W(inst_1, AN_1) \to W(inst_2, AN_2)$ there is an instantiation morphism $\bar{f} : (inst_1, AN_1) \to (inst_2, AN_2)$ with $W(\bar{f}) = f$. Given a weak instantiation morphism $f : W(inst_1, AN_1) \to W(inst_2, AN_2)$, then we have that $f : AN_1 \to AN_2$ is an AHL-morphism with

$$w(AN_2) \circ \circ Flat(f) \circ inst_1 = wFlat(f) \circ w(AN_1) \circ inst_1 = w(AN_2) \circ inst_2 \circ wFlat(f)$$

and thus, since $w(AN_2)$ is a monomorphism in **PTNets**, it follows that $Flat(f) \circ inst_1 = inst_2 \circ wFlat(f)$. Hence, $f : (inst_1, AN_1) \to (inst_2, AN_2)$ is also an instantiation morphisms with $W(f) = f$, and we have that W is full.

Further, we have that W is injective, because given instantiations $(inst_1, AN_1)$, $(inst_2, AN_2)$ with $W(inst_1, AN_1) = W(inst_2, AN_2)$, then we have $AN_1 = AN_2$ and $w(AN_1) \circ inst_1 = w(AN_1) \circ inst_2$ which by monomorphism $w(AN_1)$ implies $inst_1 = inst_2$. For morphisms f and g with $W(f) = W(g)$, we trivially have $f = W(f) = W(g) = g$. Hence, W is also injective.

The proofs for W_0 and W_1 work analogously. □

Lemma A.5.4 (Weak Instantiations as Instantiations)**.** *Let $(winst, AN)$ be a weak instantiation. If $winst_T(T_{AN}) \subseteq CT_{AN}$, where CT_{AN} is the set of consistent transition assignments of AN (see Definition 3.4.5), then there exists a unique instantiation $(inst, AN)$ such that $W(inst, AN) = (winst, AN)$.*

Proof. Let $(winst, AN)$ be a weak instantiation such that $winst_T(T_{AN}) \subseteq CT_{AN}$. Then we can define a P/T-morphism $inst : Skel(AN) \to Flat(AN)$ with $inst = (winst_P, winst_T)$. The morphism is well-defined, because $winst_P(P_{AN}) \subseteq A \otimes P = P_{Flat(AN)}$ and $winst_T(T_{AN}) \subseteq CT_{AN} = T_{Flat(AN)}$, and the compatibility of pre and post domains follows from P/T-morphism $winst$ and the fact that $Flat$ and $wFlat$ have similar pre and post domains. Since $inst$ maps all places and transitions exactly in the same way as $winst$, we have for the inclusion $w(AN)$ that $w(AN) \circ inst = winst$, and hence we have $W(inst, AN) = (winst, AN)$. The uniqueness of $(inst, AN)$ follows from injectivity of W. □

Lemma A.5.5 (Reflection of Concrete Instantiations)**.** *Given a weak instantiation morphism $f : Inst_1 \to Inst_2$ with isomorphic data type part. If $Inst_2$ is a concrete instantiation, then so is $Inst_1$.*

Proof. Let $Inst_1 = (winst_1, AN_1)$ and $Inst_2 = (winst_2, AN_2)$ be weak instantiations. If $Inst_2$ is concrete, this means that there is a a concrete instantiation $(inst_2, AN_2)$ such that $W(inst_2, AN_2) = (winst_2, AN_2)$ (see Remark 4.3.8 on page 73). As shown in Fact A.6.9, the functor $W : \mathbf{Inst} \to \mathbf{wInst}$ has creations along weak instantiation morphisms with isomorphic data type part. Thus, there is a concrete instantiation $(inst_1, AN_1)$ with $W(inst_1, AN_1) = (winst_1, AN_1)$, meaning that $(winst_1, AN_1)$ is a concrete instantiation. □

Lemma A.5.6 (Preservation of Concrete Instantiations)**.** *Given weak instantiations $(Inst_i)_{i \in \mathcal{I}}$ and $Inst_2$ and jointly surjective weak instantiation morphisms $(f_i : Inst_i \to Inst_2)_{i \in \mathcal{I}}$. If all $(Inst_i)_{i \in \mathcal{I}}$ are concrete instantiations, then so is $Inst_2$.*

Proof. Let $(Inst_i = (winst_i, AN_i))_{i \in \mathcal{I}}$ be concrete instantiations and $Inst_2 = (winst_2, AN_2)$ a weak instantiation, and let $(f_i : Inst_i \to Inst_2)_{i \in \mathcal{I}}$ be jointly surjective weak instantiation morphisms. In order to show that $Inst_2$ is concrete, according to Lemma A.5.4, it suffices to show that $winst_T(T_{AN_2}) \subseteq CT_{AN_2}$.

So, let $tv_2 \in winst_{2,T}(T_{AN_2})$ be an arbitrary assignment of $Inst_2$. Due to injectivity of $winst_2$ (Corollary A.5.2) there is a unique transition $t_2 \in T_{AN_2}$ with $winst_{2,T}(t_2) = tv_2$, and since the morphisms f_i are jointly surjective, there exists a $j \in \mathcal{I}$ and $t_j \in T_{AN_j}$ such $f_{j,T}(t_j) = t_2$. Then, $Inst_j$ being an instantiation means that $winst_{j,T}(t_j) = tv_j$ is a consistent transition assignment. This implies that for $tv_j = (t_j, v_j)$ there is a firing step

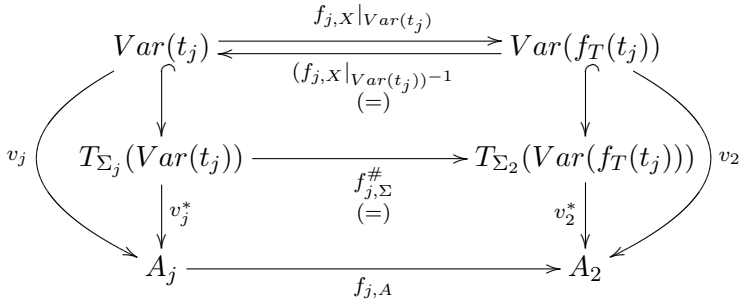
$$pre_{j,A}(t_j, v_j) \xrightarrow{t_j, v_j} post_{j,A}(t_j, v_j)$$

in AN_j. Using preservation of firing behaviour by AHL-morphisms (Fact 3.1.10), we obtain a firing step

$$(f_{j,A} \otimes f_{j,P})^{\oplus} \circ pre_{j,A}(t_j, v_j) \xrightarrow{f_{j,T}(t_j), v_2} (f_{j,A} \otimes f_{j,P})^{\oplus} \circ post_{j,A}(t_j, v_j)$$

in AN_2, where $v_2 = f_{j,A} \circ v_j \circ (f_{j,X}|_{Var(t_j)})^{-1}$.

We have $v_2 \circ f_{j,X}|_{Var(t_j)} = f_{j,A} \circ v_j \circ (f_{j,X}|_{Var(t_j)})^{-1} \circ f_{j,X}|_{Var(t_j)} = f_{j,A} \circ v_j$. Thus, by freeness of $T_{\Sigma_j}(Var(t_j))$ over $Var(t_j)$, we have $v_2^* \circ f_{j,\Sigma}^{\#} = (f_{j,A} \circ v_j)^* = f_{j,A} \circ v_j^*$.



So, let $pre_j(t_j) = \sum_{i=1}^{n}(term_i, p_i)$, then we have

$$
\begin{aligned}
(f_{j,A} \otimes f_{j,P})^{\oplus} \circ pre_{j,A}(t_j, v_j) &= (f_{j,A} \otimes f_{j,P})^{\oplus}(\sum_{i=1}^{n}(v_1^*(term_i), p_i)) \\
&= \sum_{i=1}^{n}(f_{j,A} \circ v_j^*(term_i), f_{j,P}(p_i)) = \sum_{i=1}^{n}(v_2^* \circ f_{j,\Sigma}^{\#}(term_i), f_{j,P}(p_i)) \\
&= ((v_2^* \circ f_{j,\Sigma}^{\#}) \otimes f_{j,P})^{\oplus}(\sum_{i=1}^{n}(term_i, p_i)) = (v_2^* \otimes id)^{\oplus} \circ (f_{j,\Sigma}^{\#} \otimes f_{j,P})^{\oplus} \circ pre_j(t_j) \\
&= (v_2^* \otimes id)^{\oplus} \circ pre_2(f_{j,T}(t_j)) = pre_{2,A}(f_{j,T}(t_j), v_2) = pre_{2,A}(t_2, v_2)
\end{aligned}
$$

and analogously we obtain $(f_{j,A} \otimes f_{j,P})^{\oplus} \circ post_{j,A}(t_j, v_j) = post_{2,A}(t_2, v_2)$, leading to a firing step

$$pre_{2,A}(t_2, v_2) \xrightarrow{t_2, v_2} post_{2,A}(t_2, v_2)$$

in AN_2. Thus, (t_2, v_2) is a consistent transition assignment. Finally, we have

$$
\begin{aligned}
tv_2 &= winst_T(t_2) = winst_T(f_{j,T}(t_j)) = wFlat(f_j)_T(winst_{j,T}(t_j)) = wFlat(f_j)_T(tv_j) \\
&= wFlat(f_j)_T(t_j, v_j) = (f_{j,T}(t_j), f_{j,A} \circ v_j \circ (f_{j,X}|_{Var(t_j)})^{-1}) = (t_2, v_2)
\end{aligned}
$$

which means that $tv_2 \in CT_{AN_2}$. Since this holds for all assignments of $Inst_2$, we have that $Inst_2$ is a concrete instantiation. □

The following fact states that the category of instantiations is – up to isomorphism – a subcategory of a comma category. Given two functors $F : \mathbf{A} \to \mathbf{C}$ and $G : \mathbf{B} \to \mathbf{C}$, a comma category $F \downarrow G$ consists of objects (A, B, op) with A in $\mathbf{A}$, B in $\mathbf{B}$ and $op : F(A) \to G(B)$ in $\mathbf{C}$, and morphisms $(f_A : A \to A', f_B : B \to B') : (A, B, op) \to (A', B', op')$ such that $op' \circ F(f_A) = G(f_B) \circ op$.

Fact A.5.7 (Instantiations are Comma Category)**.** *The category* **Inst** *is isomorphic to a category* $\mathbf{SubCom} \subseteq Skel \downarrow Flat$ *where* **SubCom** *is the subcategory of the comma category such that for objects* (A, B, op) *in* **SubCom** *we have* $A = B$ *and* $proj(A) \circ op = id_{Skel(A)}$*, and for morphisms* (f, g) *in* **SubCom** *we have* $f = g$*. Analogously, we have* $\mathbf{wInst} \cong \mathbf{D} \subseteq Skel \downarrow wFlat$ *where for objects in* **wSubCom** *it is required that* $wproj(A) \circ op = id_{Skel(A)}$*.*

Proof. It is easy to see that there is a bijective mapping between objects $(inst, AN)$ in **Inst** and $(AN, AN, inst)$ in **SubCom**, and morphisms $f : AN_1 \to AN_2$ in **Inst** and (f, f) in

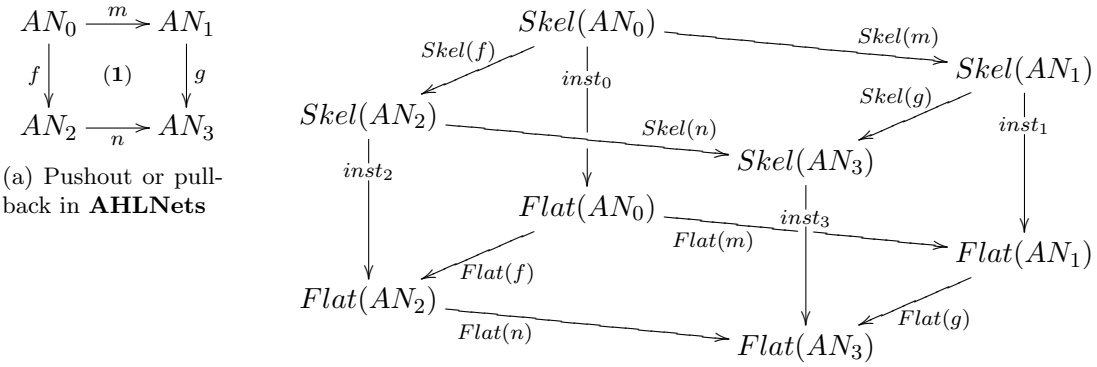
SubCom, where the compatibility property of morphisms in **Inst** and $Skel \downarrow Flat$ (and therefore also in **SubCom**) are defined analogously. Hence, we have **Inst** $\cong$ **SubCom** $\subseteq$ $Skel \downarrow Flat$. □

One of the benefits of comma categories is the inheritance of $\mathcal{M}$-adhesiveness from underlying $\mathcal{M}$-adhesive categories (see [EEPT06b, EGH10]). This can also be used for our categories of instantiations.

Fact A.5.8 (Pushouts and Pullbacks of Instantiations)**.**

1. *Given a span* $(inst_1, AN_1) \overset{m}{\leftarrow} (inst_0, AN_0) \overset{f}{\rightarrow} (inst_2, AN_2)$ *of (weak) instantiation morphisms with* $m \in \mathcal{M}_{AHL}$*, the pushout of* m *and* f *can be constructed as pushout (1) in* **AHLNets***, and* $inst_3 : Skel(AN_3) \rightarrow Flat(AN_3)$ *is uniquely induced by the pushout in the top of the cube in Figure A.7b.*

2. *Given a cospan* $(inst_1, AN_1) \overset{g}{\rightarrow} (inst_3, AN_3) \overset{n}{\leftarrow} (inst_2, AN_2)$ *of (weak) instantiation morphisms with* $n \in \mathcal{M}_{AHL}$*, the pullback of* g *and* n *can be constructed as pullback (1) in* **AHLNets***,* $inst_0 : Skel(AN_0) \rightarrow Flat(AN_0)$ *is uniquely induced by the pullback in the bottom of the cube in Figure A.7b.*



(a) Pushout or pullback in **AHLNets**

(b) Pushout or pullback in **Inst**

Figure A.7: Pushouts or pullbacks in **AHLNets** and **Inst**

Proof. We show the existence of pushouts and pullbacks only for instantiation, because the corresponding proof works analogously for weak instantiation, since the used facts for $Flat$ do also hold for $wFlat$.

1. Let $Inst_1 \overset{m}{\leftarrow} Inst_0 \overset{f}{\rightarrow} Inst_2$ be a span of instantiation morphisms with $m \in \mathcal{M}_{AHL}$. This means that we have a corresponding span $AN_1 \overset{m}{\leftarrow} AN_0 \overset{f}{\rightarrow} AN_2$ in **AHLNets**, and due to Fact A.1.11 there exists the pushout (1) in **AHLNets**, shown in Figure A.7a.

   Using Fact A.5.7, the span $Inst_1 \overset{m}{\leftarrow} Inst_0 \overset{f}{\rightarrow} Inst_2$ in **Inst** can be interpreted as a span

   $$(AN_1, AN_1, inst_1) \overset{(m,m)}{\leftarrow} (AN_0, AN_0, inst_0) \overset{(f,f)}{\rightarrow} (AN_2, AN_2, inst_2)$$

   in $Skel \downarrow Flat$, and by componentwise pushout construction in comma categories (under the condition that $Skel$ preserves pushouts, which is satisfied as shown in Fact 3.4.4; see Fact A.43 in [EEPT06b]), we obtain a pushout object $(AN_3, AN_3, inst_3)$ such that

the cube shown in Figure A.7b in **PTNets** commutes, and $inst_0$ is induced by the pushout in the top of the cube.

Now, for $(inst_3, AN_3)$ being an instantiation, it remains to show that $proj(AN_3) \circ inst_3 = id_{Skel(AN_3)}$. Note that due to universal property of the $Skel$-pushout in the top of the cube, the identity $id_{Skel(AN_3)}$ is the unique endomorphism $x : Skel(AN_3) \to Skel(AN_3)$ such that $x \circ Skel(n) = Skel(n)$ and $x \circ Skel(g) = Skel(g)$. This implies that $proj(AN_3) \circ inst_3 = id_{Skel(AN_3)}$, because using commutativity of the cube and the fact that $inst_1$ and $inst_2$ are instantiations, we obtain

$$\begin{aligned} proj(AN_3) \circ inst_3 \circ Skel(n) &= proj(AN_3) \circ Flat(n) \circ inst_2 \\ &= Skel(n) \circ proj(AN_2) \circ inst_2 \\ &= Skel(n) \circ id_{Skel(AN_2)} \\ &= Skel(n) \end{aligned}$$

and, analogously

$$\begin{aligned} proj(AN_3) \circ inst_3 \circ Skel(g) &= proj(AN_3) \circ Flat(g) \circ inst_1 \\ &= Skel(g) \circ proj(AN_1) \circ inst_1 \\ &= Skel(g) \circ id_{Skel(AN_1)} \\ &= Skel(g) \end{aligned}$$

Hence, the pushout object $(AN_3, AN_3, inst_3)$ in the comma category corresponds to a pushout object $(inst_3, AN_3)$ in the subcategory **Inst** (see Fact A.5.7).

2. For the proof of the existence of pullbacks consider a cospan $Inst_1 \xrightarrow{g} Inst_3 \xleftarrow{n} Inst_2$ of instantiations. Due to Fact A.1.11, we have the pullback (1) shown in Figure A.7a in **AHLNets**, and using again Fact A.5.7, the cospan $Inst_1 \xrightarrow{g} Inst_3 \xleftarrow{n} Inst_2$ corresponds to a cospan

$$(AN_1, AN_1, inst_1) \overset{(g,g)}{\to} (AN_0, AN_0, inst_0) \overset{(n,n)}{\leftarrow} (AN_2, AN_2, inst_2)$$

in $Skel \downarrow Flat$. In Fact A.43 in [EEPT06b] it is shown that pullbacks in the comma category can be constructed componentwise if $Flat$ preserves pullbacks. Since $n \in \mathcal{M}_{AHL}$, and by Fact 3.4.11 $Flat$ preserves pullbacks along $\mathcal{M}_{AHL}$, we obtain a pullback object $(AN_0, AN_0, inst_0)$ such that the cube in Figure A.7b commutes, and $inst_0$ is the unique morphism induced by the pullback in the bottom of the cube.

It remains to show that $inst_0$ is an instantiation, i. e. that we have $proj(AN_0) \circ inst_0 = id_{Skel(AN_0)}$. According to Fact 3.4.4, we have that the top face in the cube in Figure A.7b is a pullback, because $Skel$ preserves pullbacks along $\mathcal{M}_{AHL}$. Moreover, due to the universal pullback property, $id_{Skel(AN_0)}$ is the unique endomorphism $x : Skel(AN_0) \to Skel(AN_0)$ such that $Skel(f) \circ x = Skel(f)$ and $Skel(m) \circ x = Skel(m)$. This implies that $proj(AN_0) \circ inst_0 = id_{Skel(AN_0)}$, because we have

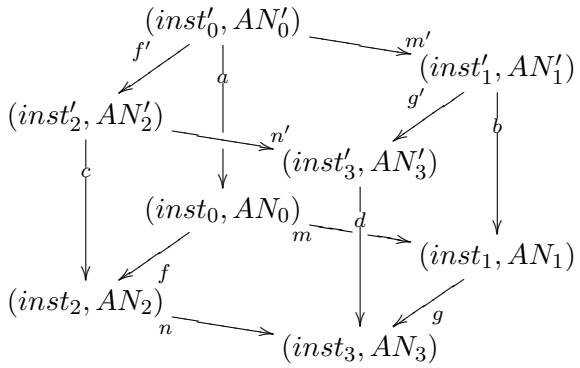
$$\begin{aligned} Skel(f) \circ proj(AN_0) \circ inst_0 &= proj(AN_2) \circ Flat(f) \circ inst_0 \\ &= proj(AN_2) \circ inst_2 \circ Skel(f) \\ &= id_{Skel(AN_2)} \circ Skel(f) \\ &= Skel(f) \end{aligned}$$

and analogously $Skel(m) \circ proj(AN_0) \circ inst_0 = Skel(m)$. Hence, the pullback object $(AN_0, AN_0, inst_0)$ corresponds to a pullback object $(inst_0, AN_0)$ in the subcategory **Inst** (see Fact A.5.7). □

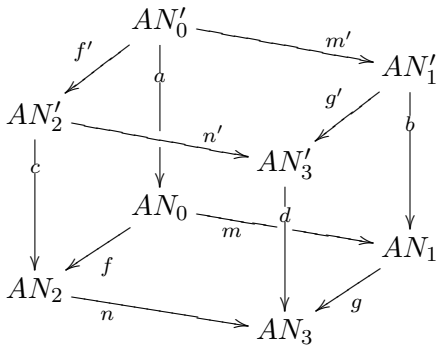
Fact A.5.9 (Instantiations are $\mathcal{M}$-Adhesive)**.** *The categories* $(\mathbf{Inst}, \mathcal{M}_{AHL})$ *of instantiations and* $(\mathbf{wInst}, \mathcal{M}_{AHL})$ *of weak instantiations are* $\mathcal{M}$*-adhesive, where* $\mathcal{M}_{AHL}$ *is the class of injective AHL-morphisms with isomorphic data type part.*

Proof. We have to show that $(\mathbf{Inst}, \mathcal{M}_{AHL})$ and $(\mathbf{wInst}, \mathcal{M}_{AHL})$ satisfy the properties of $\mathcal{M}$-adhesive categories in Definition A.1.9. We consider w. l. o. g. only the category $(\mathbf{Inst}, \mathcal{M}_{AHL})$.

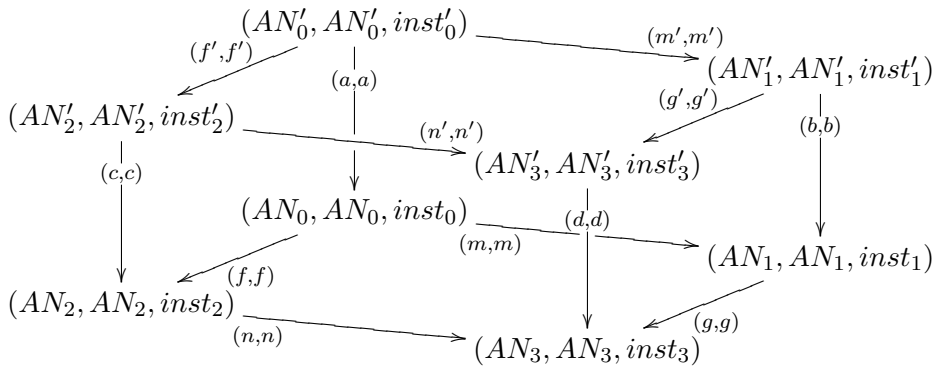
1. The class $\mathcal{M}_{AHL}$ is closed under composition, decomposition and isomorphism by Fact A.1.11.
2. The fact that **Inst** has pushouts and pullbacks along $\mathcal{M}_{AHL}$ is shown in Fact A.5.8.
3. The fact that $\mathcal{M}_{AHL}$-morphisms are closed under pushouts and pullbacks, follows from the construction of underlying pushouts and pullbacks in **AHLNets**, and the fact that $\mathcal{M}_{AHL}$ is closed under pushouts and pullbacks in **AHLNets**.
4. Consider a commutative cube of instantiations as shown in Figure A.8a. Since instantiation morphisms are special AHL-morphisms, there is a corresponding cube in **AHLNets**, depicted in Figure A.8b. Moreover, due to Fact A.5.7 we also obtain a corresponding cube in $Skel \downarrow Flat$ as depicted in Figure A.8c.



(a) Commutative cube in **Inst**



(b) Commutative cube in **AHLNets**



(c) Commutative cube in $Skel \downarrow Flat$

Figure A.8: Commutative cubes

Now, due to Fact A.5.8 and the componentwise construction of pushouts and pullbacks in comma categories, we know that if one of the faces of the cube in Figure A.8a is a pushout or a pullback, then the corresponding faces of the cubes in Figure A.8b and

Figure A.8c are also pushouts or pullbacks, respectively. Vice versa, given a pushout or pullback in one of the faces of the cube in Figure A.8c, there are corresponding pushouts or pullback, respectively, in corresponding faces of the other cubes. This implies that the satisfaction of the vertical Van-Kampen cube property in **Inst** can be directly inherited from the fact that $Skel \downarrow Flat$ is $\mathcal{M}$-adhesive (see Theorem 4.15.4 in [EEPT06b]). □

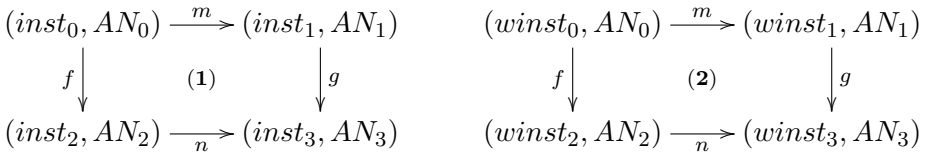
Fact A.5.10 (Pushouts and Pullbacks of Instantiations as Weak Instantiations)**.** *The functor* $W : \mathbf{Inst} \to \mathbf{wInst}$ *preserves and creates pushouts as well as pullbacks, i. e.*

1. *(Preservation of pushouts) If diagram (1) is pushout in* **Inst***, then diagram (2) is pushout in* **wInst***, where* $W(inst_i, AN_i) = (winst_i, AN_i)$ *for* $i \in \{0, \dots, 3\}$.

2. *(Preservation of pullbacks) If diagram (1) is pullback in* **Inst***, then diagram (2) is pullback in* **wInst***, where* $W(inst_i, AN_i) = (winst_i, AN_i)$ *for* $i \in \{0, \dots, 3\}$.

3. *(Creation of pushouts) If diagram (2) is pushout in* **wInst** *with* $W(inst_i, AN_i) = (winst_i, AN_i)$ *for* $i \in \{0, \dots, 2\}$*, then there exists an instantiation* $(inst_3, AN_3)$ *and morphisms* g *and* n *such that (1) is pushout in* **Inst** *and* $W(inst_3, AN_3) = (winst_3, AN_3)$.

4. *(Creation of pullbacks) If diagram (2) is pullback in* **wInst** *with* $W(inst_i, AN_i) = (winst_i, AN_i)$ *for* $i \in \{1, \dots, 3\}$*, then there exists an instantiation* $(inst_0, AN_0)$ *and morphisms* f *and* m *such that (1) is pullback in* **Inst** *and* $W(inst_0, AN_0) = (winst_0, AN_0)$.



(a) Pushout or pullback in **Inst**

(b) Pushout or pullback in **wInst**

Figure A.9: Pushout or pullback of instantiations as weak instantiations

Proof. 1. *(Preservation of pushouts)* Given pushout (1) in **Inst**, according to Fact A.5.8 we have the upper commutative cube in Figure A.10, where the top face is a pushout. Moreover, due to the fact that $w : Flat \Rightarrow wFlat$ is a natural transformation, we also have the lower commutative cube in Figure A.10. By Fact A.5.8, the pushout (2) can be obtained as $(winst_3, AN_3)$ such that $winst_3 : Skel(AN_3) \to wFlat(AN_3)$ is the unique morphism induced by the pushout in the top of the cube in Figure A.10. Due to the uniqueness of this morphism together with the fact that the front faces of the cube commute, we have that $winst_3 = w(AN_3) \circ inst_3$, and thus $W(inst_3, AN_3) = (winst_3, AN_3)$.

2. *(Preservation of pullbacks)* This proof works analogously to the one for item 1.

3. *(Creation of pushouts)* Given pushout (2) in **wInst** with $W(inst_i, AN_i) = (winst_i, AN_i)$ for $i \in \{0, \dots, 2\}$, by Fact A.5.8 we obtain pushout (1) in **Inst** such that the upper cube in Figure A.10 commutes. Analogously to item 1 above we have that $w(AN_3) \circ inst_3$ is the unique morphism $winst_3 : Skel(AN_3) \to wFlat(AN_3)$ induced by the pushout in the top of the cube, and hence we have $W(inst_3, AN_3) = (winst_3, AN_3)$.

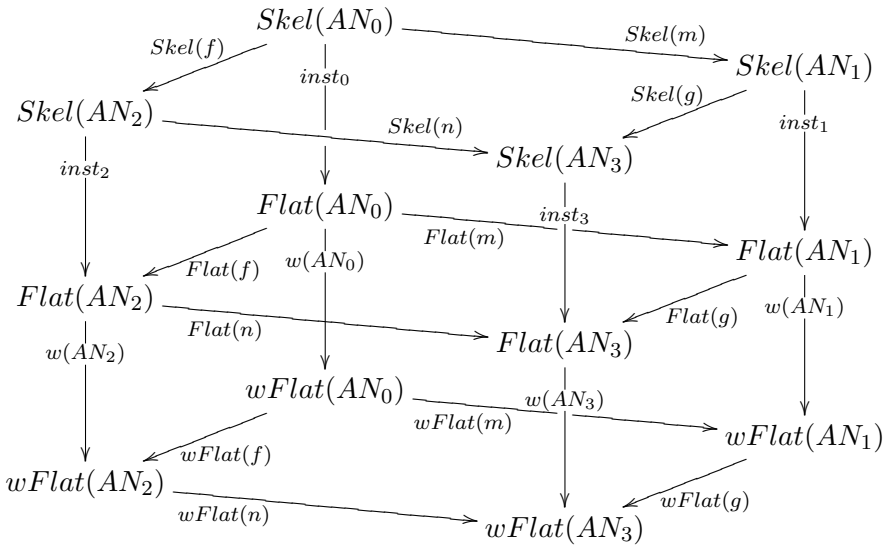


Figure A.10: Commutative cube of instantiations and weak instantiations

4. *(Creation of pullbacks)* This proof works analogously to item 3.

□

The following fact states that the existence of direct transformations of weak instantiations depends only on the existence of a corresponding transformation of the underlying AHL-net part. This is also useful for concrete instantiations, since every concrete instantiation can be seen as a special weak instantiation.

Fact A.5.11 (Direct Transformation of Weak Instantiations)**.** *Given a production for instantiations* $\varrho : (winst_L, L) \overset{l}{\leftarrow} (winst_I, I) \overset{r}{\rightarrow} (winst_R, R)$*, a weak instantiation* $(winst, AN)$ *and a (match) morphism* $m : (winst_L, L) \rightarrow (winst, AN)$*. There exists a direct transformation of weak instantiations* $(winst, AN) \overset{(\varrho,m)}{\Longrightarrow} (winst', AN')$*, iff there exists a direct transformation of AHL-nets* $AN \overset{Net(\varrho),m}{\Longrightarrow} AN'$ *using the underlying production for AHL-nets* $Net(\varrho) : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$*.*

$$\begin{array}{ccccc} \varrho: & (winst_L, L) & \overset{l}{\longleftarrow} & (winst_I, I) & \overset{r}{\longrightarrow} & (winst_R, R) \\ & \downarrow m & (1) & \downarrow c & (2) & \downarrow n \\ & (winst, AN) & \overset{d}{\longleftarrow} & (winst_0, AN_0) & \overset{e}{\longrightarrow} & (winst', AN') \end{array}$$

(a) Direct transformation of weak instantiations

$$\begin{array}{ccccc} Net(\varrho): & L & \overset{l}{\longleftarrow} & I & \overset{r}{\longrightarrow} & R \\ & \downarrow m & (3) & \downarrow c & (4) & \downarrow n \\ & AN & \overset{d}{\longleftarrow} & AN_0 & \overset{e}{\longrightarrow} & AN' \end{array}$$

(b) Direct transformation of AHL-nets

Figure A.11: Direct transformation of weak instantiations and AHL-nets

Proof. **If.** Let $AN \overset{Net(\varrho),m}{\Longrightarrow} AN'$ be a direct transformation of AHL-nets with pushouts (3) and (4) in **AHLNets** as depicted in Figure A.11b. According to Fact 4.5.4, we can construct the restriction $(winst_0, AN_0)$ of $(winst, AN)$ along d, leading to the commutative cube shown in Figure A.12 in **PTNets**. Due to pushout (3) and Fact 3.4.4, the top of the cube is a pushout in **PTNets**. Moreover, *winst* is the unique morphism induced by the pushout in the top such that the front faces of the cube commute, and

hence, diagram (1) is a pushout in **wInst** according to the construction of pushouts of weak instantiations in Fact A.5.8. Finally, by Fact A.5.8 pushout (2) can be constructed in **wInst** which means that we have a direct transformation of weak instantiations $(winst, AN) \overset{\varrho,m}{\Longrightarrow} (winst', AN')$.

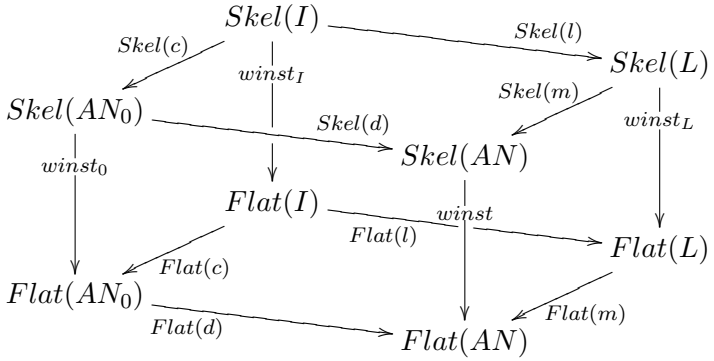


Figure A.12: Commuting cube in **PTNets**

Only If. Now, let $(winst, AN) \overset{\varrho,m}{\Longrightarrow} (winst', AN')$ be a direct transformation of weak instantiations with pushouts (1) and (2) in **wInst** shown in Figure A.11a. Then by Fact A.5.8 there are corresponding pushouts (3) and (4) in **AHLNets** shown in Figure A.11b which means that there is a direct transformation of AHL-nets $AN \overset{Net(\varrho),m}{\Longrightarrow} AN'$. □

Analogously to the direct transformation of weak instantiations above, also the existence of direct transformations of concrete instantiations depend only on the existence of the corresponding direct transformation in **AHLNets**, if the used production is a concrete production.

