

David S. Karcher

Event Structures with Higher-Order Dynamics



David S. Karcher
Event Structures with Higher-Order Dynamics

Die Schriftenreihe *Foundations of Computing* der Technischen Universität Berlin
wird herausgegeben von:

Prof. Dr. Rolf Niedermeier,

Prof. Dr. Uwe Nestmann,

Prof. Dr. Stephan Kreutzer

David S. Karcher

Event Structures with Higher-Order Dynamics

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.dnb.de> abrufbar.

Universitätsverlag der TU Berlin, 2019

<http://verlag.tu-berlin.de>

Fasanenstr. 88, 10623 Berlin

Tel.: +49 (0)30 314 76131 / Fax: -76133

hrefmailto:publikationen@ub.tu-berlin.de

Zugl.: Berlin, Techn. Univ., Diss., 2017

Gutachter: Prof. Dr. -Ing. Uwe Nestmann

Gutachter: Prof. Thomas T. Hildebrandt, Ph. D. (IT University of Copenhagen, Dänemark)

Gutachterin: Prof. Dr. Sabine Glesner

Die Arbeit wurde am 21. Dezember 2017 an der Fakultät IV unter Vorsitz von Prof. Dr. Stephan Kreutzer erfolgreich verteidigt.

Diese Veröffentlichung – ausgenommen Umschlagfoto – ist unter der CC-Lizenz CC-BY lizenziert.

Lizenzvertrag: Creative Commons Namensnennung 4.0

<http://creativecommons.org/licenses/by/4.0>

Druck: docupoint GmbH

Satz/Layout: David S. Karcher

Umschlagfoto:

https://commons.wikimedia.org/wiki/File:Herakles_Kerberos_Louvre_F204.jpg | CC0

<https://creativecommons.org/publicdomain/zero/1.0/>

ISBN 978-3-7983-2995-9 (print)

ISBN 978-3-7983-2996-6 (online)

ISSN 2199-5249 (print)

ISSN 2199-5257 (online)

Zugleich online veröffentlicht auf dem institutionellen Repositorium der Technischen Universität Berlin:

DOI 10.14279/depositonce-6989

<http://dx.doi.org/10.14279/depositonce-6989>

“Even if there is only one possible unified theory, it is just a set of rules and equations. What is it that breathes fire into the equations and makes a universe for them to describe? The usual approach of science of constructing a mathematical model cannot answer the questions of why there should be a universe for the model to describe. Why does the universe go to all the bother of existing?”

Stephen Hawking, *A Brief History of Time*

Zusammenfassung

Ereignisstrukturen wurden 1979 [18] als formales Modell eingeführt um die Theorie der Petri Netze mit der Theorie der Verbände zu verknüpfen. Ursprünglich bestanden sie aus atomaren nicht wiederholbaren Ereignissen, einer binären kausalen Abhängigkeitsrelation und einer binären Konfliktrelation auf den Ereignissen. Lange Zeit wurden verschiedene Erweiterungen des Ursprungsformalismus genutzt, um Semantiken für andere Strukturen zu definieren (z. B. Klassen von Petri Netzen und Prozesskalkülen).

In dieser Arbeit werden Ereignisstrukturen (ESs) als deklarativer Modellierungsfomalismus betrachtet und nicht, um Semantiken für andere Strukturen zu definieren.

Um hochdynamische Prozesse (also Prozesse, in denen Ereignisse die Abhängigkeiten anderer Ereignisse verändern können) aus der realen Welt in einem präzisen, aber kleinen Modell abbilden zu können, muss die Dynamik inhärenter Bestandteil des Modellierungsfomalismus sein.

Um dieses leisten zu können, wurden die Higher-Order Dynamic-Causality ESs (HDESSs) eingeführt. Sie bestehen aus einer endlichen Menge atomarer und nicht wiederholbarer Ereignisse, einer kausalen Abhängigkeitsrelation auf diesen Ereignissen und einem regelbasierten Formalismus, mit dem Ereignisse die Abhängigkeiten anderer Ereignisse (und auch eben diese Regeln) verändern können.

Der Formalismus wird bezüglich seiner Ausdrucksstärke im Vergleich zu Transitionssystemen, zu einigen Unterklassen von HDESSs und zu Dynamic Condition Response Graphs (DCR-Graphs) betrachtet.