Fact A.5.12 (Direct Transformation of Instantiations Using Concrete Production)**.** *Given a concrete production for instantiations* $\varrho : (inst_L, L) \overset{l}{\leftarrow} (inst_I, I) \overset{r}{\rightarrow} (inst_R, R)$*, an instantiation* $(inst, AN)$ *and a (match) morphism* $m : (inst_L, L) \rightarrow (inst, AN)$*. There exists a direct transformation of instantiations* $(inst, AN) \overset{(\varrho,m)}{\Longrightarrow} (inst', AN')$*, iff there exists a direct transformation of AHL-nets* $AN \overset{Net(\varrho),m}{\Longrightarrow} AN'$ *using the underlying production for AHL-nets* $Net(\varrho) : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$*.*

$$\begin{array}{rccccc} \varrho: & (inst_L, L) & \overset{l}{\longleftarrow} & (inst_I, I) & \overset{r}{\longrightarrow} & (inst_R, R) \\ & {\scriptstyle m}\downarrow & (1) & {\scriptstyle c}\downarrow & (2) & {\scriptstyle n}\downarrow \\ & (inst, AN) & \overset{d}{\longleftarrow} & (inst_0, AN_0) & \overset{e}{\longrightarrow} & (inst', AN') \end{array}$$

(a) Direct transformation of instantiations

$$\begin{array}{rccccc} Net(\varrho): & L & \overset{l}{\longleftarrow} & I & \overset{r}{\longrightarrow} & R \\ & {\scriptstyle m}\downarrow & (3) & {\scriptstyle c}\downarrow & (4) & {\scriptstyle n}\downarrow \\ & AN & \overset{d}{\longleftarrow} & AN_0 & \overset{e}{\longrightarrow} & AN' \end{array}$$

(b) Direct transformation of AHL-nets

Figure A.13: Direct transformation of instantiations using concrete production

Proof. The construction and proof for this fact work completely analogously to the ones of Fact A.5.11, due to similar construction of pushouts in the categories **wInst** and **Inst** (see Fact A.5.8). □

Fact A.5.13 (Special Morphisms Between Instantiations)**.** *Given a (weak) instantiation morphism* $f : (inst_1, AN_1) \to (inst_2, AN_2)$*, then* f *is a monomorphism (resp. isomorphism) in* **Inst** *respectively* **wInst** *iff* f *is a monomorphism (resp. isomorphism) in* **AHLNets**.

Proof. We show the fact only for instantiation morphisms, since the proof for weak instantiation morphisms works completely analogously. So let $f : (inst_1, AN_1) \to (inst_2, AN_2)$ be an instantiation morphism.

First, we consider the case that f is a monomorphism in **AHLNets**. Then for instantiation morphisms $g, h : (inst_0, AN_0) \to (inst_1, AN_1)$ with $f \circ g = f \circ h$ we have that g and h are AHL-morphisms $g, h : AN_0 \to AN_1$, and thus, due to the fact that f is monomorphism in **AHLNets**, $f \circ g = f \circ h$ implies $g = h$.

Now, let f be a monomorphism in **Inst**, and let $g, h : AN_0 \to AN_1$ be AHL-morphisms with $f \circ g = f \circ h$. We can construct the restrictions $(inst_0, AN_0)$ and $(inst'_0, AN_0)$ of $(inst_1, AN_1)$ along g and h, respectively. Due to Fact 4.5.4 we know that $(inst_1, AN_1)$ is a restriction of $(inst_2, AN_2)$ along f. Then by definition of restrictions as pullbacks (Definition 4.5.3), and using pullback composition, we obtain that $(inst_0, AN_0)$ and $(inst'_0, AN_0)$ are restrictions of $(inst_2, AN_2)$ along $f \circ g = f \circ h$. Thus, by uniqueness of pullbacks we have that $(inst_0, AN_0) = (inst'_0, AN_0)$, and we have instantiation morphisms $g, h : (inst_0, AN_0) \to (inst_1, AN_1)$. Now, from $f \circ g = f \circ h$ and the fact that f is a monomorphism in **Inst**, it follows that $g = h$.

Next, we consider the case that f is an isomorphism in **AHLNets**. Then there exists an inverse isomorphism $g : AN_2 \to AN_1$ in **AHLNets**, and since functors preserve isomorphisms, we have

$$\begin{aligned}
Flat(g) \circ inst_2 &= Flat(g) \circ inst_2 \circ id_{Skel(AN_2)} \\
&= Flat(g) \circ inst_2 \circ Skel(id_{AN_2}) \\
&= Flat(g) \circ inst_2 \circ Skel(f \circ g) \\
&= Flat(g) \circ inst_2 \circ Skel(f) \circ Skel(g) \\
&= Flat(g) \circ Flat(f) \circ inst_1 \circ Skel(g) \\
&= Flat(g \circ f) \circ inst_1 \circ Skel(g) \\
&= Flat(id_{AN_1}) \circ inst_1 \circ Skel(g) \\
&= inst_1 \circ Skel(g)
\end{aligned}$$

which means that g is an instantiation morphism. Hence, f and g are also inverse isomorphisms in **Inst**.

Now, let f be an isomorphism in **Inst**. Then there is an inverse isomorphism g in **Inst**, and since f and g are AHL-morphisms, they are also isomorphisms in **AHLNets**. □

Fact A.5.14 (Restriction of Instantiation as Weak Instantiation)**.** *Given an instantiation* $(inst_2, AN_2)$ *and an AHL-morphism* $f : AN_1 \to AN_2$*. Then, for the restrictions* $(inst_1, AN_1)$ *of* $(inst_2, AN_2)$ *along* f*, and* $(winst_1, AN_1)$ *of* $W(inst_2, AN_2)$ *along* f*, we have* $W(inst_1, AN_1) = (winst_1, AN_1)$.

Proof. By Fact 4.5.4 and restriction $(inst_1, AN_1)$ of $(inst_2, AN_2)$ we have pullback (1) in Figure A.14, and by Fact 3.4.8 we have pullback (2) in Figure A.14. So, by pullback composition we have pullback (1)+(2) which by uniqueness of pullbacks means that $w(AN_1) \circ inst_1$ is the restriction of $W(inst_2, AN_2)$ along f. Hence, we have $w(AN_1) \circ inst_1 = winst_1$ and thus $W(inst_1, AN_1) = (winst_1, AN_1)$. □

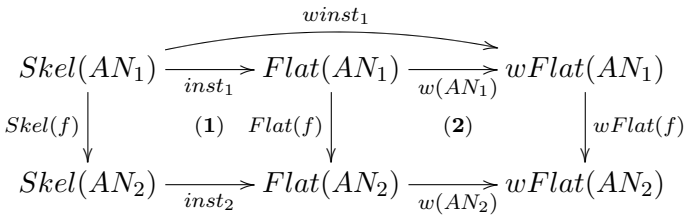


Figure A.14: Restriction of instantiation as weak instantiation

Lemma A.5.15 (Existence of Data-Images of Instantiations)**.** *Given a (weak) instantiation* $(inst_1, AN_1)$ *of an AHL-net* $AN_1 = (\Sigma_1, P_1, T_1, pre_1, post_1, cond_1, type_1, A_1)$*, and a generalised algebra homomorphism* $f = (f_\Sigma, f_A) : (\Sigma_1, A_1) \to (\Sigma_2, A_2)$*. Then the* data-image $\bar{f} : (inst_1, AN_1) \to (inst_2, AN_2)$ *of* $(inst_1, AN_1)$ *along* f *exists and* $(inst_1, AN_1)$ *is a (weak) instantiation.*

Proof. For the well-definedness of the construction, we have to show that $(inst_2, AN_2)$ is an instantiation, and that $\bar{f} : (inst_1, AN_1) \to (inst_2, AN_2)$ is an instantiation morphism. Due to Definition 3.3.1, for the P/T net $Skel(AN_2) = (P_2, T_2, pre_{2,S}, post_{2,S})$ we have $P_2 = P_1$, $T_2 = T_1$, $pre_{2,S} = pre_{1,S}$ and $post_{2,S} = post_{1,S}$ which means that $Skel(AN_2) = Skel(AN_1)$.

Moreover, for the morphism $\bar{f} = (\bar{f}_\Sigma, \bar{f}_P, \bar{f}_T, \bar{f}_A)$, we have that $\bar{f}_P$ and $\bar{f}_T$ are identities which by definition of $Skel(\bar{f})$ implies that $Skel(\bar{f}) = (\bar{f}_P, \bar{f}_T) = (id_{P_1}, id_{T_1}) = id_{Skel(AN_1)}$. Thus, using the facts that $proj$ is a natural transformation and that $(inst_1, AN_1)$ is an instantiation, we obtain

$$\begin{aligned} proj(AN_2) \circ inst_2 &= proj(AN_2) \circ Flat(\bar{f}) \circ inst_1 \\ &= Skel(\bar{f}) \circ proj(AN_1) \circ inst_1 \\ &= id_{Skel(AN_1)} \circ proj(AN_1) \circ inst_1 \\ &= id_{Skel(AN_1)} \\ &= id_{Skel(AN_2)} \end{aligned}$$

which means that $(inst_2, AN_2)$ is an instantiation. Further, we have

$$inst_2 \circ Skel(\bar{f}) = Flat(\bar{f}) \circ inst_1 \circ id_{Skel(AN_1)} = Flat(\bar{f}) \circ inst_1$$

which means that $\bar{f} : (inst_1, AN_1) \to (inst_2, AN_2)$ is an instantiation morphism.

The proof for the data-images of weak instantiations works completely analogously. □

Fact A.5.16 (Existence of Data-Shifting of Abstract Productions for Instantiations)**.** *Given an abstract production for instantiations* $\varrho : Inst_L \xleftarrow{l} Inst_I \xrightarrow{r} Inst_R$ *and a match* $m : Inst_L \to Inst$*, then the data-shifting* ϱ' *of* ϱ *along* m *as defined in Definition 4.6.5 exists.*

Proof. As shown in Corollary A.6.12, the functor $wInstData : \mathbf{wInst} \to \mathbf{Algs}$ has cocreations that are given by the data-image defined in Definition 4.4.1. Thus, according to Fact A.1.35, we have the existence of $wInstData$-shiftings of **wInst**-spans with isomorphic data type part. Since the rule-morphisms l and r are required to be $\mathcal{M}_{AHL}$-morphisms, they have an isomorphic data type part, and therefore the $wInstData$-shifting of ϱ along m exists.

Moreover, the definition of data-shiftings in Definition 4.6.5 is a special case of F-shiftings defined in Definition A.1.34 on page 186 using the functor $F = wInstData$. Hence, by Fact A.1.35 the data-shifting ϱ' of ϱ along m exists. □

Lemma A.5.17 (Data-Shifting and Direct Transformation)**.** *Given an abstract production for instantiations* $\varrho : Inst_L \overset{l}{\leftarrow} Inst_I \overset{r}{\rightarrow} Inst_R$*, a match* $m : Inst_L \rightarrow Inst$*, and let* ϱ' *together with* m' *be the data-shifting of* ϱ *along* m *as defined in Definition 4.6.5. Then for every direct transformation of weak instantiations* $Inst \overset{\varrho,m}{\Longrightarrow} Inst'$ *there exists a corresponding direct transformation of weak instantiations* $Inst \overset{\varrho',m'}{\Longrightarrow} Inst'$*.*

Proof. Given the abstract production for instantiations $\varrho : (inst_L, L) \overset{l}{\leftarrow} (inst_I, I) \overset{r}{\rightarrow} (inst_R, R)$, an instantiation $(inst, AN)$ and a (match) morphism $m : (inst_L, L) \rightarrow (inst, AN)$. Moreover, let $(inst, AN) \overset{\varrho,m}{\Longrightarrow} (inst', AN')$ be a direct transformation of weak instantiation given by pushouts (1) and (2) below.

$$\begin{array}{ccccc}
(inst_L, L) & \xleftarrow{\;l\;} & (inst_I, I) & \xrightarrow{\;r\;} & (inst_R, R) \\
{\scriptstyle m}\downarrow & (\mathbf{1}) & {\scriptstyle c}\downarrow & (\mathbf{2}) & {\scriptstyle n}\downarrow \\
(inst, AN) & \xleftarrow{\;d\;} & (inst_0, AN_0) & \xrightarrow{\;e\;} & (inst', AN')
\end{array}$$

Further, let $\varrho' : (inst'_L, L') \overset{l'}{\leftarrow} (inst'_I, I') \overset{r'}{\rightarrow} (inst'_R, R')$ together with a match morphism $m' : (inst'_L, L') \rightarrow (inst, AN)$ be the data-shifting of ϱ along m. Then we also have pushouts (3) and (4) in **wInst** below, and $m' \circ s_L = m$.

$$\begin{array}{ccccc}
(inst_L, L) & \xleftarrow{\;l\;} & (inst_I, I) & \xrightarrow{\;r\;} & (inst_R, R) \\
{\scriptstyle m}\downarrow\ {\scriptstyle s_L} & (\mathbf{3}) & {\scriptstyle c}\downarrow\ {\scriptstyle s_I} & (\mathbf{4}) & {\scriptstyle n}\downarrow\ {\scriptstyle s_R} \\
(inst'_L, L') & \xleftarrow{\;l'\;} & (inst'_I, I') & \xrightarrow{\;r'\;} & (inst'_R, R') \\
{\scriptstyle m'}\downarrow & & & & \\
(inst, AN) & \xleftarrow{\;d\;} & (inst_0, AN_0) & \xrightarrow{\;e\;} & (inst', AN')
\end{array}$$

Now, since we have $l, r \in \mathcal{M}_{AHL}$ and $\mathcal{M}_{AHL}$-morphisms are closed under pushouts, we also have $l', r', d, e \in \mathcal{M}_{AHL}$ which means that their data type parts are isomorphisms. This means that we also have a generalised algebra homomorphism $\bar{c} : wInstData(inst'_I, I') \rightarrow wInstData(inst_0, AN_0)$ given by $\bar{c} = wInstData(d)^{-1} \circ wInstData(m') \circ wInstData(l')$. A pushout in **wInst** means that there is a corresponding pushout of the underlying AHL-nets in **AHLNets** which in turn means that there is a corresponding pushout of the underlying data type part. Therefore we have

$$\begin{aligned}
\bar{c} \circ wInstData(s_I) &= wInstData(d)^{-1} \circ wInstData(m') \circ wInstData(l') \circ wInstData(s_I) \\
&= wInstData(d)^{-1} \circ wInstData(m') \circ wInstData(s_L) \circ wInstData(l) \\
&= wInstData(d)^{-1} \circ wInstData(m) \circ wInstData(l) \\
&= wInstData(d)^{-1} \circ wInstData(d) \circ wInstData(c) \\
&= wInstData(c)
\end{aligned}$$

So, due to the fact that s_I is a $wInstData$-cocreation, and there are morphisms $c : (inst_I, I) \rightarrow (inst_0, AN_0)$ in **wInst** and $\bar{c} : wInstData(inst'_I, I') \rightarrow wInstData(inst_0, AN_0)$ in **Algs** with $\bar{c} \circ wInstData(s_I) = wInstData(c)$ the universal property of cocreations implies a unique morphism $c' : (inst'_I, I') \rightarrow (inst_0, AN_0)$ such that $c' \circ s_I = c$ and $wInstData(c') = \bar{c}$.

The universal property of cocreation s_I also implies that there is a unique morphism $x : (inst'_I, I') \rightarrow (inst, AN)$ such that $x \circ s_I = m' \circ l' \circ s_I$ and $wInstData(x) = wInstData(m' \circ l')$.

Clearly, we have that $x = m' \circ l'$. Moreover, by commutativity of pushouts (1) and (3) we obtain

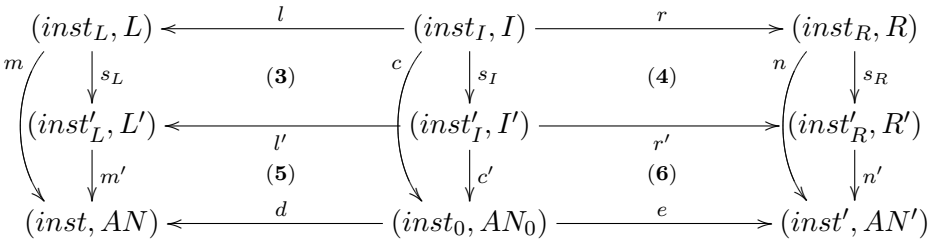
$$d \circ c' \circ s_I = d \circ c = m \circ l = m' \circ l' \circ s_I$$

and by definition of morphism $\bar{c}$ we obtain

$$\begin{aligned} wInstData(d \circ c') &= wInstData(d) \circ wInstData(c') \\ &= wInstData(d) \circ \bar{c} \\ &= wInstData(d) \circ wInstData(d)^{-1} \circ wInstData(m') \circ wInstData(l') \\ &= wInstData(m') \circ wInstData(l') \\ &= wInstData(m' \circ l') \end{aligned}$$

Thus, by uniqueness of x we have $d \circ c' = x = m' \circ l'$ which means that diagram (5) below commutes.

In a similar way, we obtain a morphism $n' : (inst'_R, R') \to (inst', AN')$ such that $n' \circ s_R = n$ and (6) commutes.



Now, we have equalities of diagrams (1)=(3)+(5) and (2)=(4)+(6) where diagrams (1)-(4) are pushouts. Thus, by pushout decomposition we obtain that also diagrams (5) and (6) are pushouts. Hence, there is a direct transformation of weak instantiations $(inst, AN) \overset{\varrho', m'}{\Longrightarrow} (inst', AN')$. □

A.6 Functor Creations and Cocreations of Processes and Instantiations

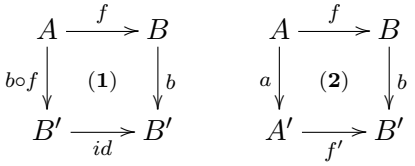
In this section we present concrete examples of functor creations in process and instantiation categories. Note that we show all facts only for instantiations and instantiated AHL-process, but not for weak instantiations and weakly instantiated AHL-processes. However, since instantiations and their weak counterparts share the same results which are used for the facts in this section, these facts do also hold for weak instantiations and weakly instantiated AHL-processes.

Fact A.6.1 (Creations in Arrow Categories)**.** *Given a category* $\mathbf{C}$ *and the arrow category* $\mathbf{C}^{\to}$*. Then the projection functor* $\pi_1 : \mathbf{C}^{\to} \to \mathbf{C}$ *has creations given by forward translation: The* π_1*-creation* $\bar{f}$ *of a* $\mathbf{C}$*-morphism* $f : A \to B$ *via* $\mathbf{C}^{\to}$*-object* $b : B \to B'$ *is given by* $\bar{f} = (f, id_{B'}) : b \circ f \to b$ *(see diagram (1) in Figure A.15).*

If $\mathbf{C}$ *has pullbacks (along* $\mathcal{M}$*), then* $\pi_2 : \mathbf{C}^{\to} \to \mathbf{C}$ *has creations (along* $\mathcal{M}$*) given by backward translation: The* π_2*-creation* $\bar{f}$ *of a* $\mathbf{C}$*-morphism* $f' : A' \to B'$ *via* $\mathbf{C}^{\to}$*-object* $b : B \to B'$ *is given by* $\bar{f} = (f, f') : a \to b$*, where* a *and* f *are obtained by pullback (2)* $\mathbf{C}$*, shown in Figure A.15.*

Proof. **Creations along** π_1**.** Given a $\mathbf{C}$-morphism $f : A \to B$ and $\mathbf{C}^{\to}$-object $b : B \to B'$.

We have to show that $\bar{f} = (f, id_{B'}) : id_A \to b$ is a π_1-creation of f via b. We have



(a) π_1-creation (b) π_2-creation

Figure A.15: Creations along projections of arrow category

$\pi_1(b \circ f) = A$ and $\pi_1(\bar{f}) = f$, and we have to show the universal property. So let $c : C \to C'$ be a $\mathbf{C}^{\to}$-object with morphism $g : c \to b$, and let $h : C \to A$ be a $\mathbf{C}$-morphism such that $f \circ h = \pi_1(g)$. We define a $\mathbf{C}^{\to}$-morphism $h^* : c \to b \circ f$ by $h^* = (h, \pi_2(g))$. The morphism h^* is a well-defined $\mathbf{C}^{\to}$-morphism, because due to the fact that g is a $\mathbf{C}^{\to}$-morphism, we have $b \circ f \circ h = b \circ \pi_1(g) = \pi_2(g) \circ c$. Moreover, we have that $\pi_1(h^*) = h$ and $\bar{f} \circ h^* = (f, id_{B'}) \circ (h, \pi_2(g)) = (f \circ h, id_{B'} \circ \pi_2(g)) = (\pi_1(g), \pi_2(g)) = g$.

It remains to show that h^* is unique. So let $k : c \to b \circ f$ be a $\mathbf{C}^{\to}$-morphism such that $\pi_1(k) = h$ and $\bar{f} \circ k = g$. Then we have $\pi_2(g) = \pi_2(\bar{f} \circ k) = \pi_2(\bar{f}) \circ \pi_2(k) = id_{B'} \circ \pi_2(k) = \pi_2(k)$ which together with $\pi_1(k) = h$ implies that $k = (h, \pi_2(g)) = h^*$. Hence, h^* is unique and thus $\bar{f}$ is a π_1-creation of f via b.

Creation along π_2. Given category $\mathbf{C}$ that has pullbacks (along $\mathcal{M}$), and a morphism $f' : A' \to B'$ (in $\mathcal{M}$). We construct the pullback (2) as in Figure A.15 in $\mathbf{C}$, and define a $\mathbf{C}^{\to}$ morphism $\bar{f} = (f, f')$ as above. We have to show that $\bar{f}$ is a π_2-creation. So let $c : C \to C'$ be a $\mathbf{C}^{\to}$-object with morphism $g : c \to b$, and let $h' : C' \to A'$ be a $\mathbf{C}$-morphism such that $f' \circ h' = \pi_2(g)$. Then due to the fact that g is $\mathbf{C}^{\to}$-morphism, we have $b \circ \pi_1(g) = \pi_2(g) \circ c = f' \circ h' \circ c$ which by pullback (2) implies that there exists a unique morphism $h : C \to A$ such that $a \circ h = h' \circ c$ and $f \circ h = \pi_1(g)$. This means that $h^* : c \to a$, defined by $h^* = (h, h')$, is a well-defined $\mathbf{C}^{\to}$-morphism, and we have $\pi_2(h^*) = h'$ and $\bar{f} \circ h^* = (f, f') \circ (h, h') = (f \circ h, f' \circ h') = (\pi_1(g), \pi_2(g)) = g$. It remains to show that h^* is unique, so let $k : c \to a$ be a $\mathbf{C}^{\to}$-morphism such that $\pi_2(k) = h'$ and $\bar{f} \circ k = g$. Then, by the fact that k is $\mathbf{C}^{\to}$-morphism, we have $a \circ \pi_1(k) = \pi_2(k) \circ c = h' \circ c$, and from $\bar{f} \circ k = g$ it follows that $f \circ \pi_1(k) = \pi_1(\bar{f}) \circ \pi_1(k) = \pi_1(\bar{f} \circ k) = \pi_1(g)$. Thus, from uniqueness of h it follows that $\pi_1(k) = h$, and hence $k = (\pi_1(k), \pi_2(k)) = (h, h') = h^*$. □

Analogously to the image construction based on the data type part of AHL-morphisms, we introduce a preimage construction based on the transition component.

Definition A.6.2 (T-Preimage of Algebraic High-Level Net)**.** Given an AHL-net $AN_2 = (\Sigma_2, P_2, T_2, pre_2, post_2, cond_2, type_2, A_2)$ and an injective function $f_T : T_1 \to T_2$. Then we define the *T-preimage* $f : AN_1 \to AN_2$ of AN_2 along f_T in the following way:

- $AN_1 = (\Sigma_1, P_1, T_1, pre_1, post_1, cond_1, type_1, A_1)$ is an AHL-net with
  - $\Sigma_1 = \Sigma_2$, $P_1 = P_2$, $A_1 = A_2$,
  - $op_1 = op_2 \circ f_T$ for $op \in \{pre, post, cond\}$, and
  - $type_1 = type_2$;
- $f : AN_1 \to AN_2 = (f_\Sigma, f_P, f_T, f_A)$ is an AHL-morphism with f_Σ, f_P and f_A being identities.

△

Proof of Well-definedness.

Well-definedness of AN_1. Due to the definition of $P_1 = P_2$ and $\Sigma_1 = \Sigma_2$, we have well-defined functions $pre_1 = pre_2 \circ f_T : T_1 \to T_{\Sigma_1}(X_1) \otimes P_1^{\oplus}$, $post_1 = post_2 \circ f_T : T_1 \to T_{\Sigma_1}(X_1) \otimes P_1^{\oplus}$, $cond_1 = cond_1 \circ f_T : T_1 \to \mathcal{P}_{fin}(Eqns(\Sigma_1; X_1))$, and also $type_1 = type_2 : P_1 \to S_1$. Thus, AN_1 is a well-defined AHL-net.

Well-definedness of f. We have to show that f satisfies the properties of an AHL-morphism in Definition 3.1.5.

1. By definition of f_Σ and f_P as identities and $pre_1 = pre_2 \circ f_T$, we obtain

$$(f_\Sigma^\# \otimes f_P)^\oplus \circ pre_1 = (id_{\Sigma_2}^\# \otimes id_{P_2})^\oplus \circ pre_2 \circ f_T = pre_2 \circ f_T$$

   and the same for the compatibility of the post domain functions.

2. Using $f_\Sigma = id_{\Sigma_2}$ and $cond_1 = cond_2 \circ f_T$, we obtain

$$cond_2 \circ f_T = cond_1 = \mathcal{P}_{fin}(id_{\Sigma_2}^\#) \circ cond_1 = \mathcal{P}_{fin}(f_\Sigma^\#) \circ cond_1$$

3. Using again $f_\Sigma = id_{\Sigma_2}$ and $f_P = id_{P_2}$, we have

$$type_2 \circ f_P = type_2 = type_1 = id_{\Sigma_2} \circ type_1 = f_\Sigma \circ type_1$$

□

Fact A.6.3 (T-Creations of Algebraic High-Level Nets)**.** *The functor T : **AHLNets** $\to$ **Sets**, mapping to the transition-component of AHL-nets and morphisms, has creations along the class $\mathcal{M}_S$ of all injective functions. The T-creation of a function $f_T : T_1 \to T_2$ via AHL-net AN_2 is given by the T-preimage $f : AN_1 \to AN_2$ of AN_2 along f_T as defined in Definition A.6.2. For an AHL-morphism $g : AN_0 \to AN_2$ and function $h_T : T(AN_0) \to T_1$ such that $f_T \circ h_T = T(g)$, the induced AHL-morphism $h : AN_0 \to AN_1$ is given by $h = (h_\Sigma, h_P, h_T, h_A)$ with $h_\Sigma = g_\Sigma$, $h_P = g_P$ and $h_A = g_A$.*

Proof. Let AN_0 be an AHL-net with AHL-morphism $g : AN_0 \to AN_2$ and function $h_T : T(AN_0) \to T_1$ such that $f_T \circ h_T = T(g)$. We define an AHL-morphism $h = (h_\Sigma, h_P, h_T, h_A) : AN_0 \to AN_1$ with $h_\Sigma = g_\Sigma$, $h_P = g_P$ and $h_A = g_A$. The domain and codomain of the single components of h fit to its definition, and we have to show that h satisfies the AHL-morphism properties in Definition 3.1.5.

1. Due to the fact that f and g are AHL-morphisms, we have

$$\begin{aligned}(h_\Sigma^\# \otimes h_P)^\oplus \circ pre_0 = (g_\Sigma^\# \otimes g_P)^\oplus \circ pre_0 = pre_2 \circ g_T = pre_2 \circ f_T \circ h_T \\ = (f_\Sigma^\# \otimes f_P)^\oplus \circ pre_1 \circ h_T = (id_{\Sigma_2}^\# \otimes id_{P_2})^\oplus \circ pre_1 \circ h_T = pre_1 \circ h_T\end{aligned}$$

   The proof for the post domain works analogously.

2. For the conditions, we have $cond_1 \circ h_T = cond_2 \circ f_T \circ h_T = cond_2 \circ g_T = cond_0$.

3. And for the type functions, we have $type_1 \circ h_P = type_2 \circ g_P = g_\Sigma \circ type_0 = h_\Sigma \circ type_0$.

Hence, h is a well-defined AHL-morphism, and we have $T(h) = h_T$. It remains to show the uniqueness of h. So let $h' : AN_0 \to AN_1 = (h'_\Sigma, h'_P, h'_T, h'_A)$ be an AHL-morphism such that $T(h') = h_T$ and $f \circ h' = g$. So, we already have that $h'_T = h_T$, and due to the fact that the Σ-, P- and A-components of f are identities, and due to the fact that commutativity $f \circ h' = g$ implies commutativity of its components, we obtain that $h' = h$. □

Fact A.6.4 (Representatives of T-Creations)**.** *Given an AHL-morphism $f : AN_1 \to AN_2 \in \mathcal{M}_{AHL}$ with an isomorphic place-component f_P. Then f is a T-creation (see Fact A.6.3).*

Proof. Since T has creations, and $f \in \mathcal{M}_{AHL}$ implies that f_T is injective, by Fact A.6.3 there is a T-creation $\bar{f} : \overline{AN}_1 \to AN_2$ of $T(f)$ via AN_2 with $\bar{f}_\Sigma$, $\bar{f}_P$ and $\bar{f}_A$ being identities, and $\bar{f}_T = T(f) = f_T$. Thus, since f_Σ, f_P and f_A are isomorphisms, we can define an AHL-morphism $i : \overline{AN}_1 \to AN_1 = (i_\Sigma, i_P, i_T, i_A)$ with $i_\Sigma = f_\Sigma^{-1}$, $i_P = f_P^{-1}$, $i_T = id_{T_1}$ and $i_A = f_A^{-1}$. We have to show that i is a well-defined AHL-morphism:

1. Considering the pre domains, and using compositionality of $(\_^\# \otimes \_)^\oplus$ shown in the appendix of [MGE+10], we obtain

$$\begin{aligned}(i_\Sigma^\# \otimes i_P)^\oplus \circ pre_{\bar{1}} &= ((f_\Sigma^{-1})^\# \otimes f_P^{-1})^\oplus \circ pre_2 \circ f_T = ((f_\Sigma^{-1})^\# \otimes f_P^{-1})^\oplus \circ (f_\Sigma^\# \otimes f_P)^\oplus \circ pre_1 \\ &= ((f_\Sigma^{-1} \circ f_\Sigma)^\# \otimes (f_P^{-1} \circ f_P))^\oplus \circ pre_1 = (id_{\Sigma_1}^\# \otimes id_{P_1})^\oplus \circ pre_1 \\ &= pre_1 \circ id_{T_1} = pre_1 \circ i_T\end{aligned}$$

   and the same for the post domain functions.

2. Considering the firing conditions, and using compositionality of $\mathcal{P}_{fin}(\_^\#)$ shown in the appendix of [MGE+10], we obtain

$$\begin{aligned}cond_1 \circ i_T = cond_1 &= \mathcal{P}_{fin}(id_{\Sigma_1}^\#) \circ cond_1 = \mathcal{P}_{fin}((f_\Sigma^{-1} \circ f_\Sigma)^\#) \circ cond_1 \\ &= \mathcal{P}_{fin}((f_\Sigma^{-1})^\#) \circ \mathcal{P}_{fin}(f_\Sigma^\#) \circ cond_1 = \mathcal{P}_{fin}((f_\Sigma^{-1})^\#) \circ cond_2 \circ f_T \\ &= \mathcal{P}_{fin}((f_\Sigma^{-1})^\#) \circ cond_{\bar{1}} = \mathcal{P}_{fin}(i_\Sigma^\#) \circ cond_{\bar{1}}\end{aligned}$$

3. Finally, considering the type functions, we obtain

$$\begin{aligned}type_1 \circ i_P = type_1 \circ f_P^{-1} &= f_\Sigma^{-1} \circ f_\Sigma \circ type_1 \circ f_P^{-1} = f_\Sigma^{-1} \circ type_2 \circ f_P \circ f_P^{-1} \\ &= f_\Sigma^{-1} \circ type_2 = i_\Sigma \circ type_{\bar{1}}\end{aligned}$$

Hence, i is a well-defined AHL-morphism, and since all its components are isomorphisms, it is an isomorphism in **AHLNets**. Thus, using Corollary A.1.19 it follows that also f is a T-creation of f_T via AN_2. □

Fact A.6.5 (Net-Creations of Instantiations)**.** *The functors Net : **Inst** $\to$ **AHLNets** and $wNet$: **wInst** $\to$ **AHLNets**, mapping (weak) instantiations and (weak) instantiation morphisms to their high-level net-component, have creations along the class $\mathcal{M}_{AHL}$ of injective AHL-morphisms with isomorphic data type part. The Net- respectively $wNet$-creation of an $\mathcal{M}_{AHL}$-morphism $f : AN_1 \to AN_2$ via (weak) instantiation $(inst_2, AN_2)$ is given by $f : (inst_1, AN_1) \to (inst_2, AN_2)$, where $(inst_1, AN_1)$ is the restriction of $(inst_2, AN_2)$ along f (see Definition 4.5.3). Given a (weak) instantiation $(inst_0, AN_0)$ together with instantiation morphism $g : (inst_0, AN_0) \to (inst_2, AN_2)$ and AHL-morphism $h : AN_0 \to AN_1$ such that $g = f \circ h$, the required unique instantiation morphism is given by $h : (inst_0, AN_0) \to (inst_1, AN_1)$.*

Proof. We prove the fact w. l. o. g. only for the functor Net, because the proof for $wNet$ works analogously. Given an $\mathcal{M}_{AHL}$-morphism $f : AN_1 \to AN_2$ together with an instantiation $(inst_2, AN_2)$. Due to Fact 4.5.4, there exists up to isomorphism a unique restriction $(inst_1, AN_1)$ of $(inst_2, AN_2)$ along f, such that we have pullback (1) shown in Figure A.16 in **PTNets**, and f is an instantiation morphism with $Net(f) = f$ and $Net(inst_1, AN_1) = AN_1$. Moreover, using the fact that g and f are instantiation morphisms, we have

$$\begin{aligned} Flat(f) \circ Flat(h) \circ inst_0 &= Flat(f \circ h) \circ inst_0 = Flat(g) \circ inst_0 \\ &= inst_2 \circ Skel(g) = inst_2 \circ Skel(f \circ h) \\ &= inst_2 \circ Skel(f) \circ Skel(h) = Flat(f) \circ inst_1 \circ Skel(h) \end{aligned}$$

which by monomorphism $Flat(f)$ (see Fact 3.4.11) implies that $Flat(h) \circ inst_0 = inst_1 \circ Skel(h)$, i. e. $h : (inst_0, AN_0) \to (inst_1, AN_1)$ is an instantiation morphism. The fact that $Net(h) = h$ and the uniqueness of h are trivially satisfied. □

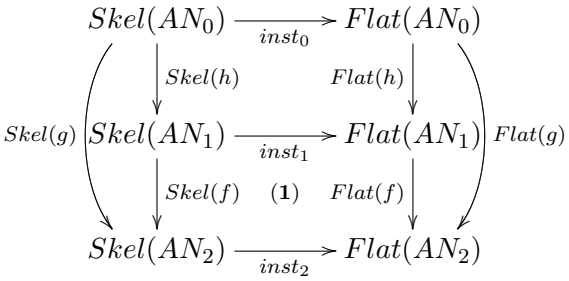


Figure A.16: Net-creation of instantiation

Fact A.6.6 (Creation of Processes and Instantiations)**.** *Consider the functors in Figure A.17.*

1. *The functors $\pi_1^{\to}$, π_1 and $Inst$ have creations,*
2. *the functors Net and $PNet$ have creations along $\mathcal{M}_{AHL}$ (see Fact A.1.11),*
3. *the functor $Proc$ has creations along $\mathcal{M}_{AHL}{}^{\to}$, where $\mathcal{M}_{AHL}{}^{\to} = \{(f, g) \mid f \in \mathcal{M}_{AHL}\}$,*
4. *the inclusion In_1 and the inclusion In_2 have creations along the class $\mathcal{T} = \{f = (f_\Sigma, f_P, f_T, f_A) \mid f_T \text{ is injective }\}$, and*
5. *the inclusion In_0 has creations along the class $\mathcal{T}^{\to} = \{(f, g) \mid f \in \mathcal{T}\}$.*

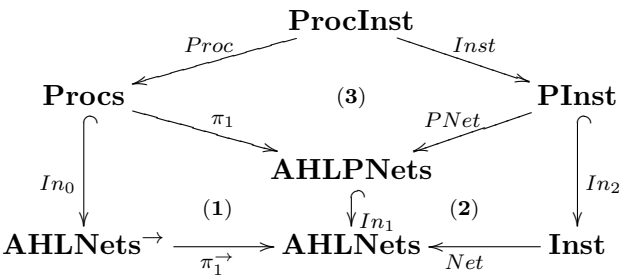


Figure A.17: Pullback diagrams of process and instantiation categories

Proof. 1. The fact that $\pi_1^{\rightarrow}$ has creations follows from Fact A.6.1. Thus, due to Fact A.1.39 and pullbacks (1) and (2) (see Fact A.1.41) also the functors π_1 and $Inst$ have creations.

2. The fact that Net has creations along $\mathcal{M}_{AHL}$ is shown in Fact A.6.5. Thus, due to Fact A.1.39 and pullback (2) (see Fact A.1.41) the functor $PNet$ has creations along $In_1^{-1}(\mathcal{M}_{AHL}) \subseteq \mathcal{M}_{AHL}$.

3. Analogously, due to pullback (3) the functor $Proc$ has creations along $\pi_1^{-1}(\mathcal{M}_{AHL}) = {\mathcal{M}_{AHL}}^{\rightarrow}$.

4. Since In_1 is a full inclusion and due to Lemma A.7.1, the In_1-creation of an AHL-morphism $f : AN \to K \in \mathcal{T}$ via AHL-process net K is trivially given by itself. Hence, due to pullback (2) also the inclusion In_2 has creations along $Net^{-1}(\mathcal{T}) = \mathcal{T}$.

5. Finally, since In_1 has creations along $\mathcal{T}$, due to pullback (1) also the inclusion In_0 has creations along $\pi_1^{\rightarrow -1}(\mathcal{T}) = \mathcal{T}^{\rightarrow}$.

□

Fact A.6.7 (*Sys*-Creations of AHL-Processes)**.** *The projection functor* Sys : **Procs** $\to$ **AHLNets**, *mapping AHL-processes* $mp : K \to AN$ *to their system part* AN *and AHL-process morphisms* (f', f) *to* f, *has creations along the class* $\mathcal{M}_{AHL}$ *of injective AHL-morphisms with isomorphic data type part. The* Sys*-creation of an AHL-morphism* $f : AN_1 \to AN_2$ *via AHL-process* $mp_2 : K_2 \to AN_2$ *is given by the restriction of* mp_2 *along* f *(see Definition 4.5.1). Given an AHL-process* $mp_0 : K_0 \to AN_0$ *together with an AHL-process morphism* $(g', g) : mp_0 \to mp_2$ *and AHL-morphism* $h : AN_0 \to AN_1$ *such that* $g = f \circ h$, *the required unique AHL-process morphism* $h^* : mp_0 \to mp_1$ *is given by* $h^* = (h', h)$ *where* h' *is induced by the pullback (1) in Figure A.18 that is obtained by restriction* mp_1 *of* mp_2.

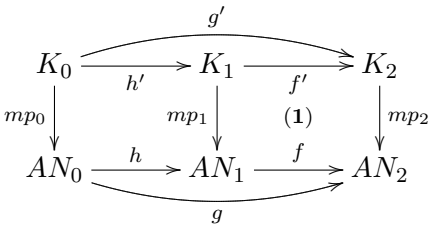


Figure A.18: *Sys*-creation of AHL-process

Proof. Using Fact A.6.1 and the fact that **AHLNets** has pullbacks along $\mathcal{M}_{AHL}$, we obtain that the projection $\pi_2^{\rightarrow}$: **AHLNets**$^{\rightarrow}$ $\to$ **AHLNets** has creations along $\mathcal{M}_{AHL}$ that can be constructed via pullback in **AHLNets**. Since $\mathcal{M}_{AHL}$-morphisms are closed under pullback, we have that the creation of an $\mathcal{M}_{AHL}$-morphism is an ${\mathcal{M}_{AHL}}^{\rightarrow}$-morphism, and we have ${\mathcal{M}_{AHL}}^{\rightarrow} \subseteq \mathcal{T}^{\rightarrow}$, where ${\mathcal{M}_{AHL}}^{\rightarrow} = \{(f, g) \mid f \in \mathcal{M}_{AHL}\}$ and $\mathcal{T}^{\rightarrow} = \{(f, g) \mid f_T \text{ is injective}\}$ (see Fact A.6.6). Moreover, from Fact A.6.6 we know that the inclusion In_0 : **Procs** $\to$ **AHLNets**$^{\rightarrow}$ has creations along $\mathcal{T}^{\rightarrow}$. Thus, using Corollary A.1.20 we have that $Sys = In_0 \circ \pi_2^{\rightarrow}$ has creations along $\mathcal{M}_{AHL}$. Finally, the construction of creations along π_2 in an arrow category (Fact A.6.1) corresponds exactly to the construction of the restriction of processes as pullback in **AHLNets** (Definition 4.5.1). □

Fact A.6.8 (*SysNet*-Creations of Instantiated AHL-Processes)**.** *The projection functor* $SysNet$: **ProcInst** $\to$ **AHLNets**, *mapping instantiated AHL-processes* $(inst, mp : K \to AN)$ *to their system part* AN *and instantiated AHL-process morphisms* (f', f) *to* f, *has*

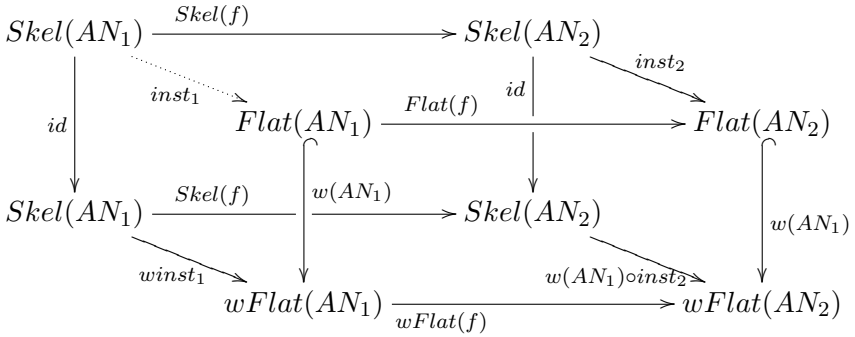
creations along the class $\mathcal{M}_{AHL}$ of injective AHL-morphisms with isomorphic data type part. The SysNet-creation of an instantiated AHL-morphism $f : AN_1 \to AN_2$ via instantiated AHL-process $(inst_2, mp_2 : K_2 \to AN_2)$ is given by the restriction of $(inst_2, mp_2)$ along f (see Definition 4.5.5).

Proof. From Fact A.6.7 we know that the projection functor $Sys : \mathbf{Proc} \to \mathbf{AHLNets}$ has creations along $\mathcal{M}_{AHL}$ that can be constructed as restriction of AHL-processes. Due to the fact that $\mathcal{M}_{AHL}$-morphisms are closed under pullbacks, it follows that Sys-creation of an $\mathcal{M}_{AHL}$-morphism is an ${\mathcal{M}_{AHL}}^{\to}$-morphism, where ${\mathcal{M}_{AHL}}^{\to} = \{(f, g) \mid f \in \mathcal{M}_{AHL}\}$. Thus, since the functor $Proc : \mathbf{ProcInst} \to \mathbf{Procs}$ has creations along ${\mathcal{M}_{AHL}}^{\to}$, by Corollary A.1.20 we obtain that the composition $SysNet = Sys \circ Proc$ has creations along $\mathcal{M}_{AHL}$ that are constructed as $Proc$-creation of a Sys-creation. According to Fact A.6.7 the Sys-creation of an AHL-morphism f via AHL-process mp can be constructed as restriction of mp along f. Moreover, considering the pullbacks (3)+(2) in Figure A.3 (see also Fact A.1.41), according to Fact A.1.39 a $Proc$-creation of an AHL-process morphism $(f' : K_1 \to K_2, f : AN_1 \to AN_2)$ via instantiated AHL-process $(inst_2, mp_2 : K_2 \to AN_2)$ is given by $(f', f) : (inst_1, mp_1) \to (inst_2, mp_2)$ where $f' : (inst_1, K_1) \to (inst_2, K_2)$ is the Net-creation (see Fact A.6.5) of f' via K_2 (taking into account the isomorphism I_3 in Fact A.1.41). According to Fact A.6.5, Net-creations can be constructed as restriction of instantiations, and hence, the $SysNet$-creation is constructed as restriction of instantiated AHL-process as defined in Definition 4.5.5. □

Fact A.6.9 (W-Creations of Weak Instantiations)**.** *The functors $W : \mathbf{Inst} \to \mathbf{wInst}$, $W_0 : \mathbf{PInst} \to \mathbf{wPInst}$ and $W_1 : \mathbf{ProcInst} \to \mathbf{wProcInst}$ from Fact A.5.3 have creations along the morphism classes $\mathcal{I}$ respectively $\mathcal{I}^{\to}$, where $\mathcal{I}$ is the class of all AHL-morphisms with isomorphic data-type part, and $\mathcal{I}^{\to}$ is the class of pairs of AHL-morphisms (f, g) where $f \in \mathcal{I}$.*

Proof. According to Fact A.1.39 and due to pullbacks (6) and (7) in Corollary A.1.42, it suffices to show that W has creations along $\mathcal{I}$. So let $(inst_2, AN_2)$ be an instantiation and $f : (winst_1, AN_1) \to W(inst_1, AN_2)$ be a weak instantiation morphism with $f \in \mathcal{I}$. This means that f has an isomorphic data type part and w. l. o. g. we can assume that the data type part of f is an identity, i. e. f can be considered to be an $\mathbf{AHLNets}(\mathbf{\Sigma}, \mathbf{A})$-morphism for a given data type part (Σ, A).

Then, we have that all faces in the diagram below commute, and since the right front face is a pullback by Fact 3.4.8, we obtain a unique induced $inst_1 : Skel(AN_1) \to Flat(AN_1)$ such that the whole cube commutes which means that $f : (inst_1, AN_1) \to (inst_2, AN_2)$ is an instantiation morphism. Moreover, f satisfies the universal property of being a W-creation of itself, and its uniqueness is guaranteed by injectivity of W.



□

Fact A.6.10 (*Data*-Cocreation of AHL-Nets)**.** *The projection functor $Data : \mathbf{AHLNets} \to \mathbf{Algs}$, that maps AHL-nets and AHL-morphisms to their data type part, has cocreations.*

Given an AHL-net AN_1 and a generalised algebra homomorphism $f : Data(AN_1) \to (\Sigma_2, A_2)$, the Data-cocreation $\bar{f} : AN_1 \to AN_2$ of f via AN_1 is given by the data-image of AN_1 along f (see Definition 3.3.1). For an AHL-net AN_3 with AHL-morphism $g = (g_\Sigma, g_P, g_T, g_A) : AN_1 \to AN_3$ and generalised algebra homomorphism $h = (h_\Sigma, h_A) : (\Sigma_2, A_2) \to Data(AN_3)$ such that $h \circ f = Data(g)$, the unique AHL-morphism $h^* : AN_2 \to AN_3$ is given by $h^* = (h_\Sigma, g_P, g_T, h_A)$.

Proof. Let AN_1 be an AHL-net with $Data(AN_1) = (\Sigma_1, A_1)$ and $f : (\Sigma_1, A_1) \to (\Sigma_2, A_2)$ a generalised algebra homomorphism. We have to show that the data-image $\bar{f} : AN_1 \to AN_2$ of AN_1 along f is a *Data*-cocreation. So let AN_3 be an AHL-net with $Data(AN_3) = (\Sigma_3, A_3)$ together with an AHL-morphism $g : AN_1 \to AN_3$ and a generalised algebra homomorphism $h : (\Sigma_2, A_2) \to (\Sigma_3, A_3)$ such that $h \circ f = Data(g)$. We define $h^* = (h^*_\Sigma, h^*_P, h^*_T, h^*_A)$ with $h^*_\Sigma = h_\Sigma$, $h^*_P = g_P$, $h^*_T = g_T$ and $h^*_A = h_A$, and we have to show that $h^* : AN_2 \to AN_3$ is a well-defined AHL-morphism. Note that $h \circ f = Data(g)$ implies that $h_\Sigma \circ f_\Sigma = g_\Sigma$ and $h_A \circ f_A = g_A$. Using the fact that g is an AHL-morphism, and that $(\_^\# \otimes \_)^\oplus$ is compositional as shown in Lemma A.2 in [MGE+10], we obtain for the pre conditions:

$$\begin{aligned} pre_3 \circ h^*_T &= pre_3 \circ g_T = (g^\#_\Sigma \otimes g_P)^\oplus \circ pre_1 = ((h_\Sigma \circ f_\Sigma)^\# \otimes (g_P \circ id_{P_1}))^\oplus \circ pre_1 \\ &= ((h^*_\Sigma \circ \bar{f}_\Sigma)^\# \otimes (h^*_P \circ \bar{f}_P))^\oplus \circ pre_1 = (h^{*\#}_\Sigma \otimes h^*_P)^\oplus \circ (\bar{f}^\#_\Sigma \otimes \bar{f}_P)^\oplus \circ pre_1 \\ &= (h^{*\#}_\Sigma \otimes h^*_P)^\oplus \circ pre_2 \circ \bar{f}_T = (h^{*\#}_\Sigma \otimes h^*_P)^\oplus \circ pre_2 \end{aligned}$$

The proof for the compatibility of the post conditions works analogously. Moreover, using the compositionality of $\mathcal{P}_{fin}(\_^\#)$ shown in Lemma A.3 in [MGE+10], we obtain for the firing conditions:

$$\begin{aligned} cond_3 \circ h^*_T &= cond_3 \circ g_T = \mathcal{P}_{fin}(g^\#_\Sigma) \circ cond_1 = \mathcal{P}_{fin}((h_\Sigma \circ f_\Sigma)^\#) \circ cond_1 \\ &= \mathcal{P}_{fin}(h^\#_\Sigma) \circ \mathcal{P}_{fin}(f^\#_\Sigma) \circ cond_1 = \mathcal{P}_{fin}(h^\#_\Sigma) \circ \mathcal{P}_{fin}(\bar{f}^\#_\Sigma) \circ cond_1 \\ &= \mathcal{P}_{fin}(h^\#_\Sigma) \circ cond_2 \circ \bar{f}_T = \mathcal{P}_{fin}(h^{*\#}_\Sigma) \circ cond_2 \end{aligned}$$

Finally, for the type functions, due to compositionality of the free extension $\_^\#$, we have

$$\begin{aligned} type_3 \circ h^*_P &= type_3 \circ g_P = g^\#_\Sigma \circ type_1 = (h_\Sigma \circ f_\Sigma)^\# \circ type_1 = (h^*_\Sigma \circ \bar{f}_\Sigma)^\# \circ type_1 \\ &= h^{*\#}_\Sigma \circ \bar{f}^\#_\Sigma \circ type_1 = h^{*\#}_\Sigma \circ type_2 \circ \bar{f}_P = h^{*\#}_\Sigma \circ type_2 \end{aligned}$$

Hence, h^* is a well-defined AHL-morphism. It remains to show that h^* is the unique required morphism to satisfy the universal property of F-cocreations. We have

$$\begin{aligned} h^* \circ \bar{f} &= (h^*_\Sigma, h^*_P, h^*_T, h^*_A) \circ (\bar{f}_\Sigma, \bar{f}_P, \bar{f}_T, \bar{f}_A) = (h_\Sigma, g_P, g_T, h_A) \circ (f_\Sigma, id_{P_1}, id_{T_1}, f_A) \\ &= (h_\Sigma \circ f_\Sigma, g_P \circ id_{P_1}, g_T \circ id_{T_1}, h_A \circ f_A) = (g_\Sigma, g_P, g_T, g_A) = g \end{aligned}$$

In order to show the uniqueness of h^*, let $h' : AN_2 \to AN_3$ be an AHL-morphism with $Data(h') = h$ and $h' \circ \bar{f} = g$. Then we have $(h^*_\Sigma, h^*_A) = Data(h^*) = h = Data(h') = (h'_\Sigma, h'_A)$ and since $\bar{f}_P$ and $\bar{f}_T$ are identities, it also follows that $h^*_P = g_P = h'_P$ and $h^*_T = g_T = h'_T$. Hence, h^* is unique and satisfies the universal property which means that $\bar{f}$ is a *Data*-cocreation. □

Fact A.6.11 (*Net*-Cocreations of Instantiations)**.** *The functors Net : **Inst** $\to$ **AHLNets** and $wNet$: **wInst** $\to$ **AHLNets** have cocreations along Data-cocreations. Given an instantiation $(inst_1, AN_1)$ and an AHL-morphism $f : AN_1 \to AN_2$ that is a Data-cocreation, then the Net-cocreation of f via $(inst_1, AN_1)$ is given by the data-image $f : (inst_1, AN_1) \to (inst_2, AN_2)$ of $(inst_1, AN_1)$ along $Data(f)$ (see Definition 4.4.1). Similar, the $wNet$-cocreation of a weak instantiation can be obtained as a data-image.*

Proof. Let $(inst_1, AN_1)$ be an instantiation and $f : AN_1 \to AN_2$ an AHL-morphism that is a $Data$-cocreation. Due to the uniqueness of cocreations, we have that f is the $Data$-cocreation of $Data(f)$ which means that $f : AN_1 \to AN_2$ itself is the data-image of AN_1 along $Data(f)$. Moreover, according to Definition 4.4.1, $inst_2$ is defined by $inst_2 = Flat(f) \circ inst_1$.

Let $(inst_3, AN_3)$ be an instantiation with an instantiation morphism $g : (inst_1, AN_1) \to (inst_3, AN_3)$ and an AHL-morphism $h : AN_2 \to AN_3$ such that $h \circ f = g$. Clearly, the unique required instantiation morphism $h^* : (inst_2, AN_2) \to (inst_3, AN_3)$ is given by $h^* = h$, because we have

$$\begin{aligned} Flat(h) \circ inst_2 &= Flat(h) \circ Flat(f) \circ inst_1 = Flat(h \circ f) \circ inst_1 \\ &= Flat(g) \circ inst_1 = inst_3 \circ Skel(g) \\ &= inst_3 \circ Skel(h \circ f) = inst_3 \circ Skel(h) \circ Skel(f) = inst_3 \circ Skel(h) \end{aligned}$$

which means that h is an instantiation morphism. □

Corollary A.6.12 ($InstData$-Cocreation of Instantiations)**.** *The functors $InstData = Data \circ Net$: **Inst** $\to$ **Algs** and $wInstData = Data \circ wNet$: **wInst** $\to$ **Algs** have cocreations. Given an instantiation $(inst_1, AN_1)$ and a generalised algebra homomorphism $f : Data(AN_1) \to (\Sigma_2, A_2)$, then the $InstData$-cocreation of f via $(inst_1, AN_1)$ is given by the data-image $\bar{f} : (inst_1, AN_1) \to (inst_2, AN_2)$ of $(inst_1, AN_1)$ along f (see Definition 4.4.1). Similar, the $wInstData$-cocreation of a weak instantiation can also be obtained as data-image.*

Proof. Since $Data$ has cocreations and Net has cocreations along $Data$-cocreations, due to Corollary A.1.32 the composition $InstData = Data \circ Net$ has cocreations. The construction as data-image corresponds to the construction of Net-cocreations (Fact A.6.11) of $Data$-cocreations (Fact A.6.10). □

A.7 Minor Technical Results

The properties of AHL-process nets are reflected by AHL-morphisms with an injective transition part, as stated in the following lemma.

Lemma A.7.1 (Reflection of AHL-Process Nets)**.** *Given an AHL-morphism $f : K_1 \to K_2 = (f_\Sigma, f_P, f_T, f_A)$ where f_T is injective. If K_2 is an AHL-process net then also K_1.*

Proof. Given an AHL-morphism $f : K_1 \to K_2 = (f_\Sigma, f_P, f_T, f_A)$ where f_T is injective, and an AHL-process net K_2. In order to show that K_1 is an AHL-process net we have to show that it is unary, there are no forward or backward conflicts and the causal relation $<_{K_1}$ is a strict partial order.

Unarity. Let us assume that K_1 is not unary, i. e. there are $p \in P_{K_1}$, $t \in T_{K_1}$ with

$$(term_1, p) \oplus (term_2, p) \leq pre_{K_1}(t) \text{ or } (term_1, p) \oplus (term_2, p) \leq post_{K_1}(t)$$

Let $(term_1, p) \oplus (term_2, p) \leq pre_{K_1}(t)$.
Since AHL-morphisms preserve pre conditions there is

$$(f_\Sigma^\sharp \otimes f_P)^\oplus \circ pre_{K_1}(t) = pre_{K_2}(f_T(t))$$

and hence

$$\begin{aligned} (f_\Sigma^\sharp(term_1), f_P(p)) \oplus (f_\Sigma^\sharp(term_2), f_P(p)) &= (f_\Sigma^\sharp \otimes f_P)^\oplus((term_1, p) \oplus (term_2, p)) \\ &\leq pre_{K_2}(f_T(t)) \end{aligned}$$

This implies that K_2 is not unary, contradicting the fact that K_2 is an AHL-process net.
The case that $(term_1, p) \oplus (term_2, p) \leq post_{K_1}(t)$ works analogously. Hence K_1 is unary.

No forward conflict. Let us assume that K_1 has a forward conflict, i. e. there is $p \in P_{K_1}$, $t_1 \neq t_2 \in T_{K_1}$ with $p \in \bullet t_1 \cap \bullet t_2$. This means that there are $term_1, term_2 \in T_{OP}(X)_{type(p)}$ such that

$$(term_1, p) \leq pre_{K_1}(t_1) \text{ and } (term_2, p) \leq pre_{K_1}(t_2)$$

and since AHL-morphisms preserve pre and post conditions we obtain

$$\begin{aligned} (f_\Sigma^\sharp(term_1), f_P(p)) &= (f_\Sigma^\sharp \otimes f_P)^\oplus(term_1, p) \\ &\leq pre_{K_2}(f_T(t_1)) \end{aligned}$$

and

$$\begin{aligned} (f_\Sigma^\sharp(term_2), f_P(p)) &= (f_\Sigma^\sharp \otimes f_P)^\oplus(term_2, p) \\ &\leq pre_{K_2}(f_T(t_2)) \end{aligned}$$

In the case that $f_T(t_1) \neq f_T(t_2)$ the fact that $f_P(p) \in \bullet f_T(t_1) \cap \bullet f_T(t_2)$ means that K_2 has a forward conflict, contradicting the fact that K_2 is an AHL-process net. In the case that $f_T(t_1) = f_T(t_2)$, the injectivity of f_T implies that $t_1 = t_2$, contradicting the assumption that $t_1 \neq t_2$.

Hence K_1 has no forward conflict.

No backward conflict. The proof for this case works analogously to the one for forward conflicts because AHL-morphisms preserve post as well as pre conditions and K_2 has no backward conflicts.

Strict partial order. We have to show that $<_{K_1}$ is irreflexive. So, let us assume that $<_{K_1}$ is not irreflexive, i. e. there exists a cycle $x <_{K_1} x$. This implies $f(x) <_{K_2} f(x)$ because AHL-morphisms preserve pre and post conditions. This contradicts the fact that $<_{K_2}$ is irreflexive because it is an AHL-process net.

Hence, also $<_{K_1}$ is also irreflexive and thus it is an AHL-process net.

□

Considering the fact that the firing behaviour of AHL-nets relies on the evaluation of terms and equations, it is interesting to note that homomorphisms preserve terms and equations (see Theorems 3.3.2 and 4.12.2 in [EM85]). This means that given assignments $v_A : X \to A$ and $v_B : X \to B$ and a homomorphism $f : A \to B$ such that $f \circ v_A = v_B$, by Theorem 3.3.2 in [EM85] we also have $f \circ v_A^* = v_B^*$ (which especially means that $(f_A \circ v_A)^* = f_A \circ v_A^*$), and if (A, v_A) satisfies an equation e, then by Theorem 4.12.2 in [EM85] also $(B, v_B) = (B, f \circ v_A)$ satisfies that equation.

Corollary A.7.2 (Basic Data-Images Preserve Firing Steps)**.** *Given a basic data-image $f : AN_1 \to AN_2$ along a Σ-homomorphism $f_A : A_1 \to A_2$ and a firing step $M_1 \xrightarrow{(t,v)} M_2$ in AN_1. Then there is a corresponding firing step $(f_A \otimes id_P)^\oplus(M_1) \xrightarrow{(t, f_A \circ v)} (f_A \otimes id_P)^\oplus(M_2)$ in AN_2.*

Proof. Due to firing step $M_1 \xrightarrow{(t,v)} M_2$ in AN_1, we have $pre_{1,A}(t,v) \leq M_1$. Note that we have $f_T(t) = t$, and by Definition 3.1.7 we have $pre_{1,A}(t,v) = \sum_{i=1}^{n}(v^*(term_i), p_i)$ for $pre_1(t) = \sum_{i=1}^{n} p_i = pre_2(t)$. Thus, using Theorem 3.3.2 in [EM85], we have

$$\begin{aligned} pre_{2,A}(t, f_A \circ v) &= \sum_{i=1}^{n}(f_A \circ v^*(term_i), p_i) \\ &= (f_A \otimes id_P)^{\oplus}(\sum_{i=1}^{n}(v^*(term_i), p_i)) \\ &= (f_A \otimes id_P)^{\oplus}(pre_{1,A}(t,v)) \leq (f_A \otimes id_P)^{\oplus}(M_1) \end{aligned}$$

which means that $(t, f_A \circ v)$ is enabled under $(f_A \otimes id_P)^{\oplus}(M_1)$. Moreover, using Theorem 4.12.2 in [EM85], we know that $(A_2, f_A \circ v)$ satisfies all equations in $cond_2(t)$, because (A_1, v) satisfies all equations in $cond_1(t)$, and we have $cond_1(t) = cond_2(t)$. Finally, using again Theorem 3.3.2 in [EM85] and the fact that $post_2(t) = post_1(t)$, we obtain analogously to above that $post_{2,A}(t, f_A \circ v) = (f_A \otimes id_P)^{\oplus}(post_{1,A}(t,v))$. Thus, we get the follower marking

$$\begin{aligned} (f_A \otimes id_P)^{\oplus}(M_2) &= (f_A \otimes id_P)^{\oplus}(M_1 \ominus pre_{1,A}(t,v) \oplus post_{1,A}(t,v)) \\ &= (f_A \otimes id_P)^{\oplus}(M_1) \ominus (f_A \otimes id_P)^{\oplus}(pre_{1,A}(t,v)) \oplus (f_A \otimes id_P)^{\oplus}(post_{1,A}(t,v)) \\ &= (f_A \otimes id_P)^{\oplus}(M_1) \ominus pre_{2,A}(t, f_A \circ v) \oplus post_{2,A}(t, f_A \circ v) \end{aligned}$$

and, hence, we have a firing step $(f_A \otimes id_P)^{\oplus}(M_1) \xrightarrow{(t, f_A \circ v)} (f_A \otimes id_P)^{\oplus}(M_2)$ in AN_2. □

Lemma A.7.3 (Application of Action Evolution Pattern)**.** *Given an action evolution pattern $\varrho : L \xleftarrow{l} I \xrightarrow{r} R$ for AHL-processes, an AHL-process net K and a match $m : L \to K$, where m_T is injective. Then there exists a direct transformation $K \xRightarrow{\varrho,m} K'$ of AHL-process nets.*

Proof. First, we show that the direct transformation exists for single action evolution patterns. So let $\varrho : L \xleftarrow{l} I \xrightarrow{r} R$ be a single action evolution pattern together with an AHL-process net K and a match $m : L \to K$. In order to show that the direct transformation exists, according to Theorem 4.2.11 we have to show that ϱ satisfies the transformation condition in Definition 4.2.10 under match m.

1. **Gluing Condition.** We have to show that all dangling points DP and identification points IP are also gluing points GP (see Definition 3.2.13). Note that all dangling points are places, and since ϱ is non-deleting on places, we have $DP \subseteq l_P(P_I) \subseteq GP$. Moreover, since m_T is injective, also all identification points are places, and therefore we also have $IP \subseteq l_P(P_I) \subseteq GP$, and thus the gluing condition is satisfied.

2. **No Cycles.** We have to show that the gluing condition $<_{(\varrho,m)}$ is a strict partial order, i. e. it is irreflexive. For this purpose we show that for the causal relations of the left- and right-hand side of ϱ the following implication holds:

   $$\forall x, y \in P_I \uplus T_I : r(x) <_R r(y) \Rightarrow l(x) <_L l(y)$$

   Let $x, y \in P_I \uplus T_I$ with $r(x) <_R r(y)$. We distinguish the following cases:

   Case 1. $x \in P_I, y \in T_I$.
   From the single action condition in Definition 4.8.8 we know that $T_L = \{t_\varrho\}$ which means that $y \in T_I$ implies $T_I = \{y\}$ with $l(y) = t_\varrho$ because l is injective. Moreover, $r(x) <_R r(y)$ means that $r(x) \notin OUT(R)$ which by contraposition of condition 3 in Definition 4.8.8 implies that $l(x) \notin OUT(L)$. Then, since all places of L are in the environment of t_ϱ, we obtain that $l(x) \in {\bullet t_\varrho} = l(y)$ and thus $l(x) <_L l(y)$.

Case 2. $x \in T_I, y \in P_I$.
This case works analogously to the first case.

Case 3. $x, y \in T_I$.
As pointed out in the first case, the single action condition of action evolution patterns implies that there is at most one transition in I which means that we have $x = y$ and thus $r(x) = r(y)$ which together with $r(x) <_R r(y)$ contradicts the fact that $<_R$ is irreflexive because R is an AHL-process net.

Case 4. $x, y \in P_I$.
Then from $r(x) <_R r(y)$ we know that $r(x) \notin OUT(R)$ and $r(y) \notin IN(R)$ which by contraposition of condition 3 in Definition 4.8.8 implies that $l(x) \notin OUT(L)$ and $l(y) \notin IN(L)$. Thus by condition 1 in Definition 4.8.8 we have that $l(x) \in {\bullet}t_\varrho$ and $l(y) \in t_\varrho{\bullet}$ which by transitivity of $<_L$ implies that $l(x) <_L l(y)$.

So the above statement holds for all $x, y \in P_I \uplus T_I$.

Now, let

$$\prec_K = \{(x, y) \in (P_K \times T_K) \uplus (P_K \times T_K) \mid x \in {\bullet}y\}$$

and consider the relation $\prec_{(\varrho,m)}$ with

$$\prec_{(\varrho,m)} = \{(x, y) \in (P_I \times T_I) \uplus (T_I \times P_I) \mid m \circ l(x) <_{(K,m)} m \circ l(y) \vee r(x) <_R r(y)\}$$

defined in Definition 4.2.9. By definition of $\prec_{(K,m)}$ as

$$\prec_{(K,m)} = \{(x, y) \in (P_K \times (T_K \setminus m_T(T_L))) \uplus ((T_K \setminus m_T(T_L)) \times P_K) \mid x \in {\bullet}y\}$$

in Definition 4.2.9, we have for $x, y \in P_I \uplus T_I$ with $m \circ l(x) <_{(K,m)} m \circ l(y)$ that $m \circ l(x) <_K m \circ l(x)$, because $<_{(K,m)}$ is the transitive closure of $\prec_{(K,m)}$ and $<_K$ is the transitive closure of $\prec_K$. Moreover, as shown above, we have for $x, y \in P_I \uplus T_I$ with $r(x) <_R r(y)$ that we have also $l(x) <_L l(y)$, and since AHL-morphisms preserve pre and post conditions, we also have $m \circ l(x) <_K m \circ l(y)$. Therefore, for any $x, y \in P_I \uplus T_I$ with $x \prec_{(\varrho,m)} y$, it follows that we also have $m \circ l(x) <_K m \circ l(y)$, and since $<_{(\varrho,m)}$ is the transitive closure of $\prec_{(\varrho,m)}$, the same follows if we have $x <_{(\varrho,m)} y$.

So, let us assume that $<_{(\varrho,m)}$ is not irreflexive. Then there is $x \in P_I \uplus T_I$ with $x <_{(\varrho,m)} x$, implying $m \circ l(x) <_K m \circ l(x)$, contradicting the fact that $<_K$ is irreflexive, beause K is an AHL-process net. Hence, $<_{(\varrho,m)}$ is a strict partial order.

3. **Non-injective Gluing.** We have to show that for all $p_1 \neq p_2 \in IN(I)$ with $m \circ l(p_1) = m \circ l(p_2)$ we have $r(p_1) \in IN(R)$ or $r(p_2) \in IN(R)$. So let $p_1 \neq p_2 \in IN(I)$ with $m \circ l(p_1) = m \circ l(p_2)$ and let us assume that $r(p_1), r(p_2) \notin IN(R)$. Then from contraposition of condition 3 in Definition 4.8.8 we obtain that $l(p_1), l(p_2) \notin IN(L)$ which by condition 1 in Definition 4.8.8 implies that $l(p_1), l(p_2) \in {\bullet}t_\varrho$, i. e. there are terms $term_1$ and $term_2$ such that $(term_1, l_P(p_1)) \oplus (term_2, l_P(p_2)) \leq pre_L(t_\varrho)$. Due to the fact that AHL-morphisms preserve pre conditions, we obtain $(m_\Sigma^\#(term_1), m_P(l_P(p_1))) \oplus (m_\Sigma^\#(term_2), m_P(l_P(p_2))) \leq pre_K(m_T(t_\varrho))$, which together with the fact that $m \circ l(p_1) = m \circ l(p_2)$ implies that K is not unary. This is a contradiction because K is an AHL-process net. Hence, we have $r(p_1) \in IN(R)$ or $r(p_2) \in IN(R)$.