Des Weiteren wird ein Web-Tool vorgestellt, das es ermöglicht, HDESSs zu definieren und zu erforschen, indem man Ereignisse ausführt, das zugehörige Transitionssystem erzeugt oder vordefinierte Transformationen anwendet.

Abstract

Event Structures were introduced in 1979 [18] as a formal model to connect the theory of Petri nets and domain theory. Originally they consisted of atomic non-repeatable events, a binary causal dependency relation, and a binary conflict relation between those events. For a long time various extensions of the original formalism were used to define semantics for other structures such as classes of Petri nets and process calculi.

In this thesis the Event Structures (ESs) are considered solely as a declarative modelling tool than as a formalism to define semantics for other structures.

In order to model highly dynamic real-world processes (i.e. processes in which occurrences of events may change the dependencies of other events) with a concise model the dynamics must be an inherent part of the modelling formalism.

Therefore, the Higher-Order Dynamic-Causality ESs (HDESs) were introduced. They consist of a finite set of atomic non-repeatable events, a causal dependency relation between these events, and a rule-based formalism for events to change the dependencies and the existing rules.

This formalism is studied with the respect to its expressive power in comparison to transition systems, some subclasses of HDESs, and to Dynamic Condition Response Graphs (DCR-Graphs).

A web tool was created in which HDESs can be defined and explored by executing events, creating the corresponding transition system, or even applying some predefined transformations.

Acknowledgements

First of all I would like to express my sincere gratitude to my supervisor Uwe for giving me the opportunity to do my Ph.D. in his group and for all his advice and support in the whole time, and for sharing my sense of humour.

My sincere thanks also goes to my referee Thomas for hosting me in Copenhagen and for very interesting discussion on DCR-Graphs and their connection to HDESSs.

Besides Uwe and Thomas, I would like to thank Stephan for making time to chairing my defence and for his insightful comments on my work.

I am also deeply grateful to the research training group SOAMED for funding my Ph.D. and giving me many opportunities for fruitful discussion with other supervisors and Ph.D. students.

I thank all my colleagues, both current and former. It was always fun working with you guys. A particular thanks goes to Youssef for sharing his research area with the new guy and introducing me to the world of Event Structures, also to Ben for all the time he invested into understanding my ideas, spotting inaccuracies and errors, creating the web tool, and for his help in proof-reading this thesis.

I thank Jonas and Julian for their help in improving my English.

I also thank all friends who only saw little of me during the past few years while I was stuck in front of my computer writing this. I hope this will change now.

Finally, I thank my parents Dorothea and Reinhard for supporting me throughout my whole life and my brother Michael for being such a reliable source of information on any topic and for his help in proof-reading this thesis.

Last but not least, I would like to thank my wife Sophia for her patience and support and my son Nikolai for being such a sunny boy, who can always lift my mood. I love you both.

Contents

1	Introduction and Motivation	1
1.1	Introduction	1
1.1.1	Declarative Modelling	3
1.1.2	Dynamic Causality-Based Formalisms	4
1.1.3	Research Question	8
1.2	Overview	8
1.3	Own Publications	10
2	Technical Preliminaries	13
2.1	Prime Event Structures	13
2.2	Event Structures for Resolvable Conflicts	14
2.3	Dynamic-Causality Event Structures	16
2.4	Dynamic Condition Response Graphs	19
3	Higher-Order Dynamics	25
3.1	Higher-Order Dynamic-Causality ESs	25
3.2	The Web Tool	31
3.2.1	The Syntax of the Web Tool	31
3.2.2	The GUI of the Web Tool	32
3.3	Concepts for Working with HDESSs	33
3.3.1	Clean-Up	33
3.3.2	Synchronisation Pattern	36
3.4	Alternative Approaches	42
3.4.1	Alternative Rule Update Strategy	42
3.4.2	Event Transition Systems	45
3.4.3	Rule Set Reduction	61
3.5	On the Expressive Power of HDESSs	64
3.5.1	Encoding DCESSs in HDESSs	65
3.5.2	Orthogonal Extensions	67
3.5.3	From Configuration Structures to HDESSs	68
3.5.4	Expressive Power with Respect to the Level of Dynamics	74