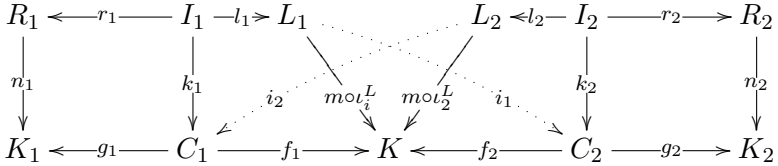
The proof that for all $p_1 \neq p_2 \in OUT(I)$ with $m \circ l(p_1) = m \circ l(p_2)$ we have $r(p_1) \in OUT(R)$ or $r(p_2) \in OUT(R)$ works analogously.

4. **No Conflicts.** For every place $x \in InP$ we have $l(x) \in IN(L)$ which by condition 3 of Definition 4.8.8 implies that $r(x) \in IN(R)$, and thus $r(InP) \subseteq IN(R)$. Accordingly, we obtain that $r(OutP) \subseteq OUT(R)$.

So the single action evolution pattern ϱ satisfies the transformation condition under match m which by Theorem 4.2.11 implies that the direct transformation $K \overset{\varrho,m}{\Longrightarrow} K'$ exists.

Now, we consider ϱ to be a (multi) action evolution pattern, i. e. there are single action evolution patterns $(\varrho_i : L_i \overset{l}{\leftarrow} I_i \overset{r}{\rightarrow} R_i)_{i\in\mathcal{I}}$ such that $\varrho = \coprod_{i\in\mathcal{I}} \varrho_i$ with coproduct injections $\iota_i^X : X_i \to X$ for $X \in \{L, I, R\}$. Due to the fact that for every $i \in \mathcal{I}$ there is $T_{L_i} = \{t_{\varrho_i}\}$ it is clear that all $\iota_{i,T}^L$ are injective and therefore also $(m \circ \iota_i^L)_T$ is injective. Therefore, as shown above, we know that all ϱ_i are applicable with match $m \circ \iota_i^L$. According to the parallelism theorem for weak adhesive HLR categories (see Theorem 5.18 in [EEPT06b]) which also holds for $\mathcal{M}$-adhesive categories [EGH10], for the existence of a direct transformation via a parallel production, it suffices to show that the direct transformations via the single productions are sequentially independent. Further, due to the local Church-Rosser theorem (see Theorem 5.12 in [EEPT06b]), sequential independence is equivalent to the case that there are corresponding parallel independent direct transformations.

Let ϱ_1, ϱ_2 be an arbitrary selection of two single action evolution patterns in $(\varrho_i : L_i \overset{l}{\leftarrow} I_i \overset{r}{\rightarrow} R_i)_{i\in\mathcal{I}}$. Parallel independence of two direct transformations $K \overset{m\circ\iota_1^L}{\Longrightarrow} K_1$ and $K \overset{m\circ\iota_2^L}{\Longrightarrow} K_2$ means that there are morphisms $i_1 : L_1 \to C_2$ and $i_2 : L_2 \to C_1$ into the context net of the respective other transformation as shown below such that $f_2 \circ i_1 = m \circ \iota_1^L$ and $f_1 \circ i_2 = m \circ \iota_2^L$.



Due to symmetry it suffices to show the existence of i_1. For the places, due to the fact that ϱ_2 is non-deleting on places by condition 2 in Definition 4.8.8, and thus for every $p \in P_{L_1}$ and $m \circ \iota_1^L(p) \in P_K$ there is also $p' \in P_{C_2}$ such that $f_2(p') = m \circ \iota_1^L(p)$. Moreover, by $l_2 \in \mathcal{M}_{AHL}$ we also have $f_2 \in \mathcal{M}_{AHL}$ which means that $f_{2,P}$ is injective and thus $p' \in P_{C_2}$ is unique such that $f_2(p') = m \circ \iota_1^L(p)$. So, we obtain a well-defined function $i_{1,P} : P_{L_1} \to P_{C_2}$ with $i_{1,P}(p) = p'$.

Further, due to the single action condition in Definition 4.8.8 there is $T_{L_1} = \{t_{\varrho_1}\}$ and $T_{L_2} = \{t_{\varrho_2}\}$, and due to construction of the coproducts as disjoint union, we have $\iota_1^L(t_{\varrho_1}) \neq \iota_2^L(t_{\varrho_2})$, and since m_T is required to be injective, we also have $m \circ \iota_1^L(t_{\varrho_1}) \neq m \circ \iota_2^L(t_{\varrho_2})$. Thus, the transition $m \circ \iota_1^L(t_{\varrho_1})$ is not deleted by ϱ_2, and can be matched in the subnet $C_2 \subseteq K$ such that we obtain an AHL-morphism $i_1 : L_1 \to C_2$ with $f_2 \circ i_1 = m \circ \iota_1^L$. Hence, the direct transformations $K \overset{m\circ\iota_1^L}{\Longrightarrow} K_1$ and $K \overset{m\circ\iota_2^L}{\Longrightarrow} K_2$ are parallel independent.

So we have that all single action evolution patterns are applicable with match $m \circ \iota_i^L$, and all single direct transformations via $(\varrho_i)_{i\in\mathcal{I}}$ are pairwise parallel independent. Using the local Church-Rosser theorem, we know that all sequences of direct transformations via all single action evolution patterns $(\varrho_i)_{i\in\mathcal{I}}$ lead to the same result which is an AHL-process net as shown above. Moreover, using iterated construction of parallel productions and the parallelism theorem, we obtain that also the direct transformation via the parallel production ϱ leads to that exact same result which is an AHL-process net. For a direct transformation of AHL-process nets, it remains to show that also the context net C of the direct transformation is an AHL-process net. This follows from the fact that there is an $\mathcal{M}_{AHL}$-morphism $f : C \to K$ using Lemma A.7.1. □

B
Detailed Proofs

B.1 Proof of Fact 3.4.4 (Skeleton Functor)

We have to show that the *Skel* construction for AHL-nets and -morphisms in Definition 3.4.1 form a functor $Skel : \mathbf{AHLNets} \to \mathbf{PTNets}$ that preserves $\mathcal{M}_{AHL}$-morphisms and coproducts as well as pushouts and pullbacks along $\mathcal{M}_{AHL}$-morphisms.

Skel **is functor.** We have to show that for an AHL-morphism $f : AN_1 \to AN_2$ the image $Skel(f)$ is a P/T-morphism $Skel(f) : Skel(AN_1) \to Skel(AN_2)$. Let $t \in T_1$ with $pre_1(t) = \sum_{i=1}^{n}(term_i, p_i)$, then by definition of AHL-morphisms we have

$$pre_2(f_T(t)) = (f_\Sigma^\# \otimes f_P)^\oplus(pre_1(t)) = (f_\Sigma^\# \otimes f_P)^\oplus(\sum_{i=1}^{n}(term_i, p_i)) = \sum_{i=1}^{n}(f_\Sigma^\#(term_i), f_P(p_i))$$

which implies

$$pre_{2,S}(Skel(f)_T(t)) = \sum_{i=1}^{n}(f_P(p_i)) = f_P^\oplus(\sum_{i=1}^{n}(p_i)) = f_P^\oplus(pre_{1,S}(t))$$

Hence, $Skel(f)$ preserves the pre domain of t. The proof for the post domain works analogously.

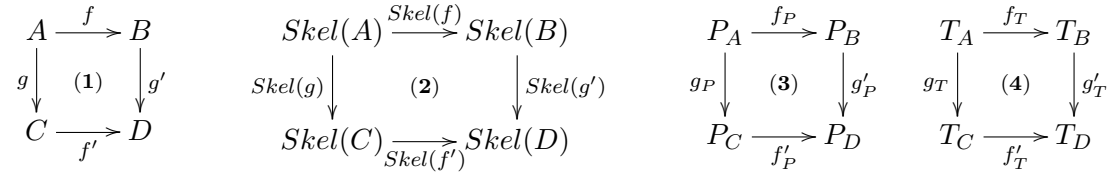
Moreover, *Skel* preserves identities and is compositional since identities as well as composition of morphisms are defined componentwise in **AHLNets** and **PTNets**. Hence, *Skel* is a functor.

Mapping of $\mathcal{M}_{AHL}$-morphisms. Given a functor $f = (f_\Sigma, f_P, f_T, f_A) \in \mathcal{M}_{AHL}$, we have that f_P and f_T are injective functions, and, thus, $Skel(f) = (f_P, f_T) \in \mathcal{M}_{PT}$.

Preservation of pushouts along $\mathcal{M}_{AHL}$. Given pushout (1) in Figure B.1a in **AHLNets** with $f \in \mathcal{M}_{AHL}$, we have to show that (2) in Figure B.1b is pushout in **PTNets**. By componentwise construction of pushouts along $\mathcal{M}_{AHL}$ in **AHLNets** we have also pushouts (3) and (4) in Figure B.1c in **Sets**. As shown above, $f \in \mathcal{M}_{AHL}$ implies $Skel(f) \in \mathcal{M}_{PT}$, and by componentwise construction of pushouts along $\mathcal{M}_{PT}$ in **PTNets** we also have that (2) is a pushout.

Preservation of coproducts. This follows from the previous item since coproducts are pushouts over an initial object which in **AHLNets** and **PTNets** are the empty nets, and we have that the skeleton of an empty AHL-net is an empty P/T net.

Preservation of pullbacks along $\mathcal{M}_{AHL}$. Given pullback (1) in Figure B.1a in **AHLNets** with $f \in \mathcal{M}_{AHL}$, we have to show that (2) in Figure B.1b is pullback in **PTNets**. This



(a) Pushout in **AHLNets**

(b) Pushout in **PTNets**

(c) Pushouts in **Sets**

Figure B.1: Preservation of pushouts along $\mathcal{M}_{AHL}$ by $Skel$

proof works analogously to the proof for pushouts above, due to the componentwise construction of pullbacks along $\mathcal{M}_{AHL}$ and $\mathcal{M}_{PT}$ in **AHLNets** and **PTNets**, respectively. □

B.2 Proof of Fact 3.4.8 (Natural Inclusion)

Given an AHL-net AN, the existence of the inclusion $w(AN) : Flat(AN) \hookrightarrow wFlat(AN)$ follows directly from the definitions of $Flat(AN)$ and $wFlat(AN)$. Moreover, given an AHL-morphism $f : AN_1 \to AN_2 \in \mathcal{M}_{AHL}$. Since the data type part of $\mathcal{M}_{AHL}$-morphisms is an inclusion, we can w. l. o. g. assume that it is an identity and we have that f is in $\mathbf{AHLNets}(\mathbf{\Sigma}, \mathbf{A})$. We have to show that naturality square (1) in Figure B.2 is a pullback.

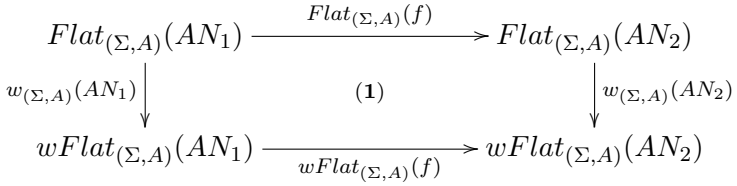
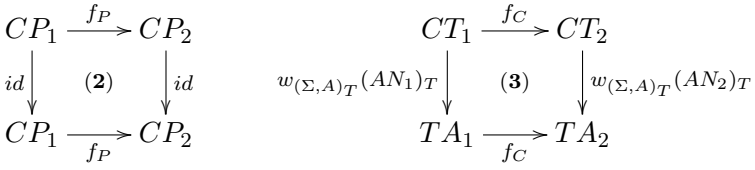


Figure B.2: Naturality square of $w_{(\Sigma,A)}$

Since $w_{(\Sigma,A)}(AN_2)$ is an inclusion and pullbacks in **PTNets** along injective morphisms can be constructed componentwise, we have to show that (2) and (3) in Figure B.3 are pullbacks in **Sets**. Due to Definition 3.4.5 we have the trivial pullback (2) in Figure B.3a of places, and it remains that diagram (3) in Figure B.3b is a pullback in **Sets**.



(a) Pullback of places

(b) Pullback of transitions

Figure B.3: Cartesian transformation $w_{(\Sigma,A)}$

Let X be a set with functions $g_1 : X \to TA_1$ and $g_2 : X \to CT_2$, we define a function $g : X \to CT_1$ with $g(x) = g_1(x)$ with $wFlat_{(\Sigma,A)}(f) \circ g_1 = w_{(\Sigma,A)} \circ g_2$. In order to show that g is a well-defined function it suffices to show that for every x there is $g_1(x) \in CT_1$, since g_1 is a function and with this definition g satisfies the universal property. Let $g_1(x) = (t_1, v_1)$

and $g_2(x) = (t_2, v_2)$. We have $(t_1, v_1) \in CT_1$ if $v_1 : Var(t_1) \to A$ is valid under A. Since we have $AN_1, AN_2 \in \mathbf{AHLNets}(\mathbf{\Sigma}, \mathbf{A})$, there is $f_A = id_A$ and $f_X|_{Var(t_1)} = id_X$ which implies $f_C(t_1, v_1) = (f_T(t_1), f_A \circ v_1 \circ f_X|^{-1}_{Var(t_1)}) = (f_T(t_1), v_1)$. Thus, by commutativity $wFlat_{(\Sigma,A)}(f) \circ g_1 = w_{(\Sigma,A)} \circ g_2$ and the fact that $w_{(\Sigma,A)}(AN_2)$ is an inclusion, we have

$$(f_T(t_1), v_1) = f_C(t_1, v_1) = wFlat_{(\Sigma,A)}(f)_T(t_1, v_1) = wFlat_{(\Sigma,A)}(f)_T(g_1(x)) = g_2(x) = (t_2, v_2)$$

Hence, we know that $v_1 = v_2$ is a valid assignment which means that $g_1(x) = (t_1, v_1) \in CT_1$ and therefore g is well-defined. □

B.3 Proof of Fact 3.4.9 (Natural Projection)

We show the universal property of (1) in Figure B.4a, so let X be a set with functions $g_1 : X \to P_1$ and $g_2 : X \to A_2 \otimes P_2$ such that $f_P \circ g_1 = wproj_{(\Sigma,A)}(AN_2)_P \circ g_2$.

$$\begin{array}{ccc} A_1 \otimes P_1 & \xrightarrow{id_A \otimes f_P} & A_2 \otimes P_2 \\ {\scriptstyle wproj_{(\Sigma,A)}(AN_1)_P}\downarrow & (1) & \downarrow{\scriptstyle wproj_{(\Sigma,A)}(AN_2)_P} \\ P_1 & \xrightarrow[f_P]{} & P_2 \end{array} \qquad \begin{array}{ccc} TA_1 & \xrightarrow{f_C} & TA_2 \\ {\scriptstyle wproj_{(\Sigma,A)}(AN_1)_T}\downarrow & (2) & \downarrow{\scriptstyle wproj_{(\Sigma,A)}(AN_2)_T} \\ T_1 & \xrightarrow[f_T]{} & T_2 \end{array}$$

(a) Pullback of places (b) Pullback of transitions

Figure B.4: Naturality squares of $P(wproj_{(\Sigma,A)})$ and $T(wproj_{(\Sigma,A)})$

We define a function $g : X \to A_1 \otimes P_1$ with

$$g(x) = (a_2, p_1) \text{ for } g_1(x) = p_1 \text{ and } g_2(x) = (a_2, p_2)$$

The function g is well-defined because g_1 and g_2 are functions. Moreover, we have

$$wproj_{(\Sigma,A)}(AN_1)_P \circ g(x) = wproj_{(\Sigma,A)}(AN_1)_P(a_2, p_1) = p_1 = g_1(x)$$

and, since $f_P \circ g_1 = wproj_{(\Sigma,A)}(AN_2)_P \circ g_2$ there is $f_P(p_1) = p_2$ and thus

$$(id_A \otimes f_P) \circ g(x) = id_A \otimes f_P(a_2, p_1) = (a_2, f_P(p_1)) = (a_2, p_2) = g_2(x)$$

Let us assume that there is $g' : X \to A_1 \otimes P_1$ with $wproj_{(\Sigma,A)}(AN_1)_P \circ g' = g_1$ and $(id_A \otimes f_P) \circ g' = g_2$, and let $x \in X$ with $g_1(x) = p_1$, $g_2(x) = (a_2, p_2)$, and $g'(x) = (a, p)$. Then we have

$$p = wproj_{(\Sigma,A)}(AN_1)_P(a, p) = wproj_{(\Sigma,A)}(AN_1)_P \circ g'(x) = g_2(x) = p_1$$

and

$$(a_2, p_2) = g_2(x) = (id_A \otimes f_P) \circ g'(x) = (id_A \otimes f_P)(a, p) = (a, f_P(p))$$

which means that $a = a_2$ and thus $g(x) = g'(x)$. Hence, g is unique, and we have that (1) is a pullback.

For the universal property of (2) in Figure B.4b let Y be a set with functions $h_1 : Y \to T_1$ and $h_2 : Y \to TA_2$ such that $f_T \circ h_1 = wproj_{(\Sigma,A)}(AN_2)_T \circ h_2$. We can define a function $h : X \to TA_1$ with

$$h(x) = (t_1, v_2) \text{ for } h_1(x) = t_1 \text{ and } h_2(x) = (t_2, v_2)$$

The proof that h is the required morphism for the universal property of pullback (2) works completely analogously to the proof for g. Hence, also (2) is a pullback. □

B.4 Proof of Fact 3.4.11 (Flattening Functors)

Flat **is functor.** The well-definedness of the functor *Flat* has been shown in [Ehr05] for a slightly different definition of AHL-nets and morphisms (see Remark 3.1.6). The proof is also valid for our definition of AHL-morphisms, since it makes no use of the restriction that the algebra part of the morphisms in [Ehr05] have to be isomorphisms.

Mapping of $\mathcal{M}_{AHL}$-morphisms. Given $f : AN_1 \to AN_2 \in \mathcal{M}_{AHL}$, we have to show that $wFlat(f), Flat(f) \in \mathcal{M}_{PT}$. The function $f_A \otimes f_P$ is injective since f_A is an isomorphism and f_P is injective. Let $(t, v) \in TA$. Since f_A is an isomorphism, and $f_X|_{Var(t)} : Var(t) \to Var(f_T(t))$ and therefore also $(f_X|_{Var(t)})^{-1}$ is a bijection by definition of AHL-morphisms, there is a bijective correspondence between v and $f_A \circ v \circ (f_X|_{Var(t)})^{-1}$. Hence, also f_C with $f_C(t, v) = (f_T(t), f_A \circ v \circ (f_X|_{Var(t)})^{-1})$ is injective because f_T is injective, and we have $wFlat(f) \in \mathcal{M}_{PT}$, and since injectivity is not violated by restricting the domain of a morphism, we also have that $Flat(f) \in \mathcal{M}_{PT}$.

$$\begin{array}{ccc} AN_0 & \xrightarrow{f} & AN_1 \\ {\scriptstyle g}\downarrow & (1) & \downarrow{\scriptstyle g'} \\ AN_2 & \xrightarrow[f']{} & AN_3 \end{array}$$

(a) Pullback or pushout in **AHLNets**

$$\begin{array}{ccc} Flat(AN_0) & \xrightarrow{Flat(f)} & Flat(AN_1) \\ {\scriptstyle Flat(g)}\downarrow & (2) & \downarrow{\scriptstyle Flat(g')} \\ Flat(AN_2) & \xrightarrow[Flat(f')]{} & Flat(AN_3) \end{array} \qquad \begin{array}{ccc} wFlat(AN_0) & \xrightarrow{wFlat(f)} & wFlat(AN_1) \\ {\scriptstyle wFlat(g)}\downarrow & (3) & \downarrow{\scriptstyle wFlat(g')} \\ wFlat(AN_2) & \xrightarrow[wFlat(f')]{} & wFlat(AN_3) \end{array}$$

(b) Pullbacks or pushouts in **PTNets**

Figure B.5: Preservation of pullbacks and pushouts along $\mathcal{M}_{AHL}$

Preservation of pullbacks along $\mathcal{M}_{AHL}$. Given pullback (1) in Figure B.5a in **AHLNets** with $f' \in \mathcal{M}_{AHL}$, we have to show that diagrams (2) and (3) in Figure B.5b are pullbacks in **PTNets**. Due to componentwise construction of pullbacks along $\mathcal{M}_{AHL}$, we have pullbacks (4),(5) in Figure B.6a in **Sets** and (6) in Figure B.6b in **Algs**, and since $wFlat(f') \in \mathcal{M}_{PT}$, for the functor $wFlat$ it suffices to show that we have pullbacks (7) and (8) in Figure B.7 in **Sets**.

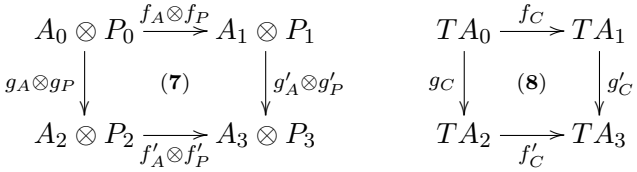
$$\begin{array}{ccc} P_0 & \xrightarrow{f_P} & P_1 \\ {\scriptstyle g_P}\downarrow & (4) & \downarrow{\scriptstyle g'_P} \\ P_2 & \xrightarrow[f'_P]{} & P_3 \end{array} \qquad \begin{array}{ccc} T_0 & \xrightarrow{f_T} & T_1 \\ {\scriptstyle g_T}\downarrow & (5) & \downarrow{\scriptstyle g'_T} \\ T_2 & \xrightarrow[f'_T]{} & T_3 \end{array}$$

(a) Pullbacks in **Sets**

$$\begin{array}{ccc} (\Sigma_0, A_0) & \xrightarrow{(f_\Sigma, f_A)} & (\Sigma_1, A_1) \\ {\scriptstyle (f'_\Sigma, f'_A)}\downarrow & (6) & \downarrow{\scriptstyle (g'_\Sigma, g'_A)} \\ (\Sigma_2, A_2) & \xrightarrow[(f'_\Sigma, f'_A)]{} & (\Sigma_3, A_3) \end{array}$$

(b) Pullback in **Algs**

Figure B.6: Preservation of pullbacks along $\mathcal{M}_{AHL}$ – Pullbacks in **Sets** and **Algs**



(a) Pullback of places

(b) Pullbacks of transitions

Figure B.7: Preservation of pullbacks along $\mathcal{M}_{AHL}$ – Pullbacks in **Sets**

First, we show the universal property for pullback (7). Since $Flat(f)$ is a functor, by commutative diagram (1) we have commutative diagram (2) and therefore we know that (7) commutes, but we cannot directly use that the product $\times$ is compatible with pullbacks, because $A_0 \otimes P_0 \subsetneq A_0 \times P_0$. So let X be a set with functions $h_1 : X \to A_1 \otimes P_1$ and $h_2 : A_2 \otimes P_2$ such that $(g'_A \otimes g'_P) \circ h_1 = (f'_A \otimes f'_P) \circ h_2$. We define a function $h : X \to A_0 \otimes P_0$ with

$$h(x) = (f_A \otimes f_P)^{-1}(h_1(x)).$$

In order to show that h is well-defined, let $x \in X$. We have to show that $h(x) \in A_0 \otimes P_0$. Since $f' \in \mathcal{M}_{AHL}$ and $\mathcal{M}_{AHL}$-morphisms are closed under pullback, there is also $f \in \mathcal{M}_{AHL}$. So we have that f_A is an isomorphism and f_P is injective which means that $(f_A \oplus f_P)$ is injective, and therefore it suffices to show that there is $(a_0, p_0) \in A_0 \otimes P_0$ such that $(f_A \otimes f_P)(a_0, p_0) = h_1(x) =: (a_1, p_1)$. Let $h_2(x) = (a_2, p_2)$, then we have

$$g'_A \otimes g'_P(a_1, p_1) = g'_A \otimes g'_P(h_1(x)) = f'_A \otimes f'_P(h_2(x)) = f'_A \otimes f'_P(a_2, p_2)$$

which means that $g'_P(p_1) = f'_P(p_2)$. By universal property of pullback (4) there is a place $p_0 \in P_0$ with $f_P(p_0) = p_1$ and $g_P(p_0) = p_2$.

Furthermore, $(a_1, p_1) \in A_1 \otimes P_1$ means that $a_1 \in A_{1,type_1(p_1)}$. Let $a_0 = f_A^{-1}(a_1) \in A_{0,f_S^{-1}(type_1(p_1))}$. We have that

$$f_S^{-1}(type_1(p_1)) = f_S^{-1}(type_1(f_P(p_0))) = f_S^{-1}(f_S(type_0(p_0))) = type_0(p_0)$$

and thus $a_0 \in A_{0,type_0(p_0)}$ which means that $(a_0, p_0) \in A_0 \otimes P_0$.

Moreover, we have $(f_A \otimes f_P) \circ h(x) = (f_A \oplus f_P) \circ (f_A \oplus f_P)^{-1}(h_1(x)) = h_1(x)$, and by commutativity of (7) we have

$$\begin{aligned}(f'_A \otimes f'_P) \circ (g_A \otimes g_P) \circ h(x) &= (f'_A \otimes f'_P) \circ (g_A \otimes g_P) \circ (f_A \oplus f_P)^{-1}(h_1(x)) \\ &= g'_A \otimes g'_P(h_1(x)) = f'_A \otimes f'_P(h_2(x))\end{aligned}$$

which by injectivity of $f'_A \otimes f'_P$ implies $g_A \otimes g_P \circ h(x) = h_2(x)$.

Let $h' : X \to A_0 \otimes P_0$ with $(f_A \otimes f_P) \circ h' = h_1$ and $(g_A \otimes g_P) \circ h' = h_2$. Then for every $x \in X$, we have

$$(f_A \otimes f_P) \circ h'(x) = h_1(x) = (f_A \otimes f_P)((f_A \otimes f_P)^{-1}(h_1(x))) = (f_A \otimes f_P) \circ h(x)$$

which by injectivity of $(f_A \otimes f_P)$ implies $h'(x) = h(x)$ and thus $h' = h$ which means that h is unique and (7) is a pullback.

Now we consider diagram (8) which also commutes due to commutativity of diagram (2). We have to show the universal property. So let Y be a set with functions $k_1 : Y \to TA_1$ and $k_2 : Y \to TA_2$ such that $g'_C \circ k_1 = f'_C \circ k_2$. We define a function $k : Y \to TA_0$ with

$$k(x) = (f_T^{-1}(t), f_A^{-1} \circ v \circ f_X|_{Var(f_T^{-1}(t))}) \text{ for } k_1(x) = (t, v)$$

We have to show that k is well-defined. By $f \in \mathcal{M}_{AHL}$ we have that f_T is injective and f_A is an isomorphism which means that the mapping of k leads to unique results. It remains to show that for every $x \in Y$ and $(t_1, v_1) \in TA_1$ with $k_1(x) = (t_1, v_1)$ there is $t_0 \in T_0$ with $f_T(t_0) = t_1$. For every $x \in Y$ and $(t_1, v_1) \in TA_1$ with $k_1(x) = (t_1, v_1)$ there is also $(t_2, v_2) \in TA_2$ with $k_2(x) = (t_2, v_2)$ and we have

$$\begin{aligned}(g'_T(t_1), g'_A \circ v_1 \circ (g'_X|_{Var(t_1)})^{-1}) &= g'_C(t_1, v_1)\\ &= g'_C \circ k_1(x)\\ &= f'_C \circ k_2(x)\\ &= f'_C(t_2, v_2)\\ &= (f'_T(t_2), f'_A \circ v_2 \circ (f'_X|_{Var(t_2)})^{-1})\end{aligned}$$

which implies that $g'_T(t_1) = f'_T(t_2)$. Thus, by pullback (5) in **Sets** there is $t_0 \in T_0$ with $f_T(t_0) = t_1$ and $g_T(t_0) = t_2$. Thus, k is well-defined. The required commutativity and uniqueness of k follows analogously to the case of h. Hence, (8) is pullback in **Sets** and therefore (2) is a pullback in **PTNets**.

It remains to show that (3) is a pullback in **PTNets**. Consider the cube in Figure B.8 where the bottom is (2) pullback from Figure B.5b, the side faces commute by natural transformation $w : Flat \Rightarrow wFlat$ and the top commutes by functor $Flat$. Due to $f, f' \in \mathcal{M}_{AHL}$ by Fact 3.4.8 we have that the left front and the right back face are pullbacks because w is $\mathcal{M}$-cartesian. Thus, by composition of pullbacks, the right back face and the bottom together form a pullback, which by commutativity of the cube implies that the top and the left front face together form a pullback. Hence, by decomposition of pullbacks, the top face which is the diagram (3) in Figure B.5b is a pullback in **PTNets**.

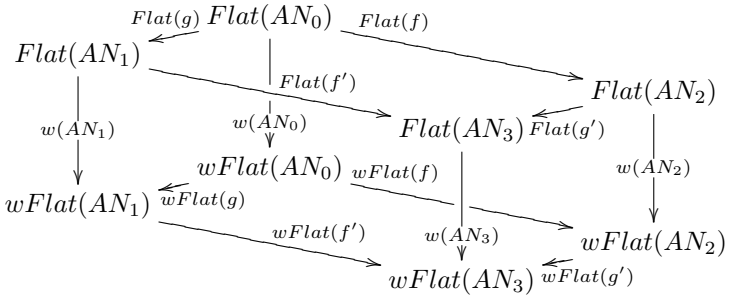


Figure B.8: Commutative cube

Preservation of pushouts along $\mathcal{M}_{AHL}$. Given pushout (1) in Figure B.5a in category **AHLNets**$(\mathbf{\Sigma}, \mathbf{A})$ with $f \in \mathcal{M}_{AHL}$. We have to show that (2) and (3) in Figure B.5b are pushouts in **PTNets**. Consider the cube in Figure B.9 where for a better overview we omitted the indices (Σ, A). Since $Flat$, $wFlat$ and $Skel$ are functors, and w as well as $wproj$ are natural transformations, the cube commutes. Moreover, by Fact 3.4.4, the

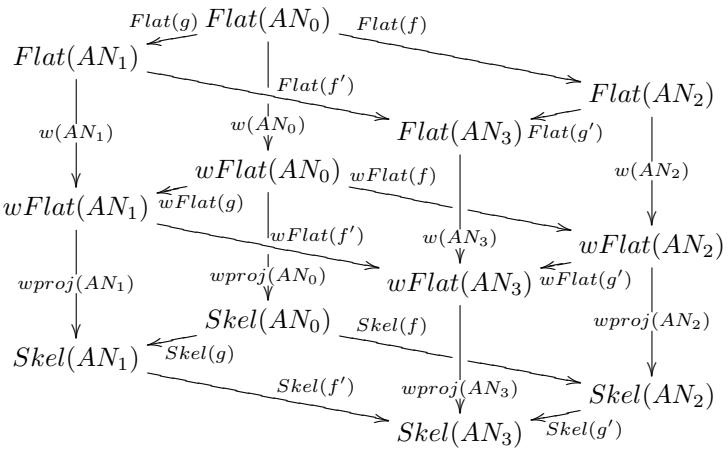
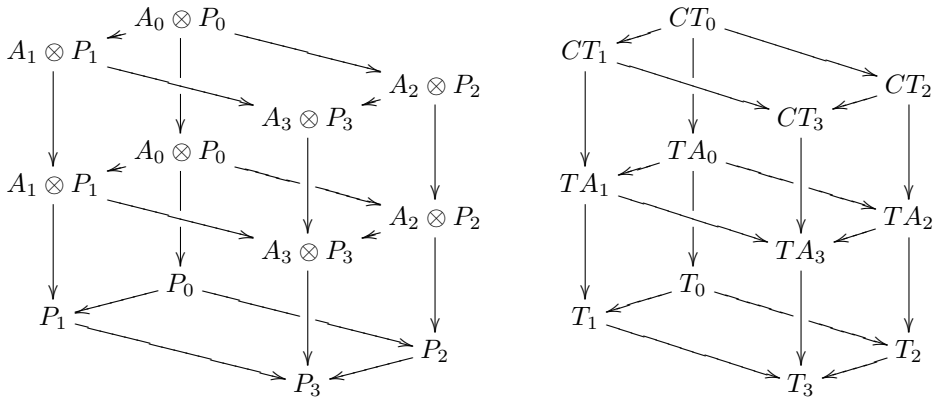


Figure B.9: Commutative cube of flattenings and skeletons

bottom of the cube is a pushout, and by Fact 3.4.8, the upper side faces are pullbacks in **PTNets**.

Further, consider the corresponding cubes of place and transition components in Figure B.10. Due to Fact 3.4.4 the $\mathcal{M}_{AHL}$-morphism f implies that $Skel(f) \in \mathcal{M}_{PT}$, and due to the fact that w is an inclusion, we also have that $w(AN_i) \in \mathcal{M}_{PT}$ for $i = 1, ..., 3$. Thus, by componentwise construction of pushouts and pullbacks along $\mathcal{M}_{PT}$, also the bottoms of the cubes in Figure B.10 are pushouts, and the upper side faces are pullbacks. Moreover, due to Fact 3.4.9 we know that the lower side faces of the cubes are pullbacks.



(a) P-components (b) T-components

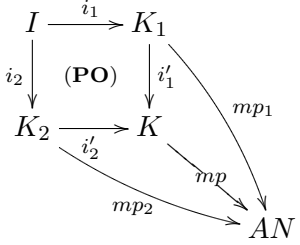
Figure B.10: Commutative cubes of place and transition components

Now, since by $Skel(f) \in \mathcal{M}_{PT}$ the components $Skel(f)_P$ and $Skel(f)_T$ are monomorphisms in **Sets**, and by Theorem 4.6 in [EEPT06b] the category **Sets** is not only $\mathcal{M}$-adhesive but adhesive, the (general) van Kampen cube property (see Remark A.1.10) implies that the middle and top faces of the cubes in Figure B.10 are pushouts in **Sets**.

Finally, as shown above, $f \in \mathcal{M}_{AHL}$ implies that also $Flat(f), wFlat(f) \in \mathcal{M}_{PT}$ which by componentwise construction of pushouts along $\mathcal{M}_{PT}$ implies that the middle and top face of the cube in Figure B.9 are pushouts. □

B.5 Proof of Fact 4.2.6 (Gluing of AHL-Process Nets)

In order to prove Fact 4.2.6, we have to show that the composability of AHL-process nets in Definition 4.2.5 is a sufficient and necessary condition for the gluing of AHL-nets (see Definition 4.2.2), i. e., we have to show that the diagram (PO) below is a gluing of AHL-process nets if and only if K_1 and K_2 are composable w. r. t. (I, i_1, i_2).



If. Given the AHL-process nets K_1, K_2 and I and morphisms i_1, i_2 as above we construct the pushout (PO) in the category **AHLNets**.

In order to show that (PO) is also a pushout in the full subcategory **AHLPNets** it suffices to show that the AHL-net K is an AHL-process net, i. e. K is unary, it has no forward or backward conflicts, and the causal relation $<_K$ is a strict partial order.

Unarity. Let us assume that K is not unary, i. e. there are $p \in P_K$, $t \in T_K$ with

$$(term_1, p) \oplus (term_2, p) \leq pre_K(t) \quad \text{or} \quad (term_1, p) \oplus (term_2, p) \leq post_K(t)$$

Let us consider the case that $(term_1, p) \oplus (term_2, p) \leq pre_K(t)$. Due to the universal property of pushout (PO) there is $a \in \{1, 2\}$ and $t' \in T_{K_a}$ with $i'_{a,T}(t') = t$ and since AHL-morphisms preserve pre and post conditions there is

$$pre_K(t) = (id_{T_{OP}(X)} \otimes i'_{a,P})^{\oplus} \circ pre_{K_a}(t').$$

So the fact that $(term_1, p) \oplus (term_2, p) \leq pre_K(t)$ implies

$$(term_1, p_1) \oplus (term_2, p_2) \leq pre_{K_a}(t')$$

with $i'_a(p_1) = p = i'_a(p_2)$.

- **Case 1.** There is $a = 1$.
  The fact that $i'_1(p_1) = p = i'_1(p_2)$ by pushout (PO) implies that there are $x_1, x_2 \in P_I$ with $i_1(x_1) = p_1$, $i_1(x_1) = p_2$ and $i_2(x_1) = i_2(x_2)$. Moreover, $(term_1, p_1) \oplus (term_2, p_2) \leq pre_{K_1}(t')$ means $i_1(x_1), i_1(x_2) \notin OUT(K_1)$.
  - **Case 1.1.** There are $x_1, x_2 \in OUT(I)$.
    Then by composability of K_1 and K_2 w. r. t. (I, i_1, i_2) we have that $i_1(x_1) \in OUT(K_1)$ or $i_1(x_2) \in OUT(K_1)$, contradicting the fact that $i_1(x_1), i_1(x_2) \notin OUT(K_1)$.
  - **Case 1.2.** There is $x_1 \notin OUT(I)$.
    This means that there is $t_0 \in T_I$ with $(term_0, x_1) \leq pre_I(t_0)$. By the fact that AHL-morphisms preserve pre conditions, we obtain $(term_0, i_1(x_1)) \leq pre_{K_1}(i_1(t_0))$. This implies that $i_1(t_0) = t'$ because K_1 is an AHL-process net which does not have any forward conflicts. Moreover, we have $term_0 = term_1$. So, using again the fact that AHL-morphisms preserve pre conditions, we obtain $(term_2, x_2) \in pre_I(t_0)$ and furthermore $(term_1, i_2(x_1)) \oplus (term_2, i_2(x_2)) \leq pre_{K_2}(i_2(t_0))$. Hence, by the fact that $i_2(x_1) = i_2(x_2)$ we have that K_2 is not unary. This is a contradiction because K_2 is an AHL-process net.

– **Case 1.3.** There is $x_2 \notin OUT(I)$.
Analogously to Case 1.2 this case leads to a contradiction because AHL-morphisms preserve post as well as pre conditions and AHL-process net K_1 does also have no backward conflicts.

- **Case 2.** There is $a = 2$.
Since i_1 is injective, also i'_2 is injective, because injective AHL-morphisms are closed under pushouts. Thus, we have $p_1 = p_2$, which means that K_2 is not unary. This is a contradiction to the fact that K_2 is an AHL-process net.

The case $(term_1, p) \oplus (term_2, p) \leq post_K(t)$ works analogously. Hence, K is unary.

No forward conflicts. Let us assume that K has a forward conflict, i. e. there are $p \in P_K$ and $t_1 \neq t_2 \in T_K$ with $p \in {\bullet}t_1 \cap {\bullet}t_2$.

- **Case 1:** There is $a \in \{1, 2\}$ such that $t_1, t_2 \in i'_{a,T}(T_{K_a})$.
Then we have $t'_1 \neq t'_2 \in T_{K_a}$ with

$$i'_{a,T}(t'_1) = t_1 \quad \text{and} \quad i'_{a,T}(t'_2) = t_2$$

and there are $p_1, p_2 \in P_{K_a}$ with

$$i'_{a,P}(p_1) = p = i'_{a,P}(p_2) \quad \text{and} \quad p_1 \in {\bullet}t'_1, p_2 \in {\bullet}t'_2$$

because AHL-morphisms preserve pre conditions.

- **Case 1.1** $p_1 = p_2$.
This means that K_a has a forward conflict which contradicts the fact that K_a is assumed to be an AHL-process net.

- **Case 1.2** $p_1 \neq p_2$.
Since i'_2 is injective, this implies that $a = 1$. Then, $i'_1(p_1) = p = i'_1(p_2)$ implies $x_1, x_2 \in P_I$ with $i_1(x_1) = p_1$, $i_1(x_2) = p_2$ and $i_2(x_1) = i_2(x_2)$. Moreover, $i_1(x_1) = p_1 \in {\bullet}t'_1$ means that $p_1 \notin OUT(K_1)$, and $i_1(x_2) = p_2 \in {\bullet}t'_2$ means that $p_2 \notin OUT(K_1)$.

  Case 1.2.1. There is $x_1, x_2 \in OUT(I)$.
  Then by composability of K_1 and K_2 w. r. t. (I, i_1, i_2) it follows that $i_1(x_1) \in OUT(I)$ or $i_1(x_2) \in OUT(K_1)$, contradicting the fact that $i_1(x_1), i_1(x_2) \notin OUT(K_1)$.

  Case 1.2.2. There is $x_1 \notin OUT(I)$, $x_2 \in OUT(I)$.
  By the fact that $x_2 \in OUT(I)$ and $i_1(x_2) \notin OUT(K_1)$ the composability of K_1 and K_2 w. r. t. (I, i_1, i_2) implies $i_2(x_2) \in OUT(K_2)$.
  Furthermore, the fact that $x_1 \notin OUT(I)$ means that there is some $t_0 \in T_I$ with $(term_0, x_1) \leq pre_I(t_0)$. By the fact that AHL-morphisms preserve pre conditions, we obtain $(term_0, i_1(x_1)) \leq pre_{K_1}(i_1(t_0))$. This implies that $i_1(t_0) = t'_1$ because K_1 is an AHL-process net which does not have any forward conflicts. Moreover, we have $term_0 = term_1$. Using again the fact that AHL-morphisms preserve pre conditions, we obtain $(term_1, i_2(x_1)) \in pre_{K_2}(i_2(t_0))$ which means that $i_2(x_1) = i_2(x_2) \notin OUT(K_2)$, contradicting the fact that $i_2(x_2) \in OUT(K_2)$.

  Case 1.2.3. There is $x_1 \in OUT(I)$, $x_2 \notin OUT(I)$.
  This case is similar to Case 1.2.2.

Case 1.2.4. There is $x_1, x_2 \notin OUT(I)$. Then we have transitions $t_0, t_0' \in T_I$ with $(term_0, x_1) \leq pre_I(t_0)$ and $(term_0', x_2) \leq pre_I(t_0')$. Analogously to Case 1.2.2 we obtain that $i_1(t_0) = t_1'$ and $i_1(t_0') = t_2'$, and using the fact that AHL-morphisms preserve pre conditions, we have $i_2(x_1) \leq pre_{K_2}(i_2(t_0))$ and $i_2(x_1) = i_2(x_2) \leq pre_{K_2}(i_2(t_0'))$. Since K_2 is an AHL-process net, it does not have any forward conflicts, implying that $i_2(t_0) = i_2(t_0')$. Thus, we have

$$t_1 = i_1'(t_1') = i_1'(i_1(t_0)) = i_2'(i_2(t_0)) = i_2'(i_2(t_0')) = i_1'(i_1(t_0')) = i_1'(t_2') = t_2$$

which contradicts the fact that $t_1 \neq t_2$.

- **Case 2:** There is $t_1 \in i_1(T_{K_1})$ and $t_2 \in i_2(T_{K_2})$.
  Then we have $t_1' \in T_{K_1}, t_2' \in T_{K_2}$ with

  $$i_1'(t_1') = t_1 \quad \text{and} \quad i_2'(t_2') = t_2$$

  and since AHL-morphisms preserve pre conditions there are $p_1 \in P_{K_1}$, $p_2 \in P_{K_2}$ with

  $$i_1'(p_1) = p,\ p_1 \in {\bullet t_1'} \quad \text{and} \quad i_2'(p_2) = p,\ p_2 \in {\bullet t_2'}.$$

  By the fact that K is a pushout object of (PO) this implies a place $p_0 \in P_I$ with

  $$i_1(p_0) = p_1 \quad \text{and} \quad i_2(p_0) = p_2.$$

- **Case 2.1:** There is $p_0 \in OUT(I)$.
  Due to the fact that $p_1 \in {\bullet t_1'}$, we have $i_1(p_0) \notin OUT(K_1)$, which by the composability of (K_1, K_2) w. r. t. (I, i_1, i_2) implies that $i_2(p_0) \in OUT(K_2)$ contradicting the fact that $i_2(p_0) = p_2 \in {\bullet t_2'}$.
- **Case 2.2:** There is $p_0 \notin OUT(I)$.
  This means that there is $t_0 \in T_I$ with $p_0 \in {\bullet t_0}$. By the fact that i_1 is an AHL-morphism which preserves pre conditions we have $p_1 \in {\bullet i_1(t_0)}$ which together with the fact that $p_1 \in {\bullet t_1'}$ means that $i_1(t_0) = t_1'$ because K_1 has no forward conflicts. Analogously, due to the fact that also K_2 has no forward conflict we obtain that $i_2(t_0) = t_2'$. Thus, by commutativity of (PO) we have

  $$t_1 = i_1'(t_1') = i_1'(i_1(t_0)) = i_2'(i_2(t_0)) = i_2'(t_2') = t_2$$

  which contradicts the assumption that $t_1 \neq t_2$.

Hence, all cases lead to a contradiction which means that K has no forward conflict.

No backward conflicts. The proof that K does not have any backward conflicts works analogously to the proof concerning the forward conflicts, because AHL-morphisms preserve post as well as pre conditions, and the definition of the composability consists of corresponding conditions for input as well as for output places.

Strict partial order. Since $<_K$ is defined as a transitive closure, it suffices to show that $<_K$ is irreflexive. Due to the fact that AHL-morphisms preserve pre and post conditions we obtain the causal relation of $<_K$ as the transitive closure of

$$\bigcup_{a \in \{1,2\}} \{(i_a'(x), i_a'(y)) \mid x, y \in P_{K_a} \uplus T_{K_a}, x <_{K_a} y\}$$

This means that elements which are causally related in K_1 or K_2 are also causally related in K. Additionally it is possible that elements in the net K are related due to the gluing of one or more elements.
Moreover, if for two interface elements $x_0, y_0 \in P_I \uplus T_I$ the images of these elements are causally related in K, i. e. we have the following **Statement (A)**:

$$\forall x_0, y_0 \in P_I \uplus T_I : i'_1(i_1(x_0)) <_K i'_1(i_1(y_0)) \Rightarrow x_0 <_{(i_1,i_2)} y_0$$

We prove this statement because we need it in the following:
Let $x_0, y_0 \in P_I \uplus T_I$ with $i'_1(i_1(x_0)) <_K i'_1(i_1(y_0))$. Then there is $a \in \{1,2\}$ such that either there is $i_a(x_0) <_{K_a} i_a(y_0)$ or there is $z_0 \in P_I \uplus T_I$ with $i_a(x_0) <_{K_a} i_a(z_0)$ and $i'_1(i_1(x_0)) <_K i'_1(i_1(z_0)) <_K i'_1(i_1(y_0))$. This recursively leads to the fact that $x_0 <_{(i_1,i_2)} y_0$ because the induced causal relation is transitive.

Let us now assume that $<_K$ is not irreflexive, i. e. there exists $x \in P_K \uplus T_K$ s.t. $x <_K x$.

- **Case 1.** There is no element $z \in P_I \uplus T_I$ with $x <_K i'_1(i_1(z)) <_K x$.
  Then there is $a \in \{1,2\}$ and $y \in P_{K_a} \uplus T_{K_a}$ s.t. $i'_a(y) = x$. Since there are no images of interface elements in the causal relation between x and x, the causal relation is completely obtained from causal relations in K_a, i. e. we have $y <_{K_a} y$. This contradicts the fact that $<_{K_a}$ is irreflexive because K_a is an AHL-process net.
- **Case 2.** There is an element $z \in P_I \uplus T_I$ with $x <_K i'_1(i_1(z)) <_K x$.
  Due to the transitivity of $<_K$ there is $i'_1(i_1(z)) <_K i'_1(i_1(z))$ because

  $$i'_1(i_1(z)) <_K x <_K i'_1(i_1(z)).$$

  By statement (A) this implies $z <_{(i_1,i_2)} z$, contradicting the fact that by the composability of K_1 and K_2 w. r. t. (I, i_1, i_2) the induced causal relation $<_{(i_1,i_2)}$ is irreflexive.

Hence, $<_K$ is irreflexive.

Only If. Given the pushout diagram (PO) in the category **AHLNets** with K_3 being an AHL-process net. We have to show that (K_1, K_2) are composable w. r. t. (I, i_1, i_2).

No cycles. Let $x, y \in P_I \uplus T_I$ with $x \prec_{(i_1,i_2)} y$.
Then by the definition of $\prec_{(i_1,i_2)}$ there is

$$i_1(x) <_{K_1} i_1(y) \quad \text{or} \quad i_2(x) <_{K_2} i_2(y)$$

and by the fact that $i'_1 \circ i_1 = i'_2 \circ i_2$, we have

$$i'_1 \circ i_1(x) <_K i'_1 \circ i_1(y)$$

because AHL-morphisms preserve pre and post conditions.
Since $<_K$ is transitive, we have also for the transitive closure $<_{(i_1,i_2)}$ of $\prec_{(i_1,i_2)}$, that $x <_{(i_1,i_2)} y$ implies $i'_1 \circ i_1(x) <_K i'_1 \circ i_1(y)$.
Now, let us assume that $<_K$ is not irreflexive, i. e. there is $x \in P_I \uplus T_I$ with $x <_{(i_1,i_2)} x$. Then there is

$$i'_1 \circ i_1(x) <_K i'_1 \circ i_1(x)$$

contradicting the fact that $<_K$ is irreflexive. Hence, $<_{(i_1,i_2)}$ is irreflexive.

Non-injective gluing. We have to show that for all $p_1 \neq p_2 \in IN(I)$ with $i_2(p_1) = i_2(p_2)$ there is $i_1(p_1) \in IN(K_1)$ or $i_1(p_2) \in IN(K_1)$.

So let $p_1 \neq p_2 \in IN(I)$ with $i_2(p_1) = i_2(p_2)$ and let us assume that $i_1(p_1) \notin IN(K_1)$ and $i_1(p_2) \notin IN(K_2)$. This means that there are $t_1, t_2 \in T_{K_1}$ and terms $term_1, term_2 \in T_\Sigma(X)$ with $(term_1, i_1(p_1)) \leq post_{K_1}(t_1)$ and $(term_2, i_1(p_2)) \leq post_{K_1}(t_2)$.

Since AHL-morphisms preserve post conditions, we have

$$(term_1, i'_1(i_1(p_1))) \leq post_K(i'_1(t_1)) \text{ and } (term_2, i'_1(i_1(p_2))) \leq post_K(i'_1(t_2))$$

Moreover, by commutativity of pushout (PO) we have

$$i'_1(i_1(p_1)) = i'_2(i_2(p_1)) = i'_2(i_2(p_2)) = i'_1(i_1(p_2))$$

We distinguish the following two cases.

Case 1. There is $i'_1(t_1) = i'_1(t_2)$.
Then we have $(term_1, i'_1(i_1(p_1))) \oplus (term_2, i'_1(i_1(p_1))) \leq post_K(i'_1(t_1))$, contradicting unarity of AHL-process net K.

Case 2. There is $i'_1(t_1) \neq i'_1(t_2)$.
Then $(term_1, i'_1(i_1(p_1))) \leq post_K(i'_1(t_1))$ and $(term_1, i'_1(i_1(p_1))) \leq post_K(i'_1(t_2))$ means that AHL-process net K has a backward conflict, which is also a contradiction.

Thus, we have $i_1(p_1) \in IN(K_1)$ or $i_1(p_2) \in IN(K_1)$.

The fact that for all $p_1 \neq p_2 \in OUT(I)$ with $i_2(p_1) = i_2(p_2)$ there is $i_1(p_1) \in OUT(K_1)$ or $i_1(p_2) \in OUT(K_1)$ follows analogously because AHL-morphisms preserve pre as well as post conditions and AHL-process net K also does not have any forward conflicts.

No conflicts. We have to show that $\forall x \in IN(I) : i_1(x) \notin IN(K_1) \Rightarrow i_2(x) \in IN(K_2)$. Let $x \in IN(I)$ with $i_1(x) \notin IN(K_1)$ and let us assume that there is $i_2(x) \notin IN(K_2)$.

Then $i_1(x)$ and $i_2(x)$ both are in the post domain of transitions, i. e. there are $t_1 \in T_{K_1}$ and $t_2 \in T_{K_2}$ such that $i_1(x) \in t_1\bullet$ and $i_2(x) \in t_2\bullet$. Since AHL-morphisms preserve post conditions there is

$$i'_1(i_1(x)) \in i'_1(t_1)\bullet \quad \text{and} \quad i'_2(i_2(x)) \in i'_1(t_2)\bullet$$

and due to the fact that (PO) commutes there is $i'_1(i_1(x)) = i'_2(i_2(x))$ which implies

$$i'_1(i_1(x)) \in i'_1(t_1)\bullet \cap i'_2(t_2)\bullet .$$

Since K is an AHL-process net it has no backward conflict implying that $i'_1(t_1) = i'_2(t_2)$. So due to the pushout property there is $t_0 \in T_I$ with

$$i_1(t_0) = t_1 \quad \text{and} \quad i_2(t_0) = t_2$$

Then by the fact that $i_1(x) \in i_1(t_0)\bullet$ together with the fact that i_1 is an AHL-morphism which preserves post domains it follows that $x \in t_0\bullet$. This contradicts the fact that $x \in IN(I)$. Hence, there is $i_2(x) \in IN(K_2)$.

Now, we show that $\forall x \in OUT(I) : i_1(x) \notin OUT(K_1) \Rightarrow i_2(x) \in OUT(K_2)$. Let $x \in OUT(I)$ with $i_1(x) \notin OUT(K_1)$ and let us assume that there is $i_2(x) \notin OUT(K_2)$.

Then $i_1(x)$ and $i_2(x)$ both are in the pre domain of transitions, i. e. there are $t_1 \in T_{K_1}$ and $t_2 \in T_{K_2}$ such that $i_1(x) \in {}^\bullet t_1$ and $i_2(x) \in {}^\bullet t_2$. Since AHL-morphisms preserve pre conditions there is

$$i'_1(i_1(x)) \in {}^\bullet i'_1(t_1) \quad \text{and} \quad i'_2(i_2(x)) \in {}^\bullet i'_1(t_2)$$

and by commutativity of (PO) we have $i'_1(i_1(x)) = i'_2(i_2(x))$ which implies

$$i'_1(i_1(x)) \in {}^\bullet i'_1(t_1) \cap {}^\bullet i'_2(t_2)$$

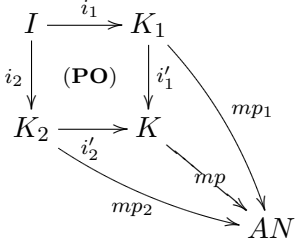
Since K is an AHL-process net it has no forward conflict implying that $i'_1(t_1) = i'_2(t_2)$. So due to the pushout property there is $t_0 \in T_I$ with

$$i_1(t_0) = t_1 \text{ and } i_2(t_0) = t_2$$

Then by the fact that $i_1(x) \in {}^\bullet i_1(t_0)$ together with the fact that i_1 is an AHL-morphism which preserves pre domains it follows that $x \in {}^\bullet t_0$. This contradicts the fact that $x \in OUT(I)$. Hence, there is $i_2(x) \in OUT(K_2)$.

Extension to Processes.

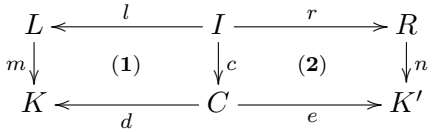
Given the pushout (PO) and additional AHL-morphisms $mp_1 : K_1 \to AN$ and $mp_2 : K_2 \to AN$ with $mp_1 \circ i_1 = mp_2 \circ i_2$.



Then we also have a morphism $mp_0 : I \to AN$ defined by $mp_0 := mp_1 \circ i_1 = mp_2 \circ i_2$. Moreover the pushout property of (PO) implies a unique morphism $mp : K \to AN$ such that (PO) is also a pushout in the slice category $\mathbf{AHLNets} \setminus AN$. As shown above the composability of K_1 and K_2 w. r. t. (I, i_1, i_2) implies that K is an AHL-process net. Hence, $mp : K \to AN$ is an AHL-process which implies that (PO) is also pushout in the full subcategory $\mathbf{Proc(AN)} \subseteq \mathbf{AHLNets} \setminus AN$ of AHL-processes. □

B.6 Proof of Theorem 4.2.11 (Direct Transformation of AHL-Process Nets)

Given a production for AHL-process nets $\varrho : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$ and an AHL-process net K together with a morphism $m : L \to K$. In order to prove Theorem 4.2.11, we have to show that the direct transformation of AHL-process nets with pushouts (1) and (2) in **AHLPNets** exists if and only if ϱ satisfies the transformation condition for AHL-process nets under m.



First, we prove the following lemma which states the equivalence of the gluing relation for a given production and match and the induced causal relation of the right-hand side of the production and the context net in **AHLNets**.

Lemma B.6.1 (Gluing Relation Lemma)**.** *Given a production for AHL-process nets $\varrho : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$, a match $m : L \to K$ where K is an AHL-process net, and pushout (1) in* **AHLNets**.

Then the gluing relation $<_{(\varrho,m)}$ is exactly the induced causal relation of C and R w. r. t. (I, c, r), i. e. $<_{(\varrho,m)} = <_{(c,r)}$.

$$\begin{array}{ccccc} L & \xleftarrow{l} & I & \xrightarrow{r} & R \\ {\scriptstyle m}\downarrow & (\mathbf{1}) & \downarrow{\scriptstyle c} & & \\ K & \xleftarrow[d]{} & C & & \end{array}$$

Proof. We define a relation $\prec_C \subseteq (P_C \times T_C) \uplus (T_C \times P_C)$ as follows:

$$\prec_C = \{(p,t) \in P_C \times T_C \mid p \in {}^\bullet t\} \cup \{(t,p) \in T_C \times P_C \mid p \in t^\bullet\}$$

The relation $\prec_C$ describes the direct causal relationship of the elements in C, i. e. the causal relation $<_C$ is the transitive closure of $\prec_C$. We show that for the relation $\prec_{(K,m)}$ in Definition 4.2.9 we have $\prec_{(K,m)} = \prec_C$, by showing that there is a subset relation in both directions.

Direction 1 ($\prec_{(K,m)} \subseteq \prec_C$). Let $x, y \in P_K \uplus (T_K \setminus m_T(T_L))$ with $x \prec_{(K,m)} y$. Due to the bipartite structure of Petri nets there are two possible cases:

- **Case 1.** There is $x \in P_K$ and $y \in T_K \setminus m_T(T_L)$.
  Due to the construction of C there is $y \in T_C$. Furthermore there is $term \in T_{OP}(X)_{type_K(x)}$ such that

$$\begin{aligned} (term, x) \leq pre_K(y) &\Leftrightarrow (term, x) \leq pre_K|_{T_C}(y) \\ &\Leftrightarrow (term, x) \leq pre_C(y) \end{aligned}$$

  and hence $x \prec_C y$.
- **Case 2.** There is $x \in T_K \setminus m_T(T_L)$ and $y \in P_K$.
  In this case we have $x \in T_C$ and there is $term \in T_{OP}(X)_{type_K(x)}$ such that

$$\begin{aligned} (term, y) \leq post_K(x) &\Leftrightarrow (term, y) \leq post_K|_{T_C}(x) \\ &\Leftrightarrow (term, y) \leq post_C(x) \end{aligned}$$

  and hence $x \prec_C y$.

Direction 2 ($\prec_C \subseteq \prec_{(K,m)}$). Let $x, y \in P_C \uplus T_C$ with $x \prec_C y$. Again we distinguish the two possible cases:

Case 1. There is $x \in P_C$ and $y \in T_C$.
Then there is $term \in T_{OP}(X)_{type_C(x)}$ such that $(term, x) \leq pre_C(y)$. Since AHL-morphisms preserve pre conditions and d is an inclusion we have

$$\begin{aligned} (term, x) \leq pre_C(y) &\Leftrightarrow (term, x) \leq d^{\oplus} \circ pre_C(y) \\ &\Leftrightarrow (term, x) \leq pre_K(d(y)) \\ &\Leftrightarrow (term, x) \leq pre_K(y) \end{aligned}$$

So the fact that $T_C = T_K \setminus m_T(T_L)$ implies $x \prec_{(K,m)} y$.

Case 2. There is $x \in T_C$ and $y \in P_C$.
Then there is $term \in T_{OP}(X)_{type_C(x)}$ such that $(term, x) \leq post_C(y)$. Since AHL-morphisms preserve not only pre but also post conditions we obtain analogously to Case 1 that $x \prec_{(K,m)} y$.

So we have that $\prec_{(K,m)}=\prec_C$ and since $<_{(K,m)}$ is the transitive closure of $\prec_{(K,m)}$ and $<_C$ is the transitive closure of $\prec_C$ it follows that $<_{(K,m)}=<_C$.
Furthermore we can use the inclusion d to obtain from the commutativity of (1) that

$$m \circ l(x) = d \circ c(x) = c(x).$$

So let $\prec_{(c,r)} \subseteq (P_I \times T_I) \uplus (T_I \times P_I)$ be the relation defined by

$$\prec_{(c,r)} = \{(x,y) \mid c(x) <_C c(y) \vee r(x) <_R r(y)\}$$

then we have

$$\begin{aligned}
\prec_{(\varrho,m)} &= \{(x,y) \in (P_I \times T_I) \uplus (T_I \times P_I) \mid m \circ l(x) <_{(K,m)} m \circ l(y) \vee r(x) <_R r(y)\} \\
&= \{(x,y) \in (P_I \times T_I) \uplus (T_I \times P_I) \mid c(x) <_C c(y) \vee r(x) <_R r(y)\} \\
&= \{(x,y) \in (P_I \times T_I) \uplus (T_I \times P_I) \mid r(x) <_R r(y) \vee c(x) <_C c(y)\} \\
&= \prec_{(r,c)}
\end{aligned}$$

and since $<_{(\varrho,m)}$ is the transitive closure of $\prec_{(\varrho,m)}$ and $<_{(r,c)}$ is the transitive closure of $\prec_{(r,c)}$ we have $<_{(\varrho,m)} = <_{(r,c)}$.

□

Now we show that the gluings of AHL-nets (1) and (2) below exist if and only if the production p under m satisfies the transformation condition for AHL-process nets.

$$\begin{array}{ccccc}
L & \xleftarrow{l} & I & \xrightarrow{r} & R \\
{\scriptstyle m}\downarrow & (1) & {\scriptstyle c}\downarrow & (2) & \downarrow{\scriptstyle n} \\
K & \xleftarrow{d} & C & \xrightarrow{e} & K'
\end{array}$$

If. Given production $\varrho : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$ satisfying the transformation condition for AHL-processes under match m. Since this implies that ϱ satisfies the the gluing condition for AHL-nets by Fact 3.2.14 there exist pushouts (1) and (2) in **AHLNets**. We have to show that (1) and (2) are also pushouts in the category **AHLPNets** of AHL-process nets.

Pushout (1). Since $\mathcal{M}_{AHL}$-morphisms are closed under pushouts, the fact that we have $l \in \mathcal{M}_{AHL}$ implies that also $d \in \mathcal{M}_{AHL}$. From AHL-process net K and injective AHL-morphism $d : C \to K$ it follows by Lemma A.7.1 that also C is an AHL-process net. So we have that all objects and morphisms in pushout (1) are in the full subcategory **AHLPNets** $\subseteq$ **AHLNets** which means that (1) is also a pushout in **AHLPNets**.

Pushout (2). We have to show that (R, C) are composable w. r. t. (I, r, c) (see Definition 4.2.5).

No cycles. The fact that the gluing relation $<_{(\varrho,m)}$ of ϱ und m is a strict partial order implies that the induced causal relation $<_{(r,c)}$ is a strict partial order because by Lemma B.6.1 there is $x <_{(\varrho,m)} y \;\Leftrightarrow\; x <_{(r,c)} y$.

Non-injective gluing. From pushout (1) in **AHLPNets** of morphisms l and c it follows by Fact 4.2.6 that (L, C) are composable w. r. t. (I, l, c).
Let $p_1 \neq p_2 \in IN(I)$ with $c(p_1) = c(p_2)$. Then we have

$$m \circ l(p_1) = d \circ c(p_1) = d \circ c(p_2) = m \circ l(p_2)$$

which by the fact that ϱ and m satisfy the transformation condition implies that $r(p_1) \in IN(R)$ or $r(p_2) \in IN(R)$.

Analogously, we obtain for $p_1 \neq p_2 \in OUT(I)$ with $c(p_1) = c(p_2)$ that there is $r(p_1) \in OUT(R)$ or $r(p_2) \in OUT(R)$.

No conflicts. Let $x \in IN(I)$ and $c(x) \notin IN(C)$ then by the composability of (L, C) w. r. t. (I, l, c) follows that $l(x) \in IN(L)$. The fact that $c(x) \notin IN(C)$ implies $t \in T_C$ with $c(x) \in t\bullet$ and $d \circ c(x) \in d(t)\bullet$ because AHL-morphisms preserve post conditions. Due to the commutativity of (1) there is $m \circ l(x) = d \circ c(x)$ which means that $m \circ l(x) \notin IN(K)$ because $m \circ l(x) \in d(t)\bullet$.

So there is $x \in InP$ and the fact that production ϱ and match m satisfy the transformation condition for AHL-processes implies that $r(x) \in IN(R)$.

Now, let $x \in OUT(I)$ and $c(x) \notin OUT(C)$ then by the composability of (L, C) w. r. t. (I, l, c) follows that $l(x) \in OUT(L)$. The fact that $c(x) \notin OUT(C)$ implies $t \in T_C$ with $c(x) \in \bullet t$ and $d \circ c(x) \in \bullet d(t)$ because AHL-morphisms preserve pre conditions. Due to the commutativity of (1) there is $m \circ l(x) = d \circ c(x)$ which means that $m \circ l(x) \notin OUT(K)$ because $m \circ l(x) \in \bullet d(t)$.

So there is $x \in OutP$ and the fact that production ϱ and match m satisfy the transformation condition for AHL-processes implies that $r(x) \in OUT(R)$.

Thus, (R, C) are composable w. r. t. (I, r, c) leading to the existence of pushout (2) in **AHLPNets**.

Only If. Given pushouts (1) and (2) in **AHLNets** and **AHLPNets**. We have to show that the transformation condition for AHL-process nets (see Definition 4.2.10) is satisfied by production ϱ under match m.

Gluing condition. Due to pushouts (1) and (2) in **AHLNets** by Fact 3.2.14 we have that the gluing condition is satisfied.

No cycles. By Fact 4.2.6 pushout (2) in **AHLPNets** implies that (R, C) are composable w. r. t. (I, r, c) which means that $<_{(r,c)}$ is a strict partial order. Due to Lemma B.6.1 we know that there is $<_{(r,c)} = <_{(\varrho,m)}$ which means that also $<_{(\varrho,m)}$ is a strict partial order.

Non-injective gluing. Let $p_1 \neq p_2 \in IN(I)$ with $m \circ l(p_1) = m \circ l(p_2)$. Since l is injective, $p_1 \neq p_2$ implies $l(p_1) \neq l(p_2)$. Then due to the fact that (1) is a pushout, there is $p \in P_C$ with $c(p_1) = p = c(p_2)$. Thus, by composability of (R, C) w. r. t. (I, r, c) it follows that $r(p_1) \in IN(R)$ or $r(p_2) \in IN(R)$.

Analogously, we obtain for $p_1 \neq p_2 \in OUT(I)$ with $m \circ l(p_1) = m \circ l(p_2)$ that $r(p_1) \in OUT(R)$ or $r(p_2) \in OUT(R)$.

No conflicts. Let $x \in InP$ which means that $x \in IN(I)$ with $l(x) \in IN(L)$ and $m \circ l(x) \notin IN(K)$. The fact that $m \circ l(x) \notin IN(K)$ implies that there is $t \in T_K$ with $m \circ l(x) \in t\bullet$.

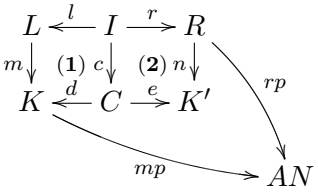
Let us assume that there is $t' \in T_L$ with $m_T(t') = t$. Then from the fact that m is an AHL-morphism, it follows that $l(x) \in t'\bullet$ because AHL-morphisms preserve post conditions. This contradicts the fact that $l(x) \in IN(K)$ and thus $t \notin m_T(T_L)$ which means that $t \in T_K \setminus m_T(T_L)$. Then by the construction of pushout complement T_C in **AHLNets** it follows that $t \in T_C$.

Moreover, we have $c(x) = m \circ l(x) \in t\bullet$ which means that $c(x) \notin IN(C)$. This implies that $r(x) \in IN(R)$ due to the composability of (R, C) w. r. t. (I, r, c) given by pushout (2) in **AHLPNets** and Fact 4.2.6.

Now, let $x \in OutP$ which means that $x \in OUT(I)$ with $l(x) \in OUT(L)$ and $m \circ l(x) \notin OUT(K)$. Then $m \circ l(x) \notin OUT(K)$ implies that there is $t \in T_K$ with $m \circ l(x) \in \bullet t$. Again, the assumption that $t' \in T_L$ with $m_T(t') = t$ leads to a contradiction which means that $t \in T_K \setminus m_T(T_L)$. Then by the construction of T_C follows that $t \in T_C$ and we have $c(x) = m \circ l(x) \in \bullet t$ which means that $c(x) \notin OUT(C)$ and hence $r(x) \in OUT(R)$ by composability of (R, C) w. r. t. (I, r, c).

Extension to Processes.