4	Encoding of DCR-Graphs	81
4.1	Concurrency Semantics	81
4.2	Encoding of DCR-Graphs into HDESs	86
4.2.1	Explanation and Definition of the Encoding	86
4.2.2	Correctness of the Encoding	93
5	Application	109
5.1	Case Study	109
5.1.1	The Surgery	109
5.1.2	Some Modelling	110
5.1.3	Conclusion	112
5.2	Requirements for Further Applicability	113
5.2.1	Syntactic Sugar	113
5.2.2	Proper Extensions	115
5.2.3	Closing Remarks	116
6	Conclusion	119
6.1	Main Contributions	119
6.2	Open Problems and Future Work	120
	Bibliography	123

List of Figures

1	Example PES of an agricultural process	2
2	The example of Figure 1 enhanced by exceptional behaviour . .	5
3	Transition introduction	6
4	Synchronisation in Petri nets	6
5	A square in a finite state machine	8
6	Transition graphs of example RCESSs	15
7	An order-sensitive DCES	17
8	Encoding of conflicting events in CRESSs into DCR-Graphs . . .	23
9	A HDES example with second-order dynamics	26
10	Sketch for Definitions 3.32 and 3.33	54
11	Sketch for the proof of Theorem 3.39	61
12	Example HDES for Subsection 3.5.2	68
13	A transition-equivalent HDES H_{ρ_H} for ρ_H (as in Definition 3.62)	75
14	The HDESs H_5 and H'_5 (as in Definitions 3.68 and 3.69)	76
15	The HDESs G_5 and G'_5 (as in Definitions 3.71 and 3.72)	78
16	The HDESs D_2 (as in Definition 3.74)	80
17	Sketch for the proof of Lemma 4.26	96
18	The rule sets for the proof of Lemma 4.26	97
19	Example with set-based dynamics	111
20	Example with set-based and second-order dynamics	112
21	Example for bundling of dynamics	115

List of Examples in the Web Tool

3.1	HDES example from Figure 9	31
3.2	HDES example for the synchronisation pattern	37
3.3	HDES example for the extended synchronisation pattern . . .	40
3.4	A HDES with a redundant rule that never gets cleaned up . . .	66
3.5	A transition-equivalent HDES encoding for the DCES in Figure 7	66
5.1	A list of all dependencies and rules of the ES in Figure 20 . . .	113

List of Algorithms

1	The HDES rule set update algorithm	29
2	An alternative HDES rule set update algorithm	44

1 Introduction and Motivation

1.1 Introduction

Formal models are nowadays needed in almost every branch of computer science. There are a lots of formal models tailored for the different needs and features of application domains. In concurrency theory a variety of models exist and some focus on the possibly observed traces, some on the branching behaviour and other aspects. Causality-based models like Petri nets or Event Structures focus on the causal dependencies between the modelled objects.

Causality-based models are in particular useful for areas where it is interesting and important to track which events were responsible for which action. The causality-based approach follows the so-called declarative modelling paradigm. Therefore the modelling is based on constraints like ordering, conflicts, causal and other dependencies. For legal reasons it is crucial to know *why* an action was performed in a hospital, it is therefore suitable to use causality-based models in this domain.

We conducted a case study in cooperation with the German Heart Center Berlin (DHZB) in which two Bachelor students modelled the treatment of a patient suffering from coronary heart disease and a mitral valve insufficiency [22, 13]. They used two different causality-based formalisms which also supported some dynamic adaptations of the processes. The dynamicity was strongly needed since the patient treatment process is not nearly static but is often adapted due to new diagnoses. The case study exhibited that causality-based modelling is well suited for medical workflows.

In this thesis we first introduce various kinds of event structures and related causality-based approaches.

Event Structures (ESs) were introduced by Nielsen, Plotkin, and Winskel [18] in 1979 to connect the theory of Petri nets and domain theory. In most cases these consist of atomic and non-repeatable events and relations between those events, mostly a causality relation (in the meaning of enabling) and a conflict relation. The semantics of an ES itself is usually provided by the sets

of traces compatible with the constraints and some also have transition-based semantics. The relations differ from structure to structure and they might be binary on the set of events or relating sets of events with events. The conflict relation might be a symmetric one. Depending on these choices for those relations and possibly additional constraints, one gets different expressive power for the resulting structures.

We first take a closer look at the earliest and one of the simplest are the Event Structures [18]. They have a partial order as causality relation and a symmetric, irreflexive conflict relation. The authors showed that these event structures can define semantics for causal nets [18]. In his PhD thesis in 1980 Winskel studied the ESs in more depth and called them Prime Event Structures (PESs) since they correspond to prime algebraic domains [25].