Given pushouts (1) and (2) in **AHLNets** and **AHLPNets** and additional morphisms $mp : K \to AN$ and $rp : R \to AN$ with $mp \circ m \circ l = rp \circ r$.



Since L, C and I are AHL-process nets we obtain AHL-processes by composition of AHL-morphisms $lp := mp \circ m : L \to AN$, $cp := mp \circ d : C \to AN$ and $ip := mp \circ m \circ l = mp \circ d \circ c : I \to AN$ such that (1) is a commuting diagram in **Proc**(**AN**).

By construction of pushouts in slice categories the pushout (1) in **AHLNets** is also a pushout in **AHLNets** $\setminus AN$. Hence, due to the fact that lp, cp, ip and mp are AHL-processes we have that (1) is a pushout in the full subcategory **Proc**(**AN**) $\subseteq$ **AHLNets** $\setminus AN$.

Finally, we have

$$cp \circ c = mp \circ d \circ c = mp \circ m \circ l = rp \circ r$$

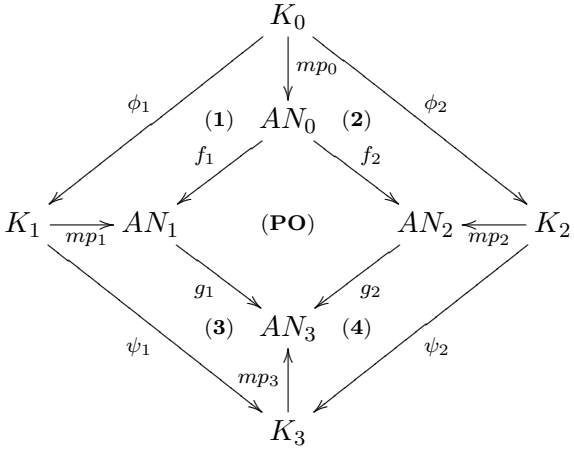
which by Fact 4.2.6 implies a unique morphism $mp' : K' \to AN$ such that (2) is also a pushout in **Proc**(**AN**). □

B.7 Proof of Theorem 4.5.13 (Amalgamation Theorem for AHL-Processes)

1. **Composition Construction.** The fact that (mp_0, ϕ_1) and (mp_0, ϕ_2) are agreement restrictions for mp_1 and mp_2 implies that (K_1, K_2) are composable w. r. t. (K_0, ϕ_1, ϕ_2) which by Fact 4.2.6 implies that the composition $K_3 = K_1 +_{(K_0,\phi_1,\phi_2)} K_2$ exists and is an AHL-process net. According to Definition 4.2.2 the gluing of AHL-process nets is also a pushout in **AHLNets**, and we have

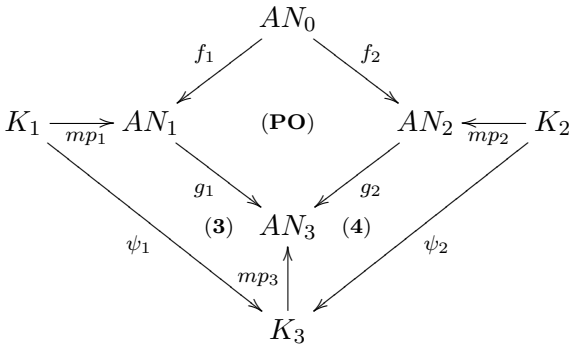
   $$g_1 \circ mp_1 \circ \phi_1 = g_2 \circ mp_2 \circ \phi_2$$

   Thus, the pushout property of pushout (PO) in **AHLNets** implies a unique AHL-morphism $mp_3 : K_3 \to AN_3$ such that (3) and (4) below commute.



From the fact that $f_1, f_2 \in \mathcal{M}_{AHL}$ it follow that also ϕ_1 and ϕ_2 are $\mathcal{M}_{AHL}$-morphisms, because $\mathcal{M}_{AHL}$-morphisms are closed under pullbacks. Moreover, due to closure under pushouts it follows that we also have $g_1, g_2, \psi_1, \psi_2 \in \mathcal{M}_{AHL}$. This means that the above diagram is a weak Van Kampen cube with pushouts as top and bottom faces and pullbacks as back faces, where all horizontal morphisms are $\mathcal{M}_{AHL}$-morphisms. Since $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ is a weak adhesive HLR category (see [EEPT06b]), the Van Kampen-property (see Definition A.1.9) implies that also the front faces (3) and (4) are pullbacks and hence mp_3 is the amalgamation $mp_3 = mp_1 +_{m_0} mp_2$.

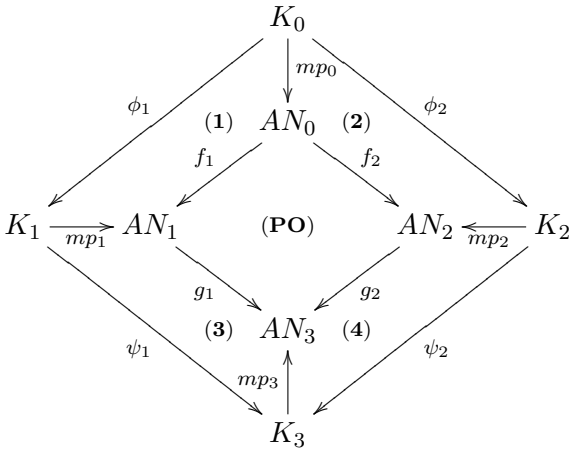
2. **Decomposition Construction.** Given restrictions (mp_1, ψ_1) and (mp_2, ψ_2), we have pullbacks (3) and (4) below in **AHLNets**.



Then we obtain the restriction (mp_0, ϕ_1) of mp_1 along f_1 as pullback (1) below in **AHLNets**. Furthermore there is

$$g_2 \circ f_2 \circ mp_0 = g_1 \circ f_1 \circ mp_0 = g_1 \circ mp_1 \circ \phi_1 = mp_3 \circ \psi_1 \circ \phi_1$$

which by the pullback property of (2) implies that there is a unique AHL-morphism $\phi_2 : K_0 \to K_2$ such that diagram (2) and the outer square below commute.



By composition of pullbacks we have that (1)+(3) is a pullback. Since (PO) and the outer square commute there is (1)+(3) = (2)+(4) which implies that (2)+(4) is a pullback, and hence by pullback decomposition (2) is a pullback.

So we have that the above cube is a weak Van Kampen cube where all side faces are pullbacks and the bottom is a pushout. Hence, the Van Kampen property implies that the top face (i. e. the outer square) is a pushout in **AHLNets**. Since K_3 and its restrictions K_0, K_1 and K_2 all are AHL-process nets, the outer square in the diagram which is a pushout in **AHLNets** is also a gluing of AHL-process nets which by Fact 4.2.6 implies that K_1 and K_2 are composable w. r. t. (K_0, ϕ_1, ϕ_2). Hence, (mp_0, ϕ_1) and

(mp_0, ϕ_2) are agreement restrictions for mp_1 and mp_2 which means that mp_3 is an amalgamation of mp_1 and mp_2.

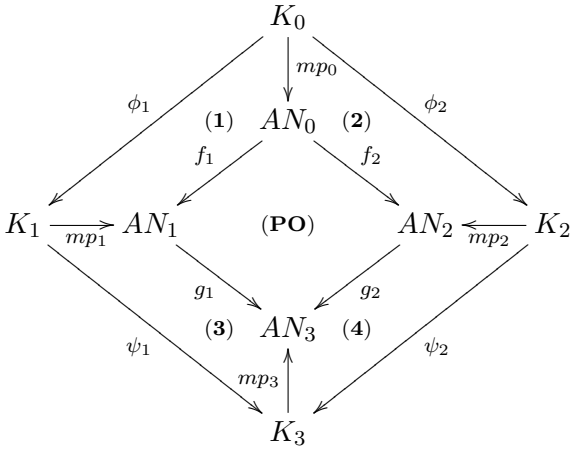
3. **Bijective Correspondence.** We define

$$Comp(\Big[mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2\Big]) = [mp_3]$$

where the AHL-process net K_3 is obtained as composition of AHL-process nets $K_3 = K_1 +_{(K_0,\phi_1,\phi_2)} K_2$ and the morphism $mp_3 : K_3 \to AN_3$ is the unique morphism induced by the pushout property of the corresponding pushout in **AHLNets**. Hence, $mp_3 = mp_1 \circ_{\phi_1,\phi_2} mp_2$ is unique up to isomorphism which means that the function *Comp* is well-defined. Moreover, we define

$$Decomp([mp_3]) = \Big[mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2\Big]$$

where $mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2$ is the amalgamation decomposition of mp_3 constructed as given in item 2 above. The amalgamation decomposition constructed via pullbacks (1)-(4) in **AHLNets** is unique up to isomorphism due to the uniqueness of pullbacks. Thus, the function *Decomp* is well-defined.



Given an agreeing span $mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2$ with respect to pushout (PO). Then by definition of agreement restrictions diagrams (1) and (2) above are pullbacks in **AHLNets**. The composition $mp_3 : K_3 \to AN_3$ is constructed via the pushout which is the outer square in the diagram above. Then the pushout (PO) is a weak Van Kampen square implying that (3) and (4) are pullbacks in **AHLNets**. Since the decomposition of mp_3 is constructed via pullback (1)-(4) and pullbacks are unique up to isomorphism the result is isomorphic to $mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2$, i. e.

$$Decomp(Comp(\Big[mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2\Big])) = \Big[mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2\Big]$$

Vice versa, given an AHL-process $mp_3 : K_3 \to AN_3$ the amalgamation decomposition $mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2$ of mp_3 is constructed via pullbacks (1)-(4) leading to the fact that (PO) is a weak Van Kampen square. This implies that the outer square is a pushout which defines exactly the composition of $mp_1 \overset{\phi_1}{\leftarrow} mp_0 \overset{\phi_2}{\rightarrow} mp_2$. Since pushouts are unique up to isomorphism there is

$$Comp(Decomp([mp_3])) = [mp_3]$$

Hence, *Comp* and *Decomp* are inverse to each other which means that they are bijections.

4. **Instantiations.** The proof for instantiated AHL-processes works completely analogously. Note that the restriction and gluing constructions for instantiations of AHL-process nets imply corresponding restriction or gluing of the underlying AHL-process net. Moreover, according to Fact 4.5.6, the restriction of an instantiated AHL-process is unique up to isomorphism. The uniqueness is also the case for the gluing of instantiated AHL-process nets as pushout which exists if and only if the corresponding span of AHL-process nets is composable (see Definition 4.5.7).

B.8 Proof of Theorem 4.6.8 (Direct Transformation of Instantiations Using Abstract Productions)

Given an abstract production for instantiations $\varrho : (inst_L, L) \overset{l}{\leftarrow} (inst_I, I) \overset{r}{\rightarrow} (inst_R, R)$, an instantiation $(inst, AN)$ and a (match) morphism $m : (inst_L, L) \rightarrow (inst, AN)$. We have to show that there exists a direct transformation of instantiations $(inst, AN) \overset{(\varrho,m)}{\Longrightarrow} (inst', AN')$, if and only if ϱ and m satisfy the instantiation condition for abstract productions and there exists a direct transformation of AHL-nets $AN \overset{Net(\varrho),m}{\Longrightarrow} AN'$ using the underlying production for AHL-nets $Net(\varrho) : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$.

We show the two directions of the proof separately.

If. Let ϱ and m satisfy the instantiation condition for abstract productions and let exist a direct transformation of AHL-nets $AN \overset{Net(\varrho),m}{\Longrightarrow} AN'$ using the underlying production for AHL-nets $Net(\varrho) : L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$. Then, according to Fact A.5.11 there exists a direct transformation $(inst, AN) \overset{\varrho,m}{\Longrightarrow} (inst', AN')$ of weak instantiations with pushouts (1) and (2) in **wInst** below.

$$\begin{array}{ccccc}
(inst_L, L) & \xleftarrow{\;l\;} & (inst_I, I) & \xrightarrow{\;r\;} & (inst_R, R) \\
{\scriptstyle m}\downarrow & (\mathbf{1}) & {\scriptstyle c}\downarrow & (\mathbf{2}) & {\scriptstyle n}\downarrow \\
(inst, AN) & \xleftarrow{\;d\;} & (inst_0, AN_0) & \xrightarrow{\;e\;} & (inst', AN')
\end{array}$$

According to Lemma A.5.17 there is a direct transformation of weak instantiations $(inst, AN) \overset{\varrho',m'}{\Longrightarrow} (inst', AN')$ using the data-shifting ϱ' of ϱ along m.

Then, due to the fact that the data-shifting ϱ' of ϱ along m is required to be a concrete production of instantiations, according to Fact A.5.12 there exists a direct transformation of instantiations $(inst, AN) \overset{\varrho',m'}{\Longrightarrow} (inst', AN')$ that coincides with the corresponding transformation of weak instantiations (see Fact A.5.10). Hence, $(inst_0, AN_0)$ and $(inst', AN')$ are concrete instantiations which means that the direct transformation $(inst, AN) \overset{\varrho,m}{\Longrightarrow} (inst', AN')$ given by pushouts (1) and (2) also is a direct transformation of concrete instantiations.

Only If.

Now, let there be a direct transformation of instantiations $(inst, AN) \overset{(\varrho,m)}{\Longrightarrow} (inst', AN')$. This means that there are pushouts (1) and (2) below in **wInst** which by construction of pushouts in **wInst** implies that there are corresponding pushouts (3) and (4) in **AHLNets**. Hence, there is a direct transformation $AN \overset{Net(\varrho),m}{\Longrightarrow} AN'$ of AHL-nets.

$$
\begin{array}{lccccc}
\varrho: & (inst_L, L) & \xleftarrow{l} & (inst_I, I) & \xrightarrow{r} & (inst_R, R) \\
& {\scriptstyle m}\downarrow & (1) & {\scriptstyle c}\downarrow & (2) & {\scriptstyle n}\downarrow \\
& (inst, AN) & \xleftarrow{d} & (inst_0, AN_0) & \xrightarrow{e} & (inst', AN') \\
Net(\varrho): & L & \xleftarrow{l} & I & \xrightarrow{r} & R \\
& {\scriptstyle m}\downarrow & (3) & {\scriptstyle c}\downarrow & (4) & {\scriptstyle n}\downarrow \\
& AN & \xleftarrow{d} & AN_0 & \xrightarrow{e} & AN'
\end{array}
$$

According to Lemma A.5.17 there is a direct transformation of weak instantiations $(inst, AN) \overset{\varrho', m'}{\Longrightarrow} (inst', AN')$ using the data-shifting ϱ' of ϱ along m, given by pushouts (5) and (6) below.

$$
\begin{array}{ccccc}
(inst'_L, L') & \xleftarrow{l'} & (inst'_I, I') & \xrightarrow{r'} & (inst'_R, R') \\
{\scriptstyle m'}\downarrow & (5) & {\scriptstyle c'}\downarrow & (6) & {\scriptstyle n'}\downarrow \\
(inst, AN) & \xleftarrow{d} & (inst_0, AN_0) & \xrightarrow{e} & (inst', AN')
\end{array}
$$

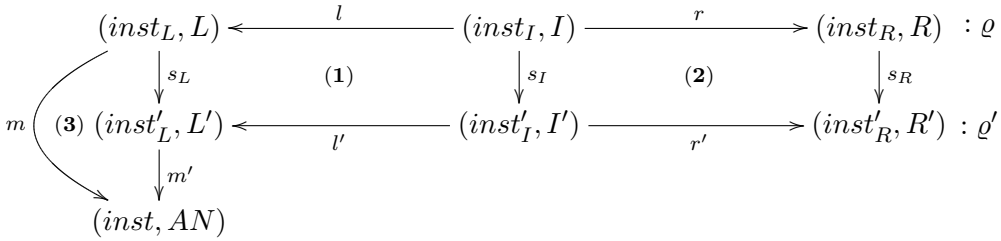
Due to the fact that ϱ' together with m' is a data-shifting, according to Definition 4.6.5 the data type part $wInstData(m')$ of m' is an isomorphism. Moreover, pushout (5) along $\mathcal{M}_{AHL}$-morphism l' is also pullback. Since the construction of pushouts and pullbacks of weak instantiations implies the corresponding construction of pushouts and pullbacks in **AHLNets** and thus also in **Algs**, we obtain that also $wInstData(c')$ and $wInstData(n')$ are isomorphisms. So, using Fact A.6.9, we obtain W-creations of m', c' and n' via $(inst, AN)$, $(inst_0, AN_0)$ and $(inst', AN')$, respectively, which are trivially given by m', c' and n', respectively. The domains of these morphisms are the unique instantiations corresponding to $(inst'_L, L')$, $(inst'_I, I')$ and $(inst'_R, R')$ (see Remark 4.3.8). Hence, the data-shifting ϱ' of ϱ along m is a concrete production, and thus ϱ and m satisfy the instantiation condition.

The proof for the instantiated AHL-process nets follows immediately from combination of the proof above and the fact that direct transformations of AHL-process nets are also direct transformations of AHL-nets. □

B.9 Proof of Fact 4.6.13 (Equivalence of Consistent Creation Condition and Instantiation Condition)

Given an abstract production for instantiations $\varrho : (inst_L, L) \overset{l}{\leftarrow} (inst_I, I) \overset{r}{\rightarrow} (inst_R, R)$, where the data type part of l and r are identities, a concrete instantiation $(inst, AN)$ and a match $m : (inst_L, L) \rightarrow (inst, AN)$. We have to show that ϱ and m satisfy the consistent creation condition if and only if they satisfy the instantiation condition.

Let ϱ' together with m' as depicted below be the data-shifting of ϱ along m. Then according to Definition 4.6.5 morphism m' has an isomorphic data type part which by Lemma A.5.5 implies that $(inst'_L, L)$ is a concrete instantiation. Moreover, by pushout (1) and $l \in \mathcal{M}_{AHL}$ we also have that $l' \in \mathcal{M}_{AHL}$. So we also have that l' has an isomorphic data type part and hence using again Lemma A.5.5 we obtain that also $(inst'_I, I)$ is a concrete instantiation. So the question whether or not the instantiation condition is satisfied can be reduced to the question if the right-hand side of the data-shifting is a concrete instantiation.



Further, due to the fact that $s_L : (inst_L, L) \to (inst'_L, L')$ is the data-image of weak instantiation $(inst_L, L)$ along m, we have by Definition 4.4.1 that $s_L : L \to L'$ is the data-image of AHL-net L along m, and thus by Definition 3.3.1 s_L has (m_Σ, m_A) as data type part. Then by Fact A.5.8 the pushouts (1) and (2) imply corresponding underlying pushouts in **AHLNets** which by componentwise construction of pushouts in **AHLNets** means that there are also pushouts in the **Algs**-component. So, since l and r have an identical data type part, the pushouts in **Algs** imply that also s_I and s_R have (m_Σ, m_A) as data type parts.

Moreover, by Definition 3.3.1 we know that

$$cond_{R'} = \mathcal{P}_{fin}(m_\Sigma^\#)(cond_R)$$

and by Definition 4.4.1 we know that $inst'_R = wFlat(s_R) \circ inst_R$[37] which means that for $(t, v) \in inst_{R,T}(T_R)$ we have

$$(s_{R,T}(t), m_A \circ v \circ (m_X|_{Var(t)})^{-1}) = wFlat(s_R)_T(t, v) \in inst'_{R,T}(T_{R'})$$

Now we distinguish between the two directions of the fact:

If. We consider the case that ϱ and m satisfy the instantiation condition. Then we have that $(inst'_R, R')$ is a concrete instantiation which means that all transition assignment in the instantiation are consistent assignments, i. e. for all $(t', v') \in inst'_{R,T}(T_{R'})$ we have that $v' \vDash cond_{R'}(t')$.

Now, let $t \in T_R \setminus r_T(T_I)$ with $inst_{R,T}(t) = (t, v)$. As shown above we know that $(s_{R,T}(t), m_A \circ v \circ (m_X|_{Var(t)})^{-1}) \in inst'_{R,T}(T_{R'})$, and thus, since all transition assignments in $(inst'_R, R')$ are consistent, we know that

$$m_A \circ v \circ (m_X|_{Var(t)})^{-1} \vDash cond_{R'}(s_{R,T}(t))$$

Due to the fact that s_R is a data-image, it has an identical T-component which means that $s_{R,T}(t) = t$. So, as we have shown above, for the conditions we know that

$$cond_{R'}(s_{R,T}(t)) = cond_{R'}(t) = \mathcal{P}_{fin}(m_\Sigma^\#(cond_R))$$

Hence, we have that

$$m_A \circ v \circ (m_X|_{Var(t)})^{-1} \vDash cond_{R'}(s_{R,T}(t)) \vDash \mathcal{P}_{fin}(m_\Sigma^\#(cond_R))$$

and since this holds for all $t \in T_R \setminus r_T(T_I)$, we have that ϱ and m satisfy the consistent creation condition.

Only If. We consider the case that ϱ and m satisfy the consistent creation condition. Note that the argumentation in the If-case works also in the other direction, i. e. the fact that

[37] In Definition 4.4.1 we consider only the case of concrete instantiations, where we use *Flat* instead of *wFlat*, but the construction works analogously for weak instantiations using *wFlat*.

the consistent creation condition is satisfied implies that the images of the transition assignments of transitions $t \in T_R \setminus r_T(T_I)$ are consistent assignments.

It remains to investigate all other transition assignments for transitions that are not in the image $s_{R,T}(T_R \setminus r_T(T_I))$. For all $t \in T_R$ with $t \in r_T(T_I)$ there exists a transition $t_0 \in T_I$ with $r_T(t_0) = t$, and also a $t' \in T_{I'}$ with $s_{I,T}(t_0) = t'$. Then by commutativity of diagram (2) we have that $s_{R,T}(t) = r'_T(t')$. Moreover, due to the fact that (2) is a pushout, there is also an underlying pushout of the T-component, which means that $s_{R,T}$ and r'_T are jointly surjective. Hence, for all transitions in $T_{R'}$ that do not have a preimage in $T_R \setminus r_T(T_I)$, we know that the transition has a preimage in $T_{I'}$.

So let $t \in T_{R'}$ and $t' \in T_{I'}$ such that $r'_T(t') = t$. For $inst'_{I,T}(t') = (t', v')$ we know that (t', v') is a consistent transition assignment, because $(inst'_I, I')$ is a concrete instantiation. Furthermore, since r has an identical data type part, the **Algs**-component of pushout (2) implies that the morphism r' has an isomorphic data type part and we can w. l. o. g. assume that the data type part (r'_Σ, r'_A) is an identity.

Then by the fact that AHL-morphisms preserve firing conditions, we obtain that

$$cond_{R'}(t) = cond_{R'}(r'_T(t') = \mathcal{P}_{fin}(r^{\#}_\Sigma)(cond_{I'}(t')) = cond_{I'}(t')$$

Moreover, using the fact that r'_T is a weak instantiation morphism, we obtain

$$\begin{aligned} inst'_{R,T}(t) &= inst'_{R,T}(r'_T(t')) = inst'_{R,T}(Skel(r')_T(t')) \\ &= wFlat(r')_T(inst'_{I,T}(t')) = wFlat(r')_T(t', v') \\ &= (r'_T(t'), v') = (t, v') \end{aligned}$$

This means that the image t of t' has the same assignment v'.

Now, the fact that (t', v') is a consistent transition assignment means that v' satisfies $cond_{I'}(t')$, and since $cond_{I'}(t') = cond_{R'}(t)$, we have that also (t, v') is a consistent transition assignment. Due to injectivity of $inst'_R$, the assignment v' is the only assignment that corresponds to t, and since this holds for all $t \in r'_T(T_{I'})$, we know that all transition assignments in $(inst'_R, R')$ are consistent. Hence, $(inst'_R, R')$ is a concrete instantiation which together with the fact that also $(inst'_L, L')$ and $(inst'_I, I')$ are concrete instantiations means that ϱ' is a concrete production, and thus the instantiation condition is satisfied. □

B.10 Proof of Theorem 4.8.4 (Extension of AHL-Process based on AHL-Net Transformation)

Given an AHL-net AN, an AHL-process $mp : K \to AN$ and a direct transformation $AN \overset{\varrho,m}{\Rightarrow} AN'$ with pushouts (1) and (2) in **AHLNets** as depicted in Figure 4.26 on page 102. We have to show that there exists an extension $mp' : K \to AN'$ of mp if and only if mp and ϱ, m satisfy the extension condition.

If. Let mp and p, m satisfy the extension condition. We define $mp_0 : K \to AN_0$ as $mp_0 = f^{-1} \circ mp$. Since f is injective, for the well-definedness of mp_0 it suffices to show that for all elements x in in K there exists an element y in AN_0 with $f(y) = mp(x)$. Let $p \in P_K$.

Case 1. There exists $x \in P_L$ with $m(x) = mp(p)$.
Then there is $x \in PP$ and since mp and ϱ, m satisfy the extension condition, we have $x' \in P_I$ with $l(x') = x$. Thus, we have $y = k(x')$ with $f(y) = f(k(x')) = m(l(x')) = m(x) = mp(p)$.

Case 2. There exists no $x \in P_L$ with $m(x) = mp(p)$.
Then by construction of pushout complements in **AHLNets** there exists $y \in AN_0$ with $f(y) = mp(p)$.

The proof for the existence of the transitions works analogously. Injectivitiy of f implies that we have a well-defined morphism $mp_0 : K \to AN_0$ with $f \circ mp_0 = f \circ f^{-1} \circ mp = mp$, and we obtain the required extension $mp' : K \to AN'$ by composition $mp' = g \circ mp_0$.

Only If. Let $mp' : K \to AN'$ be the extension of mp, i. e. there is $mp_0 : K \to AN_0$ with $f \circ mp_0 = mp$ and $g \circ mp_0 = mp'$. We have to show that all process points are gluing points. So, let $x \in PP$ and let w. l. o. g. $x \in P_L$. Then there is $p \in P_K$ with $mp(p) = m(x)$. Moreover, we have $y = mp_0(p) \in P_{AN_0}$ with $f(y) = f(mp_0(p)) = mp(p) = m(x)$. Since (1) is pushout in **AHLNets**, this implies that there is $x_0 \in P_I$ with $k(x_0) = y$ and $l(x_0) = x$. Hence, we have $x \in GP$. □

B.11 Proof of Theorem 4.8.14 (Process Evolution based on Action Evolution)

Given an action evolution $AN \overset{\varrho,m}{\Longrightarrow} AN'$ via production (ϱ^*, ϱ) with $\varrho^* = \coprod_{i \in \mathcal{I}} \varrho_i^*$, and a process $mp : K \to AN$.

We have to show that for every choice of matches for occurrences $(m_{i,o} : L_i^* \to K)_{i \in \mathcal{I}, o \in occ_i}$ (see Definition 4.8.12) there exists a production (ϱ^+, ϱ) for AHL-processes and a direct transformation $mp \overset{(\varrho^+,\varrho)}{\Longrightarrow} mp'$ as depicted in Figure 4.31.
Construction:

1. The action evolution pattern ϱ^+ is constructed as parallel production (componentwise coproduct) $\varrho^+ = L^+ \overset{l^+}{\leftarrow} I^+ \overset{r^+}{\rightarrow} R^+ = \coprod_{i \in \mathcal{I}} \coprod_{o \in occ_i} \varrho_i^*$ with coproduct injections $\mu_{i,o}^X : X_i^* \to X^+$ for $X \in \{L, I, R\}$, which together with morphisms $mp_X \circ \iota_i^X$ induce unique morphisms $mp_X^+ : X^+ \to X$ such that $(\varrho^+, \varrho) = mp_L^+ \overset{(l^+,l)}{\leftarrow} mp_I^+ \overset{(r^+,r)}{\rightarrow} mp_R^+$ is a production for AHL-processes (see Definition 4.8.7).

2. A match m^+ is induced by coproduct $L^+ = \coprod_{i \in \mathcal{I}} \coprod_{o \in occ_i} L_i^*$ and matches $m_{i,o}$.

3. Then, using Lemma A.7.3, we obtain a direct transformation $K \overset{\varrho^+,m^+}{\Longrightarrow} K'$ of AHL-process nets in the lower back of Figure 4.31.

4. The process $mp_0 : K_0 \to AN_0$ can be obtained by construction of K_0 as pullback in the left bottom of Figure 4.31, and the process $mp' : K' \to AN'$ is induced by universal property of the pushout in the lower right back of the cube.

We have to show that the construction above is well-defined.

1. Due to the fact that all productions ϱ_i for $i \in \mathcal{I}$ are single action evolution patterns, the parallel production $\varrho^+ = L^+ \overset{l^+}{\leftarrow} I^+ \overset{r^+}{\rightarrow} R^+ = \coprod_{i \in \mathcal{I}} \coprod_{o \in occ_i} \varrho_i^*$ is an action evolution pattern. Then, due to morphisms $mp_X \circ \iota_i^X$ for $X \in \{L, I, R\}$, the universal coproduct property of X^+ implies a unique $mp^+ : X^+ \to X$ such that $mp^+ \circ \mu_{i,o}^X = mp_X \circ \iota_i^X$.
It remains to show that $(\varrho^+, \varrho) = mp_L^+ \overset{(l^+,l)}{\leftarrow} mp_I^+ \overset{(r^+,r)}{\rightarrow} mp_R^+$ is a production for AHL-processes (see Definition 4.8.7), i. e. that we have $mp_L^+ \circ l^+ = l \circ mp_I^+$ and $mp_R^+ \circ r^+ = r \circ mp_I^+$.

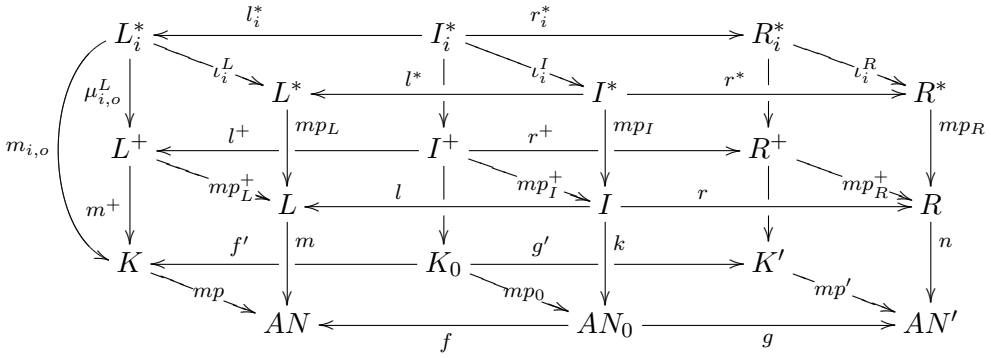


Figure B.11: Process evolution based on action evolution

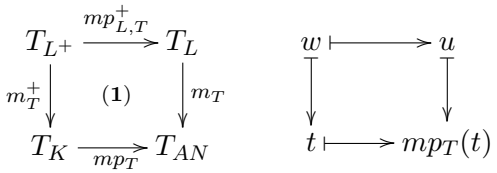
Considering the left-hand side, for all $i \in \mathcal{I}$ and $o \in occ_i$ we have

$$\begin{aligned} mp_L^+ \circ l^+ \circ \mu_{i,o}^I &= mp_L^+ \circ \mu_{i,o}^L \circ l_i^* = mp_L \circ \iota_i^L \circ l_i^* \\ &= mp_L \circ l^* \circ \iota_i^I = l \circ mp_I \circ \iota_i^I \\ &= l \circ mp_I^+ \circ \mu_{i,o}^I \end{aligned}$$

which by jointly epic $\mu_{i,o}^I$ implies $mp_L^+ \circ l^+ = l \circ mp_I^+$, and analogously we obtain also $mp_R^+ \circ r^+ = r \circ mp_I^+$.

2. By universal coproduct property of L^+ and morphisms $m_{i,o} : L_i^* \to K$, we obtain a unique morphism $m^+ : L^+ \to K$ such that $m^+ \circ \mu_{i,o}^L = m_{i,o}$.

3. In order to use Lemma A.7.3, we have to show that m^+ is transition-injective, i. e. that m_T^+ is injective. For this purpose we show that the diagram (1) below, corresponding to the transition-component of the lower left face in Figure B.11, is a pullback.



Using standard construction of pullbacks in **Sets**, in order to show that T_{L^+} is a pullback of K and L, we have to show that

$$T_{L^+} \cong \{(t, u) \in T_K \times T_L \mid mp_T(t) = m_T(u)\}$$

First we show that for $(t, u) \in T_K \times T_L$ with $mp_T(t) = m_T(u)$ there is $w \in T_{L^+}$ such that $m^+(w) = t$ and $mp_L^+(w) = u$. So let $t \in T_K$ and $u \in T_{L^*}$ which by bijection $mp_{L,T}$ implies that there is a unique $u' \in T_{L^*}$ with $mp_{L,T}(u') = u$. Then by $mp_T(t) = m_T(u) = m_T \circ mp_T(u')$, there is an $i \in \mathcal{I}$ such that $T_{L_i^*} = \{v\}$ and $\iota_i^L(v) = u'$, and therefore there is a match unique match $m_{i,o} : L_i^* \to K$ with $m_{i,o}(v) = t$ because $(m_{i,o})_{i \in \mathcal{I}, o \in occ_i}$ is a choice of matches. This implies that we also have $w \in T_{L^+}$ with $w = \mu_{i,o}^L(v)$ such that

$$mp_L^+(w) = mp_L^+ \circ \mu_{i,o}^L(v) = mp_L \circ \iota_i^L(v) = mp_L(u') = u$$

and

$$m^+(w) = m^+ \circ \mu^L_{i,o}(v) = m_{i,o}(v) = t$$

Due to construction of L^+ as coproduct, the transition w is uniquely determined by i and o.

Moreover, we show that the diagram commutes. Given $w \in T_{L^+}$, there is $i \in \mathcal{I}$ and $o \in occ_i$ such that $w = \mu^L_{i,o}(v)$ for v being the single transition in $T_{L^*_i}$. Let $t \in T_K$ with $t = m^+(w)$ and $u \in T_L$ with $u = mp^+(w)$, then we have

$$\begin{aligned} mp(t) &= mp \circ m^+(w) = mp \circ m^+ \circ \mu^L_{i,o}(v) = mp \circ m_{i,o}(v) = m \circ mp_L \circ \iota^L_i(v) \\ &= m \circ mp^+_L \circ \mu^L_{i,o}(v) = m \circ mp^+_L(w) = m(u) \end{aligned}$$

Hence, (1) is a pullback in **Sets**.

Since it is required for an action evolution to have a transition-injective match m (see Definition 4.8.9), we have that m_T is injective and thus a monomorphism in **Sets**. So, by closure of monomorphisms under pullbacks, it follows that also m^+_T is injective.

Therefore, using Lemma A.7.3, we obtain a direct transformation $K \overset{o^+,m^+}{\Longrightarrow} K'$ of AHL-process nets in the lower back of Figure B.11.

4. In order to show that the left bottom face of Figure B.11 is a pullback, we first show that the diagram (2)+(1) below is a pullback in **Sets**.

$$\begin{array}{ccc} T_{I^+} & \xrightarrow{mp^+_{I,T}} & T_I \\ {\scriptstyle l^+_T}\downarrow & (\mathbf{2}) & \downarrow{\scriptstyle l_T} \\ T_{L^+} & \xrightarrow{mp^+_{L,T}} & T_L \\ {\scriptstyle m^+_T}\downarrow & (\mathbf{1}) & \downarrow{\scriptstyle m_T} \\ T_K & \xrightarrow[mp_T]{} & T_{AN} \end{array}$$

Using again standard construction of pullbacks in **Sets**, we have to show that

$$T_{I^+} \cong \{(t,u) \in T_K \times T_I \mid mp_T(t) = m_T \circ l_T(u)\}$$

So let $t \in T_K$ and $u \in T_I$ with $mp_T(t) = m_T \circ l_T(u)$. Then for $u' = l_T(u) \in T_L$ we have that $mp_T(t) = m_T(u')$ which by pullback (1) shown in item 3 implies that there is $v \in T_{L^+}$ with $m^+_T(v) = t$ and $mp^+_{L,T}(v) = u'$. Then by construction of L^+ as coproduct, there is $i \in \mathcal{I}$ and $o \in occ_i$ such that we have $w \in T_{L^*_i}$ with $\mu_{i,o,T}(w) = v$ and $m_{i,o}(w) = m^+_T \circ \mu_{i,o,T}(w) = m^+_T(v) = t$.

Now, consider diagram (3) below, corresponding to the upper left front face of Figure B.11 which is required to be a pullback by Definition 4.8.9. Due to componentwise construction of pullbacks along $\mathcal{M}_{AHL}$ and the fact that $l \in \mathcal{M}_{AHL}$, we also have that diagram (4) below is a pullback in **Sets**.

$$\begin{array}{ccc} I^*_i & \xrightarrow{l^*_i} & L^*_i \\ {\scriptstyle \iota^I_i}\downarrow & & \downarrow{\scriptstyle \iota^L_i} \\ I^* & \xrightarrow{l^*} & L^* \\ {\scriptstyle mp_I}\downarrow & (\mathbf{3}) & \downarrow{\scriptstyle mp_L} \\ I & \xrightarrow{l} & L \end{array} \qquad \begin{array}{ccc} T_{I^*_i} & \xrightarrow{l^*_{i,T}} & T_{L^*_i} \\ {\scriptstyle \iota^I_{i,T}}\downarrow & & \downarrow{\scriptstyle \iota^L_{i,T}} \\ T_{I^*} & \xrightarrow{l^*_T} & T_{L^*} \\ {\scriptstyle mp_{I,T}}\downarrow & (\mathbf{4}) & \downarrow{\scriptstyle mp_{L,T}} \\ T_I & \xrightarrow{l_T} & T_L \end{array} \qquad \begin{array}{ccc} x' & \longmapsto & w \\ \downarrow & & \downarrow \\ x & \longmapsto & w' \\ \downarrow & & \downarrow \\ u & \longmapsto & u' \end{array}$$

So, by $w' = \iota^L_{i,T}(w) \in T_{L^*}$ and $u \in T_I$ with

$$\begin{aligned} mp_{L,T}(w') &= mp_{L,T} \circ \iota^L_{i,T}(w) = mp^+_{L,T} \circ \mu^L_{i,o,T}(w) \\ &= mp^+_{L,T}(v) = u' = l_T(u) \end{aligned}$$

due to pullback (4) there is $x \in T_{I^*}$ with $l^*_T(x) = w'$ and $mp_{I,T}(x) = u$. Then, since l^*_T is a coproduct $\coprod_{i\in\mathcal{I}} l^*_{i,T}$, it follows that there is also $x' \in T_{I^*_i}$ with $\iota^I_{i,T}(x') = x$ and $l^*_{i,T}(x') = w$. Thus, there is also $y \in T_{I^+}$ with $\mu^I_{i,o,T}(x') = y$, and we have

$$l^+_T(y) = l^+_T \circ \mu^I_{i,o,T}(x') = \mu^L_{i,o,T} \circ l^*_{i,T}(x') = \mu^L_{i,o,T}(w) = v$$

Hence, we have

$$m^+_T \circ l^+_T(y) = m^+_T(v) = t$$

and

$$mp^+_{I,T}(y) = mp^+_{I,T} \circ \mu_{i,o,T}(x') = mp_{I,T} \circ \iota^I_i(x') = mp_{I,T}(x) = u$$

which means that $y \in T_{I^+}$ is the required transition corresponding to (t, u). Due to construction of I^+ as coproduct, y is uniquely determined by i and o.

Now, we construct T_{K_0} as pullback in the bottom of the cube shown in Figure B.12 in **Sets**, inducing a unique morphism $k^+_T : T_{I^+} \to T_{K_0}$ such that the cube commutes because

$$\begin{aligned} mp_T \circ m^+_T \circ l^+_T &= m_T \circ mp^+_{L,T} \circ l^+_T \\ &= m_T \circ l_T \circ mp^+_{I,T} \\ &= f_T \circ k_T \circ mp^+_{I,T} \end{aligned}$$

Due to componentwise construction of pushouts in **AHLNets** along $\mathcal{M}_{AHL}$, we have that the front face, corresponding to the transition-component of the pushout in the left bottom of Figure B.11, is a pushout. Moreover, the left face is the pullback (1) shown in item 3, and the top face is the diagram (2) which is a pullback by decomposition of pullbacks (1) and (1)+(2). Then by commutativity of the cube also the bottom and right face of the cube form a pullback which by decomposition of pullbacks implies that also the right face is a pullback. Hence, with monomorphism l_T and the fact that **Sets** is adhesive (see Remark A.1.10, and Theorem 4.6 in [EEPT06b]), the Van Kampen property (see Definition A.1.9 and Remark A.1.10) implies that the back face of the cube is a pushout in **Sets**.

Further, the back of the cube in Figure B.12 corresponds to the transition-component of the pushout in the bottom-left back face in Figure B.11 because the pushout along $l^+ \in \mathcal{M}_{AHL}$ can be constructed componentwise, and **Sets** has unique pushout complements. Now, since $\mathcal{M}$-morphisms are closed under pushouts and $l \in \mathcal{M}_{AHL}$, we have that $f \in \mathcal{M}_{AHL}$, and by componentwise construction of pushouts and the fact that l_P is a bijection, we also have that f_P is a bijection. Thus, using Fact A.6.4 we obtain that f is a T-creation of f_T via AN which by the commuting square in the bottom of Figure B.12 implies that there exists a unique AHL-morphism $mp_0 : K_0 \to AN_0$ such that $f \circ mp_0 = mp \circ f'$ and $T(mp_0) = mp_{0,T}$. Analogously to f also f' is a T-creation

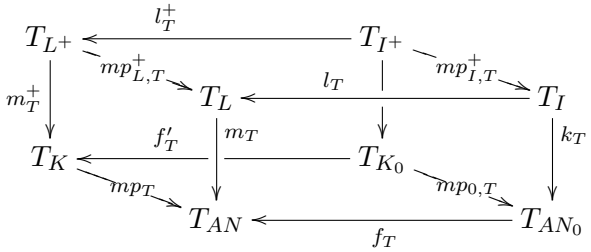


Figure B.12: Cube in **Sets**

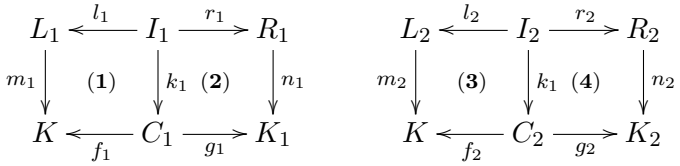
and thus using Fact A.1.27 we obtain that the left bottom of the cube in Figure B.11 is a pullback.

Finally, due to commutativity of all squares in the cube, the pushout in the bottom-right back of Figure B.11 implies a unique $mp' : K' \to AN'$ such that also the two new squares commute. Hence, by the fact that K' is an AHL-process net, we have a direct transformation $mp \overset{(\varrho^+,\varrho)}{\Longrightarrow} mp'$. □

B.12 Proof of Theorem 5.2.4 (Local Church-Rosser Theorem for AHL-Process Net Transformations)

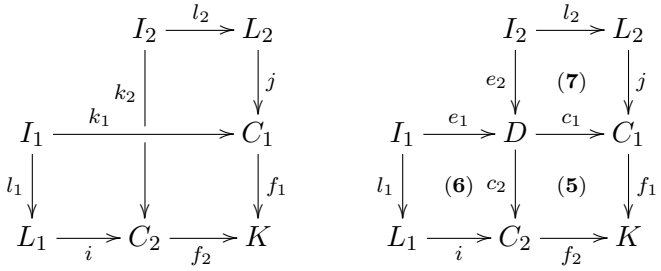
The proof works very similar to the general proof for weak adhesive HLR categories in [EEPT06b]. However, we need to recall the construction in order to check that the resulting transformations are in fact direct transformations of AHL-process nets and that the resulting independences are strong.

1. Given two strongly parallel independent direct transformations $K_0 \overset{\varrho_1,m_1}{\Longrightarrow} K_1$ and $K_0 \overset{\varrho_2,m_2}{\Longrightarrow} K_2$ of AHL-process nets, then there exist morphisms $i : L_1 \to C_2$ and $j : L_2 \to C_1$ such that $f_2 \circ i = m_1$ and $f_1 \circ j = m_2$, and we have that ϱ_1 and $g_2 \circ i$ as well as ϱ_2 and $g_1 \circ j$ satisfy the transformation condition for AHL-process nets.

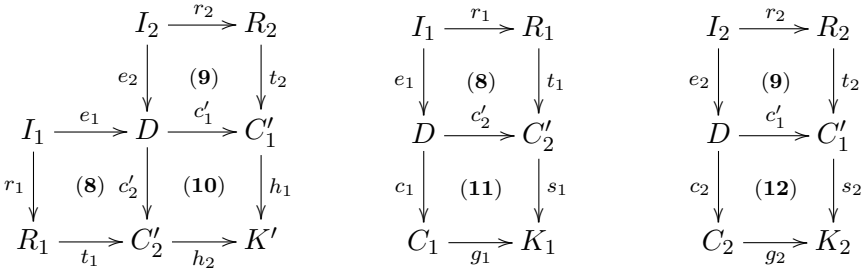


We combine the pushouts (1) and (3) in **AHLNets** as in the left diagram below. Then, by construction of the pullback (5) in the right diagram below, we obtain morphisms $e_1 : I_1 \to D$ and $e_2 : I_2 \to D$ such that $c_1 \circ e_1 = k_1$ and $c_2 \circ e_2 = k_2$, and diagrams (6) and (7) commute.

Now, by $l_1, l_2 \in \mathcal{M}_{AHL}$ and pushouts (1) and (3) we also have that $f_1, f_2 \in \mathcal{M}_{AHL}$. Further, by $\mathcal{M}_{AHL}$-morphisms l_1 and f_2, pushout (6)+(5) and pullback (5) in **AHLNets** by Fact A.4.11 (and $\mathcal{M}_{AHL} \subseteq \mathcal{M}'_{AHL}$) it follows that (6) and (5) are pushouts and pullbacks, and analogously, that (7) is a pushout and a pullback.

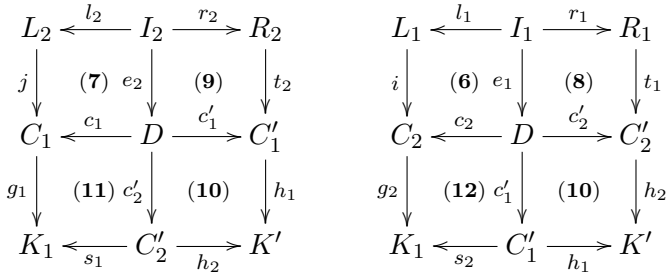


Now, we construct the pushouts (8) over e_1 and $r_1 \in \mathcal{M}_{AHL}$, (9) over e_2 and $r_2 \in \mathcal{M}_{AHL}$ and (10) over $c_1, c_2 \in \mathcal{M}_{AHL}$ in **AHLNets**.



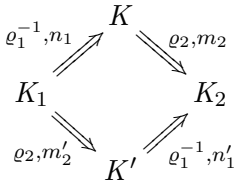
From pushout (8), we obtain a morphism $s_1 : C'_2 \to K_1$ such that (11) commutes and (2)=(8)+(11), and due to pushout decomposition we obtain that (11) is pushout. Analogously we obtain pushout (12) in **AHLNets**.

Combining all these pushouts, we obtain direct transformations of AHL-nets $K_1 \overset{\varrho_2}{\Rightarrow} K'$ and $K_2 \overset{\varrho_1}{\Rightarrow} K'$:



Due to the fact that the parallel independence is strong, we have that ϱ_2 and $g_1 \circ j$ as well as ϱ_1 and $g_2 \circ i$ satisfy the transformation condition. Thus, by Theorem 4.2.11 we have that $K_1 \overset{\varrho_2}{\Rightarrow} K'$ and $K_2 \overset{\varrho_1}{\Rightarrow} K'$ are direct transformations of AHL-process nets. The morphisms into the context net of the respected other direct transformation that is required for sequential independence are given by $t_1 : R_1 \to C'_2$ with $s_1 \circ t_1 = n_1$, because (8)+(11)=(2), and $t_2 : R_2 \to C'_2$ with $s_2 \circ t_2 = n_2$, because (9)+(12)=(4). Note that the pushouts above also represent direct transformations of AHL-process nets $K' \overset{\varrho_2^{-1}}{\Rightarrow} K_1$ at match $h_1 \circ t_1$ and $K' \overset{\varrho_1^{-1}}{\Rightarrow} K_2$ at match $h_2 \circ t_2$. Since by Theorem 4.2.11 satisfaction of the transformation condition is necessary for the existence direct transformations of AHL-process nets, we have that ϱ_2^{-1} and $h_1 \circ t_1$ as well as ϱ_1^{-1} and $h_2 \circ t_2$ satisfy the transformation condition. Hence, we have strongly sequentially independent direct transformations $K \Rightarrow K_1 \Rightarrow K'$ and $K \Rightarrow K_2 \Rightarrow K'$.

2. Given two strongly sequentially independent direct transformations $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2'}{\Longrightarrow} K'$ of AHL-process nets with comatches n_1 and n_2' respectively, using Remark 5.2.3 we obtain strongly parallel independent direct transformations $K \overset{\varrho_1^{-1},n_1}{\Longleftarrow} K_1 \overset{\varrho_2,m_2'}{\Longrightarrow} K'$. Then by part 1 of the proof there are strongly sequentially independent direc transformations $K_1 \overset{\varrho_1^{-1},n_1}{\Longrightarrow} K \overset{\varrho_2,m_2}{\Longrightarrow} K_2$ and $K_1 \overset{\varrho_2,m_2'}{\Longrightarrow} K' \overset{\varrho_1^{-1},n_1'}{\Longrightarrow} K_2$. Applying again Remark 5.2.3 to the first transformation means that $K_1 \overset{\varrho_1,n_1}{\Longleftarrow} K \overset{\varrho_2,m_2}{\Longrightarrow} K_2$ are the required parallel independent direct transformations.

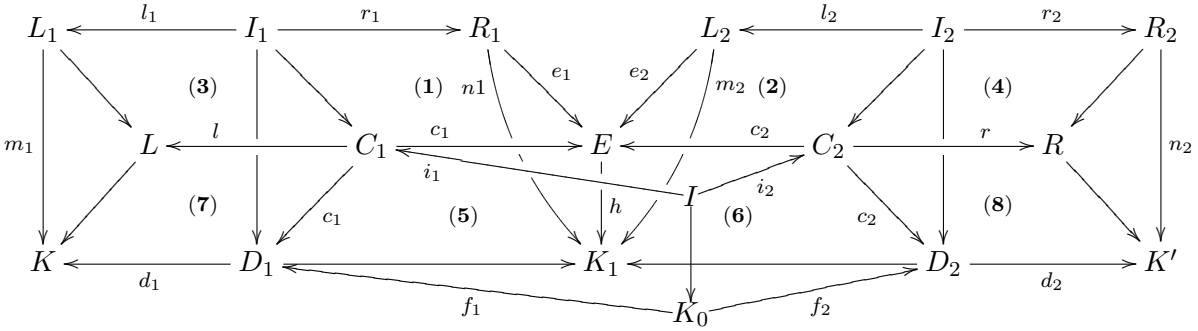


□

B.13 Proof of Theorem 5.2.14 (Concurrency Theorem for AHL-Process Net Transformations)

We have to show that the synthesis and analysis constructions for strong E-dependency relations (E, e_1, e_2) in Theorem 5.2.14 exist, and that they establish a bijective correspondence.

1. *(Synthesis)* Given an E-related transformation sequence of AHL-process nets $K \Rightarrow K_1 \Rightarrow K'$ via ϱ_1 and ϱ_2, by Definition 4.2.2 the direct transformations are also direct transformations of AHL-nets. Moreover, according to Remark 5.2.12 the strong E-dependency relation (E, e_1, e_2) is also a an E-dependency relation. Thus, by item 1 of Theorem 5.1.12 we obtain direct transformation of AHL-nets $K \Rightarrow K'$ via $\varrho_1 *_E \varrho_2$ such that we have pushouts (1)-(8)in **AHLNets**, and also the squares in front of (5) and (6) are pushouts:



Since K and K' already are AHL-process nets by assumption, it suffices to show that K_0 is an AHL-process net, in order to show that $K \Rightarrow K'$ is a transformation of AHL-process nets. From $l \circ i_1 \in \mathcal{M}_{AHL}$ and the pushout in front of pushout (5) composed with pushout (7), we obtain that also $d_1 \circ f_1 \in \mathcal{M}_{AHL}$ which means that $d_1 \circ f_1$ is transition-injective, and thus, by Lemma A.7.1 we have that K_0 is an AHL-process net.

Further, by composition of pushout (5) and the pushout in front of it, we have that AHL-process net K_1 is the gluing of E and K_0 over $(I, k, c_1 \circ i_1)$ which by Fact 4.2.6 implies that E and K_0 are composable w. r. t. $(I, k, c_1 \circ i_1)$. Hence, $K \Rightarrow K'$ is a strongly E-concurrent direct transformation of AHL-process nets.

2. *(Analysis)* Given a strongly E-concurrent direct transformation of AHL-process nets $K \Rightarrow K'$ via $\varrho_1 *_E \varrho_2$, from Definition 4.2.2 we know that $K \Rightarrow K'$ is also a direct

transformation of AHL-nets. Thus, by item 2 of Theorem 5.1.12 we obtain again the diagram above with pushouts (1)-(8)in **AHLNets**. Due to closure of $\mathcal{M}$-morphisms under pushouts, from $l_1, r_2 \in \mathcal{M}_{AHL}$ we obtain $d_1, d_2 \in \mathcal{M}_{AHL}$ which means that d_1 and d_2 are transition-injective and thus by Lemma A.7.1 we have that D_1 and D_2 are AHL-process nets. Since $K \Rightarrow K'$ is a strongly E-concurrent direct transformation of AHL-process nets, we have that E and K_0 are composable w. r. t. $(I, k, c_1 \circ i_1)$. Then, from composition of pushout (5) and the pushout in front of it, and the uniqueness of pushouts, we obtain that K_1 is the gluing of AHL-process nets $E +_I K_0$, and thus it is also an AHL-process net (see Fact 4.2.6).

Hence, with K, D_1, K_0, D_2 and K' being AHL-process nets, we have a direct transformation of AHL-process nets $K \Rightarrow K_1$ via ϱ_1 with pushouts (3)+(7) and (1)+(5), and a direct transformation of AHL-process nets $K_1 \Rightarrow K'$ via ϱ_2 with pushouts (2)+(6) and (4)+(8).

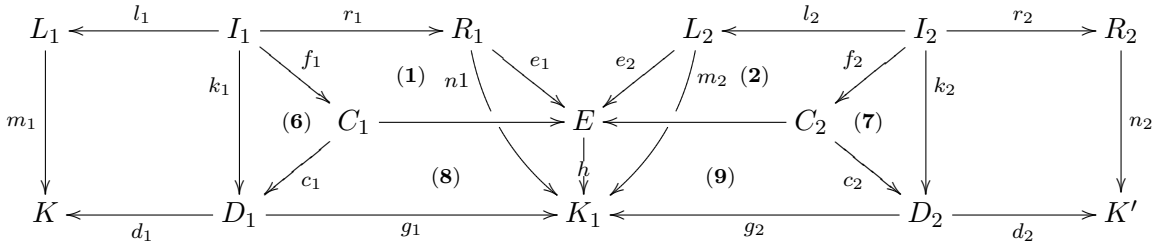
3. *(Bijective correspondence)* The synthesis and analysis constructions lead to exactly the same AHL-nets as the corresponding constructions in Theorem 5.1.12, and thus, by item 3 of Theorem 5.1.12 the constructions are inverse to each other. □

B.14 Proof of Fact 5.2.15 (Construction of Strongly E-Related AHL-Process Net Transformations)

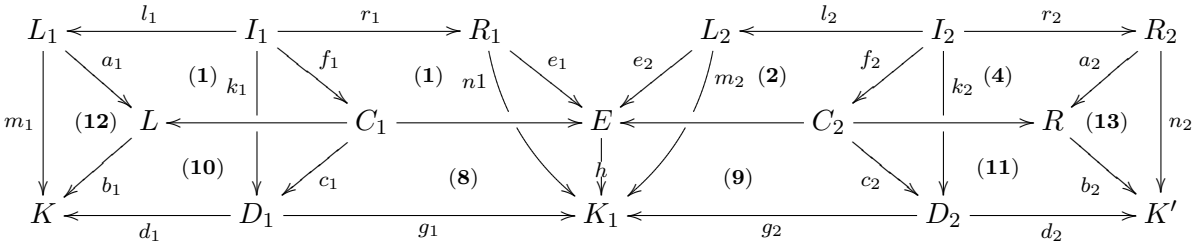
We have to show that for each pair of direct transformations of AHL-process nets $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2}{\Longrightarrow} K'$ we have a strong E-dependency relation E such that $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2}{\Longrightarrow} K'$ is strongly E-related. The proof works very similar to the corresponding proof of Fact 5.29 in [EEPT06b]. However, we need to extend the proof in order to verify that the resulting E-dependency relation is strong.

Given a sequence of direct transformations of AHL-process nets $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2}{\Longrightarrow} K'$, using Fact A.4.4 we can construct the $\mathcal{E}'$-$\mathcal{M}'_{AHL}$ pair factorisation $(e_1, e_2) \in \mathcal{E}'$ and $h \in \mathcal{M}'_{AHL}$ of n_1 and m_2 with $h \circ e_1 = n_1$ and $h \circ e_2 = m_2$.

Now, we construct the pullbacks (8) and (9) below. Since $h \circ e_1 \circ r_1 = n_1 \circ r_1 = g_1 \circ k_1$, from universal property of pullback (8) we obtain a unique AHL-morphism $f_1 : I_1 \to C_1$ such that (1) and (6) commute. Then, since $h \in \mathcal{M}'_{AHL}$, $r_1 \in \mathcal{M}_{AHL}$, (8) is pullback and (1)+(8) is a pushout, from the fact that $(\mathbf{AHLNets}, \mathcal{M}_{AHL})$ with $\mathcal{M}'_{AHL}$ has the $\mathcal{M}$-$\mathcal{M}'$ pushout-pullback decomposition property (Fact A.4.11) it follows that (1) and (8) are pushouts. Analogously, we obtain from pullback (9) a morphism $f_2 : I_2 \to C_2$ such that (7) commutes and (2) and (9) are pushouts.



The construction above can be repeated, constructing pullbacks (10) and (11), leading to AHL-morphisms $a_1 : L_1 \to L$ such that (3) and (12) commute, and $a_2 \to R_2 \to R$ such that (4) and (13) commute. By monomorphism h and pullbacks (8) and (9), we also have monomorphisms $c_1, c_2 \in \mathcal{M}'_{AHL}$, and thus, we obtain pushouts (3), (10), (4) and (11), using the $\mathcal{M}$-$\mathcal{M}'$ pushout-pullback decomposition property (Fact A.4.11).



Further, due to pullbacks (10) and (11) and monomorphisms c_1 and c_2, we also obtain monomorphisms b_1 and b_2. Then from the fact that all AHL-nets in the bottom row of the diagram above are AHL-process nets, due to transition-injective AHL-morphisms b_1, c_1, h, c_2 and b_2 by Lemma A.7.1 we have that also L, C_1, E, C_2 and R are AHL-process nets. This means that diagrams (1) and (3) correspond to a direct transformation of AHL-process nets $E \Rightarrow L$ via ϱ_1^{-1} at match e_1, and diagrams (2) and (4) correspond to a direct transformation of AHL-process nets $E \Rightarrow R$ via ϱ_2 at match e_2. Hence, by Theorem 4.2.11 we have that ϱ_1^{-1} and e_1 as well as ϱ_2 and e_2 satisfy the transformation condition of AHL-process nets which by Definition 5.2.11 means that (e_1, e_2) is a strong E-dependency relation such $K \overset{\varrho_1,m_1}{\Longrightarrow} K_1 \overset{\varrho_2,m_2}{\Longrightarrow} K'$ is strongly E-related. □

B.15 Proof of Theorem 5.3.3 (Concrete Realisation of AHL-Process)

1. *(Instantiation)* Let $\hat{v} : \hat{X} \to A$ be a solution for the realisation conditions of K. We define a P/T-morphism $inst : Skel(K) \to Flat(K)$ by

   - $inst_P(p) = (\hat{v}(p), p)$, and
   - $inst_T(t) = (t, \hat{v} \circ d_t)$.

   We have for all $p \in P_K$ that $\hat{v}(p) \in A_s$ for $s = type_K(p)$ and therefore $inst_P(p) \in A \otimes P_K$. Moreover, the composition of $d_T : Var(t) \to \biguplus_{t \in T_K} Var(t) \subseteq \bar{X}$ and $\hat{v} : \bar{X} \to A$ yields an assignment $\hat{v} \circ d_T : Var(t) \to A$. In order to show that $inst_T$ is well-defined it remains to show that the defined assignments are consistent. So let $t \in T_K$ with $inst_T(t) = (t, v)$, and let $(e_l, e_r) \in cond_K(t)$. We have to show that $v^*(e_l) = v^*(e_r)$. Due to the fact that $(e_l, e_r) \in cond_K(t)$, there is $(d_t^*(e_l), d_t^*(e_r)) \in Cond_K \subseteq Real_K$, and since $\hat{v}$ is a solution for the realisation conditions, we have $\hat{v}^*(d_t^*(e_l) = \hat{v}^*(d_t^*(e_r))$. Thus, we have

   $$\begin{aligned} v^*(e_l) &= (\hat{v} \circ d_t)^*(e_l) = \hat{v}^*(d_t^*(e_l) = \hat{v}^*(d_t^*(e_r)) \\ &= (\hat{v} \circ d_t)^*(e_r) = v^*(e_r) \end{aligned}$$

   which means that v is a consistent assignment, and hence $inst_T$ is well-defined.

   In order to show that $inst$ is a P/T-morphism, we have to show that $inst$ is compatible with pre and post domains. So, for the pre domains, we have to show that for all $t \in T_K$: $inst_P^{\oplus} \circ pre_{Skel(K)}(t) = pre_{Flat(K)} \circ inst_T(t)$. Let $t \in T_K$ with $pre_K(t) = \sum_{i=1}^{n}(term_i, p_i)$. Due to unarity of AHL-process net K, all p_i in $pre_K(t)$ are pairwise distinct, and therefore, it suffices to show that for all p_i we have for $inst_P(p_i) = (a, p_i)$ and $inst_T(t) = (t, v)$ that $a = v^*(term_i)$. According the the definition of $inst$ above, we have $a = \hat{v}(p)$, and $v^*(term_i) = (\hat{v} \circ d_t)^*(term_i) = \hat{v}^*(d_t^*(term_i))$. Together with the fact that $(term_i, p_i) \leq pre_K(t)$, we have that

   $$(d_t^*(term_i), p_i) \in V_{\cup}((d_t^* \otimes id_{P_K})^{\oplus}(pre_K(t))) \subseteq Pre_K \subseteq Real_K$$

and since $\hat{v}$ is a solution for the realisation conditions of K, we have

$$v^*(term_i) = \hat{v}^*(d_t^*(term_i)) = \hat{v}^*(p_i) = a$$

The compatibility of the post domains holds analogously, and hence, *inst* is a well-defined P/T-morphism.

Finally, the fact that $proj(K) \circ inst = id_{Skel(K)}$ can be easily seen in the definitions of $inst_P$ and $inst_T$ above. Hence, *inst* is a concrete instantiation of K.

2. *(Solution)* Let $(inst, K)$ be a concrete instantiation. We define an assignment $\hat{v} : \hat{X} \to A$ as follows:

   - For $p \in P_{K,s}$ with $inst_P(p) = (a, p)$ let $\hat{v}(p) = a$, and
   - for $\hat{x} \in \biguplus_{t \in T_K} Var(t)_s$ with $t \in T_K$ and $x \in Var(t)_s$ such that $d_{t,s}(x) = \hat{x}$ and $inst_T(t) = (t, v)$, let $\hat{v}(\hat{x}) = v(x)$.

   Note that for every $\hat{x} \in \biguplus_{t \in T_K} Var(t)_s$ there exists a unique $t \in T_K$ and $x \in Var(t)_s$ such that $d_{t,s}(x) = \hat{x}$ due to the fact that $\biguplus_{t \in T_K} Var(t)_s$ is a disjoint union. Furthermore, $x \in Var(t)_s$ with $d_{t,s}(x) = \hat{x}$ is unique because $d_{t,s}$ is an injection. Moreover, for every $t \in T_K$, we have for all $x \in Var(t)$ and $inst_T(t) = (t, v)$ that $\hat{v}(d_t(x)) = v(x)$, and thus

   $$\hat{v} \circ d_t = v.$$

   We have to show that the assignment $\hat{v}$ is a solution for the realisation conditions, i. e. that $(A, \hat{v}) \vDash Real_K$ which means that for all equations $(e_l, e_r) \in Real_K$ there is $\hat{v}^*(e_l) = \hat{v}^*(e_r)$.

   Case 1. $(e_l, e_r) \in Cond_K$.

   Then there exists at least one transition $t \in T_K$ such that $(e_l, e_r) = (d_T^*(e_l'), d_T^*(e_r'))$ and $(e_l', e_r') \in cond_K(t)$. Moreover, there is $(t, v) = inst_T(t)$ with $v : Var(t) \to A$ such that $(A, v) \vDash cond_K(t)$, because *inst* is a concrete instantiation, and especially we have that $(A, v) \vDash (e_l', e_r')$, i. e. $v^*(e_l') = v^*(e_r')$. Further, we know that all variables in e_l' and e_r' are in $Var(t)$, and thus all variables in e_l and e_r are in $d_T(Var(t))$. Now, this implies

   $$\begin{aligned} \hat{v}^*(e_l) &= \hat{v}^*(d_T^*(e_l')) = (\hat{v} \circ d_T)^*(e_l') = v^*(e_l') \\ &= v^*(e_r') = (\hat{v} \circ d_T)^*(e_r') = \hat{v}^*(d_T^*(e_r')) \\ &= \hat{v}^*(e_r) \end{aligned}$$

   which means that $(A, \hat{v}) \vDash (e_l, e_r)$.

   Case 2. $(e_l, e_r) \in Pre_K$.

   Then there is $e_r = p$ for some $p \in P_{K,s}$ and $e_l = d_{t,s}^*(term)$ for some $t \in T_K$ and $term \in T_\Sigma(Var(t))_s$ such that $(term, p) \leq pre_K(t)$. By the fact that $(term, p) \leq pre_K(t)$, we have that $p \leq pre_{Skel(K)}(t)$. Due to the fact that $inst : Skel(K) \to Flat(K)$ is a P/T-morphism that is compatible with pre and post domains of transitions, we have

   $$\begin{aligned} inst_P^{\oplus} \circ pre_{Skel(K)}(t) &= pre_{Flat(K)} \circ inst_T(t) \\ &= pre_{Flat(K)}(t, v) = pre_A(t, v) \end{aligned}$$

for some $v : Var(t) \to A$ such that $inst_T(t) = (t, v)$. Since K is an AHL-process net, it is unary, implying that for $term'$ with $(term', p) \leq pre_K(t)$ we have $term' = term$. Accordingly for any $(a, p) \leq pre_A(t, v)$ there is $a = v^*(term)$ and $(a, p) = inst_P(p)$.

So, according to the definition of $\hat{v}$ above, we have

$$\hat{v}^*(e_r) = \hat{v}^*(p) = \hat{v}(p) = a \text{ for } (a, p) = inst_P(p)$$

for the right-hand side, and

$$\begin{aligned}\hat{v}^*(e_l) &= \hat{v}^*(d_t^*(term)) = (\hat{v} \circ d_t)^*(term) \\ &= v^*(term) = a\end{aligned}$$

for the left-hand side. Thus, we have $\hat{v}^*(e_l) = a = \hat{v}^*(e_r)$.

Case 3. $(e_l, e_r) \in Post_K$.

This case works analogously to case 2.

3. *(Bijective correspondence)* The construction from instantiations to solutions in item 2 can be interpreted as a function $s : Inst(K) \to Sol(K)$, and the construction from solutions to instantiations as a function $i : Sol(K) \to Inst(K)$. We show that s and i are inverse isomorphisms.

   Let $(inst, K) \in Inst(K)$ be an instantiation and $\hat{v} = s(inst, K)$. As stated in item 2, for all $p \in P_K$ and $inst_P(p) = (a, p)$ we have $\hat{v}(p) = a$, and for all $t \in T_K$ and $inst_T(t) = (t, v)$ we have $\hat{v} \circ d_t = v$.

   Further, let $(inst', K) = i(\hat{v}) = i(s(inst, K))$. Then, according to the definition of $inst'$ in item 1, we have for all $p \in P_K$:

$$inst'_P(p) = (\hat{v}(p), p) = (a, p) = inst_P(p)$$

   and for all $t \in T_K$:

$$inst'_T(t) = (t, \hat{v} \circ d_t) = (t, v) = inst_T(t)$$

   Hence, we have $i(s(inst, K)) = (inst, K)$ for all $(inst, K) \in Inst(K)$, and thus $i \circ s = id_{Inst(K)}$.

   Now, let $\hat{v} \in Sol(K)$ be a solution for the realisation conditions of K, and $(inst, K) = i(\hat{v})$. Then we have $inst_P(p) = (\hat{v}, p)$ for all $p \in P_K$, and $inst_T(t) = (t, \hat{v} \circ d_t)$ for all $t \in T_K$.

   Further, let $\hat{v}' = s(inst, K) = s(i(\hat{v}))$. Then for $p \in P_K$ with $inst_P(p) = (a, p)$ we have $\hat{v}'(p) = a$, and since $inst_P(p) = (\hat{v}(p), p)$, we have $\hat{v}'(p) = \hat{v}(p)$.

   Moreover, for $t \in T_K$ with $inst_T(t) = (t, v)$ we have $\hat{v}' \circ d_t = v$ as stated in item 2. This means that we have $\hat{v}' \circ d_t = v = \hat{v} \circ d_t$, and since coproduct injections $(d_t)_{t \in T_K}$ are jointly epic, we have $\hat{v}' = \hat{v}$. Hence, we also have $s(i(\hat{v})) = \hat{v}$ for all $\hat{v} \in Sol(K)$, and thus $s \circ i = id_{Sol(K)}$, meaning that s and i are inverse isomorphisms, establishing a bijection of sets $Inst(K) \cong Sol(K)$. □

Bibliography

[AB09] Andrea Asperti and Nadia Busi. Mobile petri nets. *Mathematical Structures in Computer Science*, 19:1265–1278, 12 2009.

[AHS90] Jiří Adámek, Horst Herrlich, and George E. Strecker. *Astract and Concrete Categories*. Series in Pure And Applied Mathematics. John Wiley and Sons, 1990.

[AM75] Michael A. Arbib and Ernest G. Manes. *Arrows, Structures and Functors: The Categorical Imperative*. Academic Press, New York, 1975.

[Arb03] Stefan Arbanowski. *I-Centric Communications*. PhD thesis, Technische Universität Berlin, 2003.

[AvdMSPZ01] Stefan Arbanowski, Sven van der Meer, Stephan Steglich, and Radu Popescu-Zeletin. The Human Communication Space: Towards I-centric Communications. *Personal Ubiquitous Comput.*, 5(1):34–37, January 2001.

[BBB+09] Anthony Baxter, Jochen Bekmann, Daniel Berlin, Joe Gregorio, Soren Lassen, and Sam Thorogood. Google Wave Federation Protocol Over XMPP. http://wave-protocol.googlecode.com/hg/spec/federation/wavespec.html, July 2009.

[BCD+06] Manfred Broy, Michelle L. Crane, Juergen Dingel, Alan Hartman, Bernhard Rumpe, and Bran Selic. 2nd UML 2 semantics symposium: formal semantics for UML. In *Proceedings of the 2006 international conference on Models in software engineering*, MoDELS'06, pages 318–323, Berlin, Heidelberg, 2006. Springer-Verlag.

[BCE+07] Paolo Baldan, Andrea Corradini, Hartmut Ehrig, Reiko Heckel, and Barbara Knig. Bisimilarity and behaviour-preserving reconfigurations of open petri nets. In Till Mossakowski, Ugo Montanari, and Magne Haveraaen, editors, *Algebra and Coalgebra in Computer Science*, volume 4624 of *Lecture Notes in Computer Science*, pages 126–142. Springer Berlin Heidelberg, 2007.

[BCEH01] Paolo Baldan, Andrea Corradini, Hartmut Ehrig, and Reiko Heckel. Compositional Modeling of Reactive Systems Using Open Nets. In *CONCUR '01: Proceedings of the 12th International Conference on Concurrency Theory*, pages 502–518, London, UK, 2001. Springer.

[BD91] Bernard Berthomieu and Michael Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Transactions on Software Engineering*, 17, 1991.

[BDH92] Eike Best, Raymond Devillers, and JonG. Hall. The box calculus: A new causal algebra with multi-label communication. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 21–69. Springer Berlin Heidelberg, 1992.

[BEGG10] Benjamin Braatz, Hartmut Ehrig, Karsten Gabriel, and Ulrike Golas. Finitary $\mathcal{M}$-Adhesive Categories. In Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors, *Graph Transformations*, volume 6372 of *Lecture Notes in Computer Science*, pages 234–249. Springer Berlin Heidelberg, 2010.

[BEHM07] Enrico Biermann, Claudia Ermel, Frank Hermann, and Tony Modica. A Visual Editor for Reconfigurable Object Nets based on ECLIPSE Graphical Editor Framework. In Gabriel Juhás and Jörg Desel, editors, *Proc. 14th Workshop on Algorithms and Tools for Petri Nets (AWPN'07)*, 2007.

[BFH85] Paul Boehm, Harald-Reto Fonio, and Annegret Habel. Amalgamation of graph transformations with applications to synchronization. In Hartmut Ehrig, Christiane Floyd, Maurice Nivat, and James Thatcher, editors, *Mathematical Foundations of Software Development*, volume 185 of *Lecture Notes in Computer Science*, pages 267–283. Springer Berlin Heidelberg, 1985.

[BH07] MengChu Zhou Branislav Hruz. *Modeling and Control of Discrete-event Dynamic Systems: with Petri Nets and Other Tools*. Springer, 2007.

[BHH+97] Ruth Breu, Ursula Hinkel, Christoph Hofmann, Cornel Klein, Barbara Paech, Bernhard Rumpe, and Veronika Thurner. Towards a formalization of the Unified Modeling Language. In Mehmet Akşit and Satoshi Matsuoka, editors, *ECOOP'97 Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 344–366. Springer Berlin Heidelberg, 1997.

[BJR99] Grady Booch, Ivar Jacobson, and James Rumbaugh. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[BK84] Jan A. Bergstra and Jan W. Klop. Process algebra for synchronous communication. *Inform. and Control*, pages 109–137, 1984.

[BK85] Jan A. Bergstra and Jan W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(0):77 – 121, 1985.

[BRJ98] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Professional, Reading, Massachusetts etc., September 1998.

[BW90] Michael Barr and Charles Wells. *Category Theory for Computing Science*. Series in Computer Science. Prentice Hall International, London, 1990.

[CCCS92] Javier Campos, Giovanni Chiola, José-Manuel Colom, and Manuel Silva. Properties and performance bounds for timed marked graphs. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transac-*

tions on, 39(5):386–401, 1992.

[CDH99] Carla Capellmann, Heinz Dibold, and Uwe Herzog. Using high-level petri nets in the field of intelligent networks. In Jonathan Billington, Michel Diaz, and Grzegorz Rozenberg, editors, *Application of Petri Nets to Communication Networks*, volume 1605 of *Lecture Notes in Computer Science*, pages 1–36. Springer Berlin Heidelberg, 1999.

[Cm13] Co-meeting. http://www.co-meeting.com/, September 2013.

[DHP91] Christian Dimitrovici, Udo Hummert, and Laure Petrucci. Composition and Net Properties of Algebraic High-Level Nets. *Advances of Petri Nets, Lecture Notes in Computer Science*, 483, 1991.

[Dib88] Heinz Dibold. A method for the support of specifying the requirements of telecommunication systems. In *Digital Communications, 1988. Mapping New Applications onto New Technologies, 1988 International Zurich Seminar on*, pages 115–122, 1988.

[DMM89] Pierpaolo Degano, José Meseguer, and Ugo Montanari. Axiomatizing Net Computations and Processes. In *Proc. on Logic in Computer Science (LICS)*, pages 175–185. IEEE Computer Society, 1989.

[EEHP04] Hartmut Ehrig, Karsten Ehrig, Annegret Habel, and Karl-Heinz Pennemann. Constraints and application conditions: From graphs to high-level structures. In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors, *Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 287–303. Springer Berlin Heidelberg, 2004.

[EEPT06a] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental Theory for Typed Attributed Graphs and Graph Transformation based on Adhesive HLR Categories. *Fundam. Inf.*, 74(1):31–61, October 2006.

[EEPT06b] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in TCS. Springer, 2006.

[EG11] Hartmut Ehrig and Karsten Gabriel. Transformation of Algebraic High-Level Nets and Amalgamation of Processes with Applications to Communication Platforms. *Festschrift in Honour of Manfred Broy's 60th Birthday. International Journal of Software and Informatics*, 5(1-2,Part1), 2011.

[EGH10] Hartmut Ehrig, Ulrike Golas, and Frank Hermann. Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach. *Bulletin of the EATCS*, 102:111–121, 2010.

[EGH+12a] Hartmut Ehrig, Ulrike Golas, Annegret Habel, Leen Lambers, and Fernando Orejas. $\mathcal{M}$-Adhesive Transformation Systems with Nested Application Conditions. Part 1: Parallelism, Concurrency and Amalgamation.

Mathematical Structures in Computer Science, 2012. To appear.

[EGH+12b] Hartmut Ehrig, Ulrike Golas, Annegret Habel, Leen Lambers, and Fernando Orejas. $\mathcal{M}$-Adhesive Transformation Systems with Nested Application Conditions. Part 2: Embedding, Critical Pairs and Local Confluence. *Fundamenta Informaticae*, 118(1-2):35 – 63, 2012.

[EGLT11] Claudia Ermel, Jürgen Gall, Leen Lambers, and Gabriele Taentzer. Modeling with plausibility checking: inspecting favorable and critical signs for consistency between control flow and functional behavior. In *Proceedings of the 14th international conference on Fundamental approaches to software engineering: part of the joint European conferences on theory and practice of software*, FASE'11/ETAPS'11, pages 156–170, Berlin, Heidelberg, 2011. Springer-Verlag.

[EH86] Hartmut Ehrig and Annegret Habel. Graph grammars with application conditions. In *The Book of L*, pages 87–100. Springer Berlin Heidelberg, 1986.

[EHGP09] Hartmut Ehrig, Kathrin Hoffmann, Karsten Gabriel, and Julia Padberg. Composition and Independence of High-Level Net Processes. In *Proceedings of the First Workshop on Formal Methods for Wireless Systems (FMWS 2008)*, volume 242(2) of *Electronic Notes in Theoretical Computer Science*, pages 59–71, 2009.

[EHKPP91a] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, and Francesco Parisi-Presicce. From graph grammars to high level replacement systems. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 269–291. Springer Berlin Heidelberg, 1991.

[EHKPP91b] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, and Francesco Parisi-Presicce. From graph grammars to high level replacement systems. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 269–291. Springer Berlin Heidelberg, 1991.

[EHKPP91c] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, and Francesco Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Mathematical Structures in Computer Science*, 1:361–404, 11 1991.

[EHP+02] Hartmut Ehrig, Kathrin Hoffmann, Julia Padberg, Paolo Baldan, and Reiko Heckel. High-Level Net Processes. In *Formal and Natural Computing*, volume 2300 of *LNCS*, pages 191–219. Springer, 2002.

[EHP+07] Hartmut Ehrig, Kathrin Hoffmann, Julia Padberg, Ulrike Prange, and Claudia Ermel. Independence of net transformations and token firing in reconfigurable place/transition systems. In Jetty Kleijn and Alex

Yakovlev, editors, *Petri Nets and Other Models of Concurrency ICATPN 2007*, volume 4546 of *Lecture Notes in Computer Science*, pages 104–123. Springer Berlin Heidelberg, 2007.

[EHPP06] Hartmut Ehrig, Annegret Habel, Julia Padberg, and Ulrike Prange. Adhesive High-Level Replacement Systems: A New Categorical Framework for Graph Transformation. *Fundamenta Informaticae*, 74(1):1–29, 2006.

[EHPP07] Hartmut Ehrig, Kathrin Hoffmann, Ulrike Prange, and Julia Padberg. Formal foundation for the reconfiguration of nets. Technical report, Technische Universität Berlin, 2007.

[Ehr71] Hartmut Ehrig. *Übertragung universeller und spezieller Probleme in F-Morphismendarstellung.* PhD thesis, Technische Universität Berlin, 1971.

[Ehr79] Hartmut Ehrig. Introduction to the Algebraic Theory of Graph Grammars (A Survey). In Volker Claus, Hartmut Ehrig, and Grzegorz Rozenberg, editors, *Graph Grammars and Their Application to Computer Science and Biology, Lecture Notes in Computer Science,* No. 73, pages 1–69. Springer, 1979.

[Ehr05] Hartmut Ehrig. Behaviour and Instantiation of High-Level Petri Net Processes. *Fundamenta Informaticae*, 65(3):211–247, 2005.

[EM45] Samuel Eilenberg and Saunders MacLane. General Theory of Natural Equivalences. *Transactions of the American Mathematical Society*, 58(2):231–294, 1945.

[EM85] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1.* Springer, 1985.

[Eng91] Joost Engelfriet. Branching Processes of Petri Nets. *Acta Informatica*, 28(6):575–591, 1991.

[EPR92] Hartmut Ehrig, Julia Padberg, and Leila Ribeiro. Algebraic High-Level Nets: Petri Nets Revisited. In Hartmut Ehrig, editor, *COMPASS/ADT*, volume 785 of *Lecture Notes in Computer Science*, pages 188–206. Springer, 1992.

[EPS73] H. Ehrig, M. Pfender, and H. J. Schneider. Graph-grammars: An algebraic approach. In *Switching and Automata Theory, 1973. SWAT '08. IEEE Conference Record of 14th Annual Symposium on*, pages 167–180, 1973.

[ER76] Hartmut Ehrig and Barry K. Rosen. Commutativity of Independent Transformations on Complex Objects. Technical Report RC 6251, IBM Research, 1976.

[ER97] Hartmut Ehrig and Wolfgang Reisig. An Algebraic View on Petri Nets. *Bulletin of the EATCS*, 61:52–58, February 1997.

[Fac13] Facebook. http://www.facebook.com/, September 2013.

[Fis10] Winzent Fischer. Entwicklung einer Werkzeugumgebung für Algebraische High-Level Netze mit Anwendung auf ein Szenario zur Einsatzsteuerung bei der Berliner Feuerwehr. Diploma thesis, Technische Universität Berlin, 2010.

[Gab09] Karsten Gabriel. Composition and Transformation of High Level Petri Net-Processes. Diploma thesis, Technische Universität Berlin, 2009.

[Gab10] Karsten Gabriel. Algebraic High-Level Nets and Processes Applied to Communication Platforms. Technical Report 2010/14, Technische Universität Berlin, 2010.

[Gab11] Karsten Gabriel. Modelling of Google Wave Using Transformation of High-Level Petri Nets and their Processes. In G. E. Lasker and T. Soboll, editors, *Proceedings of the Symposium on Multiagent Systems, Robotics and Cybernetics, volume IV*. The International Institute for Advanced Studies in Systems Research and Cybernetics, 2011.

[Gab12a] Karsten Gabriel. Formal Modelling and Consistency of Apache Wave Platform Evolutions and Waves. In *Proceedings of the Symposium on Multiagent Systems and Cybernetics, volume V*, 2012.

[Gab12b] Karsten Gabriel. Process Evolution based on Transformation of Algebraic High-Level Nets with Applications to Communication Platforms. In *Proceedings of the 5th International Workshop on Petri Nets, Graph Transformation and other Concurrency Formalisms (PNGT 2012), ECEASST 51*, 2012.

[GBEG12] Karsten Gabriel, Benjamin Braatz, Hartmut Ehrig, and Ulrike Golas. Finitary $\mathcal{M}$-Adhesive Categories. *Mathematical Structures in Computer Science*, 2012. To appear.

[GE00] Maike Gajewsky and Claudia Ermel. Transition Invariants in Algebraic High-Level Nets, 2000.

[GE12a] Karsten Gabriel and Hartmut Ehrig. Modelling Evolution of Communication Platforms and Scenarios Based on Transformations of High-Level Nets and Processes. *Theor. Comput. Sci.*, 429:87–97, April 2012.

[GE12b] Karsten Gabriel and Hartmut Ehrig. Modelling of Communication Platforms Using Algebraic High-Level Nets and Their Processes. *Software Service and Application Engineering*, LNCS 7365:10–25, 2012.

[GEH10] Ulrike Golas, Hartmut Ehrig, and Annegret Habel. Multi-amalgamation in adhesive categories. In Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors, *Graph Transformations*, volume 6372 of *Lecture Notes in Computer Science*, pages 346–361. Springer Berlin Heidelberg, 2010.

[Gen87] Hartmann J. Genrich. Predicate/transition nets. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Petri Nets: Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer*

Science, pages 207–247. Springer Berlin Heidelberg, 1987.

[Gir71] Jean Giraud. Cohomologie non abélienne. Springer, Berlin, 1971.

[GL81] Hartmann J. Genrich and Kurt Lautenbach. System modelling with high-level Petri nets. *Theoretical Computer Science*, 13(1):109 – 135, 1981. Special Issue Semantics of Concurrent Computation.

[GLE12] Karsten Gabriel, Pascal Lingnau, and Claudia Ermel. Algebraic approach to timed petri nets. In *Proceedings of the 11th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2012), ECEASST 47*, 2012.

[GLMR05] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier H. Roux. Romo: A tool for analyzing time petri nets. In *Proc. CAV'05*, volume 3576 of *LNCS*, pages 418–423. Springer, 2005.

[GMMP91] Carlo Ghezzi, Dino Mandrioli, Sandro Morasca, and Mauro Pezzè. A unified high-level petri net formalism for time-critical systems. *IEEE Trans. Softw. Eng.*, 17:160–172, 1991.

[GN09] Joe Gregorio and Alex North. Google Wave Conversation Model. http://wave-protocol.googlecode.com/hg/spec/conversation/convspec.html, October 2009.

[Gog06] Joseph A. Goguen. Information Integration in Institutions. *Jon Barwise Memorial Volume*, 2006.

[Gol11] Ulrike Golas. *Analysis and Correctness of Algebraic Graph and Model Transformations*. PhD thesis, Technische Universität Berlin, 2011.

[GPP98] Martin Gogolla and Francesco Parisi-Presicce. State Diagrams in UML: A Formal Semantics using Graph Transformations - or Diagrams are nice, but graphs are worth their price. In *University of Munich*, pages 55–72, 1998.

[GR83] Ursula Goltz and Wolfgang Reisig. The Non-sequential Behavior of Petri Nets. *Information and Control*, 57(2/3):125–147, 1983.

[Gro57] Alexander Grothendieck. Sur quelques points d'algèbre homologique, I. *Tôhoku Math. J.*, 9(2):119–221, 1957.

[Gro71] Alexander Grothendieck. Catégories fibrées et descente, *Revêtements étales et groupe fondamental, Séminaire de Géométrie Algébrique du Bois-Marie 1960/61 (SGA 1)*,Exposé VI, 3rd ed. Institut des Hautes Études Scientifiques, Paris, 1963; reprint. *Lecture Notes in Mathematics*, vol. 224:145–194, 1971.

[GVH04] Szilvia Gyapay, Dniel Varró, and Reiko Heckel. Graph transformation with time. *Fundamenta Informaticae*, 58:1–22, 2004.

[HB08] Awatef Hicheur and Kamel Barkaoui. Modelling collaborative workflows using recursive ecatnets. In *Proceedings of the 8th international confer-*

ence on New technologies in distributed systems, NOTERE '08, pages 36:1–36:11, New York, NY, USA, 2008. ACM.

[HCE12] Frank Hermann, Andrea Corradini, and Hartmut Ehrig. Analysis of Permutation Equivalence in $\mathcal{M}$-adhesive Transformation Systems with Negative Application Conditions. *Mathematical Structures in Computer Science*, 2012. To appear.

[Hei10] Tobias Heindel. Hereditary pushouts reconsidered. In Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schrr, editors, *Graph Transformations*, volume 6372 of *Lecture Notes in Computer Science*, pages 250–265. Springer Berlin Heidelberg, 2010.

[HEM05] Kathrin Hoffmann, Hartmut Ehrig, and Till Mossakowski. High-level nets with nets and rules as tokens. In Gianfranco Ciardo and Philippe Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 268–288. Springer Berlin Heidelberg, 2005.

[Hen88] Matthew Hennessy. *Algebraic theory of processes*. MIT Press, Cambridge, MA, USA, 1988.

[HHT95] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26:287–313, 1995.

[HM03] Kathrin Hoffmann and Till Mossakowski. Algebraic higher-order nets: Graphs and petri nets as tokens. In Martin Wirsing, Dirk Pattinson, and Rolf Hennicker, editors, *Recent Trends in Algebraic Development Techniques*, volume 2755 of *Lecture Notes in Computer Science*, pages 253–267. Springer Berlin Heidelberg, 2003.

[Hoa85] Charles A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Hof06] Kathrin Hoffmann. *Formal Approach and Applications of Algebraic Higher-Order Nets*. PhD thesis, Technische Universität Berlin, 2006.

[HP99] Serge Haddad and Denis Poitrenaud. Theoretical aspects of recursive petri nets. In Susanna Donatelli and Jetty Kleijn, editors, *Application and Theory of Petri Nets 1999*, volume 1639 of *Lecture Notes in Computer Science*, pages 228–247. Springer Berlin Heidelberg, 1999.

[HP00] Serge Haddad and Denis Poitrenaud. Modelling and analyzing systems with recursive petri nets. In R. Boel and G. Stremersch, editors, *Discrete Event Systems*, volume 569 of *The Springer International Series in Engineering and Computer Science*, pages 449–458. Springer US, 2000.

[HP05] Annegret Habel and Karl-Heinz Pennemann. Nested constraints and application conditions for high-level structures. In Hans-Jörg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *Formal Methods in Software and Systems Modeling*,

volume 3393 of *Lecture Notes in Computer Science*, pages 293–308. Springer Berlin Heidelberg, 2005.

[Hum89] Udo Hummert. *Algebraische High-Level Netze.* PhD thesis, Technische Universtät Berlin, 1989.

[Jen91] Kurt Jensen. Coloured Petri Nets: A High-level Language for System Design and Analysis. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *LNCS*, pages 342–416. Springer, 1991.

[Jen97a] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, 1997.

[Jen97b] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods.* Monographs in Theoretical Computer Science. Springer-Verlag, 1997.

[Jen97c] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3, Practical Use.* Monographs in Theoretical Computer Science. Springer-Verlag, 1997.

[JK09] Kurt Jensen and Lars Michael Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems.* Springer, 2009.

[JLT09] Hung-Chin Jang, Yao-Nan Lien, and Tzu-Chieh Tsai. Rescue information system for earthquake disasters based on MANET emergency communication platform. In *Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly*, IWCMC '09, pages 623–627, New York, NY, USA, 2009. ACM.

[Joh02] Peter T. Johnstone. *Sketches of an Elephant - A Topos Theory Compendium.* Volume 1. Clarendon Press, Oxford, 2002.

[JR91] Kurt Jensen and Grzegorz Rozenberg, editors. *High-Level Petri Nets. Theory and Application.* Springer-Verlag, 1991.

[KEB94] Maciej Koutny, Javier Esparza, and Eike Best. Operational semantics for the petri box calculus. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR '94: Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science*, pages 210–225. Springer Berlin Heidelberg, 1994.

[KGKK02] Sabine Kuske, Martin Gogolla, Ralf Kollmann, and Hans-Jörg Kreowski. An Integrated Semantics for UML Class, Object and State Diagrams Based on Graph Transformation. In Michael Butler, Luigia Petre, and Kaisa Sere, editors, *Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, pages 11–28. Springer Berlin Heidelberg, 2002.

[KK99] Hans-Jörg Kreowski and Sabine Kuske. Graph transformation units with interleaving semantics. In *Formal Aspects of Computing*, pages 11–6,

1999.

[KR07] Michael Köhler and Heiko Rölke. Dynamic transition refinement. *Electron. Notes Theor. Comput. Sci.*, 175(2):119–134, June 2007.

[Krä87] Bernd Krämer. SEGRAS – a formal and semigraphical language combining Petri nets and abstract data types for the specification of distributed systems. In *Proceedings of the 9th international conference on Software Engineering*, ICSE '87, pages 116–125, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[Krä89] Bernd Krämer. *Concepts, Syntax and semantics of Segras, a specification language for distributed systems.* PhD thesis, Technische Universtät Berlin, 1989.

[Kus98] Sabine Kuske. More about control conditions for transformation units. In *Proc. Theory and Application of Graph Transformations, volume 1764 of Lecture Notes in Computer Science*, pages 323–337, 1998.

[LGM12] Pascal Lingnau, Karsten Gabriel, and Tony Modica. An algebraic approach to timed petri nets with applications to communication networks – extended version. Technical report, Technische Universität Berlin, 2012.

[Lil91] Johan Lilius. On the Compositionality and Analysis of Algebraic High-Level Nets. In *Research Report A16, Digital Systems Laboratory*, 1991.

[LNW03] Edward A. Lee, Stephen Neuendorffer, and Michael J. Wirthlin. Actor-oriented design of embedded hardware and software systems. *Journal of Circuits, Systems, and Computers*, 12:231–260, 2003.

[LS04] Stephen Lack and Pawel Sobociński. Adhesive Categories. In *Proc. FOSSACS 2004*, volume 2987 of *LNCS*, pages 273–288. Springer, 2004.

[LTB+12] Evanela Lapi, Nikolay Tcholtchev, Louay Bassbouss, Florian Marienfeld, and Ina Schieferdecker. Identification and Utilization of Components for a Linked Open Data Platform. *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, 0:112–115, 2012.

[Lur09] Jacob Lurie. *Higher Topoi.* Princeton University Press, 2009.

[MBE09] Tony Modica, Enrico Biermann, and Claudia Ermel. An eclipse framework for rapid development of rich-featured gef editors based on emf models. In Stefan Fischer, Erik Maehle, and Rdiger Reischuk, editors, *GI Jahrestagung*, volume 154 of *LNI*, pages 2972–2985. GI, 2009.

[MEE10] Maria Maximova, Hartmut Ehrig, and Claudia Ermel. Formal relationship between petri net and graph transformation systems based on functors between m-adhesive categories. *ECEASST*, 40, 2010.

[MEE12] Maria Maximova, Hartmut Ehrig, and Claudia Ermel. Transfer of local confluence and termination between petri net and graph transformation systems based on m-functors. *ECEASST*, 51, 2012.

[MGE+10] Tony Modica, Karsten Gabriel, Hartmut Ehrig, Kathrin Hoffmann, Shareef Shareef, Claudia Ermel, Ulrike Golas, Frank Hermann, and Enrico Biermann. Low- and High-Level Petri Nets with Individual Tokens. Technical Report 2009/13, Technische Universität Berlin, 2010.

[MGH10] Tony Modica, Karsten Gabriel, and Kathrin Hoffmann. Formalization of Petri Nets with Individual Tokens as Basis for DPO Net Transformations. *ECEASST*, 40, 2010.

[Mil73] Robin Milner. Processes: a mathematical model of computing agents. In *Logic Colloquium*, 1973.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[Mil99] Robin Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, 1999.

[MM90] José Meseguer and Ugo Montanari. Petri Nets Are Monoids. *Information and Computation*, 88(2):105–155, 1990.

[MMS97] José Meseguer, Ugo Montanari, and Vladimiro Sassone. On the Semantics of Place/Transition Petri Nets. *Mathematical Structures in Computer Science*, 7(4):359–397, 1997.

[Mod12] Tony Modica. *Formal Modeling, Simulation, and Validation of Communication Platforms*. PhD thesis, Technische Universität Berlin, 2012.

[MSM97] Naoto Makinae, Hikaru Suzuki, and Satoshi Minamoto. Communication Platform for Service Operations Systems in Advanced Intelligent Network. In *ICC (2)*, pages 872–877, 1997.

[Nov08] Peter Novák. Communication platform for open heterogeneous mass. Technical Report IfI-08-13, TU Clausthal, 2008.

[NPS95] Mogens Nielsen, Lutz Priese, and Vladimiro Sassone. Characterizing behavioural congruences for petri nets. In Insup Lee and ScottA. Smolka, editors, *CONCUR '95: Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 175–189. Springer Berlin Heidelberg, 1995.

[Pad92] Julia Padberg. Theory of High-Level-Replacement Systems and its Applications to Petri-Nets. Diploma thesis, Technische Universität Berlin, 1992.

[Pad96] Julia Padberg. *Abstract Petri Nets - Uniform Approach and Rule-Based Refinement*. PhD thesis, Technische Universität Berlin, 1996.

[Pas12] Marcus Pascal. Konzeption und implementierung eines werkzeugs zur rekonfiguration von algebraischen high-level-netzen. Diploma thesis, Technische Universität Berlin, 2012.

[PE01] Julia Padberg and Hartmut Ehrig. Parameterized Net Classes: A Uniform Approach to Petri Net Classes. In Hartmut Ehrig, Julia Padberg,

Gabriel Juhás, and Grzegorz Rozenberg, editors, *Unifying Petri Nets*, volume 2128 of *Lecture Notes in Computer Science*, pages 173–229. Springer Berlin Heidelberg, 2001.

[PEHP08] Ulrike Prange, Hartmut Ehrig, Kathrin Hoffmann, and Julia Padberg. Transformations in reconfigurable place/transition systems. In Pierpaolo Degano, Rocco Nicola, and Jos Meseguer, editors, *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 96–113. Springer Berlin Heidelberg, 2008.

[PER95] Julia Padberg, Hartmut Ehrig, and Leila Ribeiro. Algebraic High-Level Net Transformation Systems. *Mathematical Structures in Computer Science*, 80:217–259, 1995.

[Pet62] Carl A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Universität Bonn, 1962.

[Pra07] Ulrike Prange. Algebraic High-Level Nets as Weak Adhesive HLR Categories. *ECEASST*, 2:1–13, 2007.

[Pra08] Ulrike Prange. Towards Algebraic High-Level Systems as Weak Adhesive HLR Categories. In Hartmut Ehrig, Jochen Pfalzgraf, and Ulrike Prange, editors, *Proceedings of the ACCAT workshop at ETAPS 2007*, volume 203 / 6 of *Electronic Notes in Theoretical Computer Science*, pages 67–88, Amsterdam, 2008. Elsevier.

[Pro13] Google Wave Protocol. http://www.waveprotocol.org/, September 2013.

[PW98] Lutz Priese and Harro Wimmel. A uniform approach to true-concurrency and interleaving semantics for petri nets. *Theoretical Computer Science*, 206(1–2):219–256, 1998.

[Rei85] Wolfgang Reisig. *Petrinetze, Eine Einführung*. Springer Verlag, Berlin, 1985.

[Rei90] Wolfgang Reisig. Petri Nets and Algebraic Specifications. Technische Universität München, SFB-Bericht 342/1/90 B, March, 1990.

[Riz13] Rizzoma. https://rizzoma.com/, September 2013.

[Roz87] Grzegorz Rozenberg. Behaviour of Elementary Net Systems. In *Petri Nets: Central Models and Their Properties, Advances in Petri Nets*, volume 254 of *LNCS*, pages 60–94. Springer, 1987.

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol 1:* Foundations. World Scientific, Singapore, 1997.

[SB91] Christophe Sibertin-Blanc. Cooperative objects for the conceptual modelling of organizational information systems. In Van Assche, F., Moulin, B., and Rolland, C., editors, *The Object-Oriented Approach in Information Systems, IFIP TC8 Conference, 28-31 October 1991*, pages 275–296.

North-Holland, 1991.

[SB94] Christophe Sibertin-Blanc. Cooperative nets. In *Proceedings of 15th International Conference on the Application and Theory of Petri Nets, Lecture Notes in Computer Science 815*, pages 471–490. Springer-Verlag, 1994.

[SEM+12] Hanna Schölzel, Hartmut Ehrig, Maria Maximova, Karsten Gabriel, and Frank Hermann. Satisfaction, Restriction and Amalgamation of Constraints in the Framework of M-Adhesive Categories. In *ACCAT*, pages 83–104, 2012.

[Sie04] Frank Siegemund. A Context-Aware Communication Platform for Smart Objects. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 69–86. Springer Berlin Heidelberg, 2004.

[Sky13] Skype. http://www.skype.com/, September 2013.

[SMM+08] Krzysztof Szczypiorski, Igor Margasiński, Wojciech Mazurczyk, Krzysztof Cabaj, and PawełRadziszewski. Trustmas: Trusted communication platform for multi-agent systems. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems: OTM 2008*, volume 5332 of *Lecture Notes in Computer Science*, pages 1019–1035. Springer Berlin Heidelberg, 2008.

[Ste03] Stephan Steglich. *I-Centric User Interaction*. PhD thesis, Technische Universität Berlin, 2003.

[SV01] Alberto Sangiovanni-Vincentelli. Platform-based design. In *IEEE Design e Test, 18(6):23 33, Novembro*, 2001.

[Tae04] Gabriele Taentzer. Agg: Agraph transformation environment for modeling and validation of software. In John L. Pfaltz, Manfred Nagl, and Boris Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance*, volume 3062 of *Lecture Notes in Computer Science*, pages 446–453. Springer Berlin Heidelberg, 2004.

[Tav02] Ahmad R. K. Tavakoli. Transformation of Open and Algebraic High-Level Peti Net Classes. Technical Report 2002-24, Technische Universtät Berlin, 2002.

[TBG91] Andrzej Tarlecki, Rod M. Burstall, and Joseph A. Goguen. Some fundamental algebraic tools for the semantics of computation: Part 3. indexed categories. *Theoretical Computer Science*, 91(2):239 – 264, 1991.

[Twi13] Twitter. http://www.twitter.com/, September 2013.

[UP08] Conny Ullrich and Julia Padberg. Reconfigurable open algebraic high-level systems. *ECEASST*, 14, 2008.

[Urb03] Milan Urbasek. Net Transformations for Petri Net Technology. *Bulletin of EATCS, Formal Specification Column*, 80:77–94, June 2003.

[Val78] Rüdiger Valk. Self-modifying nets, a natural extension of petri nets. In Giorgio Ausiello and Corrado Böhm, editors, *Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 464–476. Springer Berlin Heidelberg, 1978.

[Val98] Rüdiger Valk. Petri nets as token objects. In Jörg Desel and Manuel Silva, editors, *Application and Theory of Petri Nets 1998*, volume 1420 of *Lecture Notes in Computer Science*, pages 1–24. Springer Berlin Heidelberg, 1998.

[Val01] Rüdiger Valk. Concurrency in communicating object petri nets. In Gul A. Agha, Fiorella Cindio, and Grzegorz Rozenberg, editors, *Concurrent Object-Oriented Programming and Petri Nets*, volume 2001 of *Lecture Notes in Computer Science*, pages 164–195. Springer Berlin Heidelberg, 2001.

[Wal13] Google Walkaround. http://wavereactor.appspot.com/, September 2013.

[Wav13a] Apache Wave. http://incubator.apache.org/wave/, September 2013.

[Wav13b] Wave in a box. http://waveinabox.net/, September 2013.

[Wik13a] Wikipedia. http://www.wikipedia.org/, September 2013.

[Wik13b] WikiWikiWeb. http://c2.com/cgi/wiki?WikiWikiWeb/, September 2013.

[Yon10] Tsvetelina Yonova. Formal Description and Analysis of Distributed Online Collaboration Platforms. Bachelor thesis, Technische Universität Berlin, 2010.

Index

T

U

V

W