Figure 1 shows a small example Prime Event Structure without conflicting events: Before planting crop, it is needed to plough and water the field. After the planting one might harvest. The events are denoted as dots and dependencies as solid arrows. We omitted reflexive and transitive arrows for simplicity.

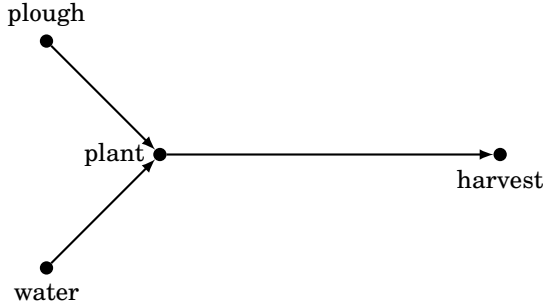


Figure 1: Example for a PES modelling an agricultural process.

Later on other types of event structures were used to define semantics for process calculi [6], [17].

In 2004 Plotkin and v. Glabbeek [12] introduced the general Event Structures for Resolvable Conflicts (RCES). They can provide semantics for general Petri nets and are at least as expressive as all ESs defined up to then.

In 2015 Arbach, Karcher, Peters and Nestmann [3] developed the Dynamic-Causality ESs (DCESs) which represents a generalisation of the ESs in [18]. The purpose of DCES is to be used to model highly dynamic processes (i.e. processes

in which events may change the causal dependencies of other events). Such processes can be found in the medical sector (e.g. the above-mentioned treatment process of a patient in a hospital). Surprisingly DCEs and RCEs are incomparable with respect to expressive power. In 2015 Karcher and Nestmann [16] presented as well a generalisation of DCEs named Higher-Order Dynamic-Causality ESs (HDESs). The HDESs are not only more expressive than DCEs but also more expressive than RCEs.

In this thesis we analyse the HDESs, on their own and in comparison with other structures.

1.1.1 Declarative Modelling

This subsection follows [21].

The essence of formal modelling is to create abstractions of real-world or virtual entities. With these abstractions the modeller can focus on those aspects he or she is interested in. Furthermore, formal models allow for formal analysis to be applied (e.g. model checking, reachability checking).

Similar to the area of programming languages, there are different paradigms one can follow while modelling. *Procedural* modelling focusses on *how* things are executed; this is often times achieved with an explicit control flow. In the *declarative* approach, the modelling is based on the *what*; this is typically achieved by imposing constraints (e.g. conflicts, conditioning, ordering).

One commonly known example of a declarative programming language is the general-purpose logic programming language Prolog. In contrast to traditional procedural programming, the logic of a computation is expressed without any control flow. In Prolog, one defines facts and rules, a computation is then initiated by a query. This declarative approach is also very interesting for process modelling. But first we consider a traditional approach.

In imperative workflow modelling (e.g. BPMN [23]), the focus is on *how* a process should be executed. *“This often leads to rigid and overspecified process descriptions, that fail to capture why the activities must be done in the given order.”* ([21] page 1). In the declarative approach a process is modelled by its constraints, which means in *what* particular order events can or must be executed. As long as events do not violate these constraints, they might occur in any order. With this more flexibility is gained in the execution and the model more accurately reflects the reasons for the specific observed orderings of events.

Retrospectively, the declarative approach often enables us to reason about *why* actions (or events) that took place since we know the constraints (e.g. causal dependencies, conflicts).

In contrast to the imperative approaches, it can get complicated to create a step-by-step process description since the process might be underspecified when modelling declaratively.

1.1.2 Dynamic Causality-Based Formalisms

As result of the case studies, we are looking for dynamic causality-based formalisms. Since Event Structures (ESs) are causality-based, we investigated their extensions. For a long time the use of Event Structure had been limited to define semantics for other structures like Petri nets, process calculi, etc. (e.g. RCEs can give semantics to general Petri nets). However, when ES-based formalism were used for modelling [14, 3], there was a need for more dynamicity than any ESs could offer: For example, the above-mentioned RCEs lacked the capability of generally adding or dropping causal predecessors for other events.

First, we introduce Higher-order Dynamic-Causality ESs (HDESs). Second, we mention Dynamic Condition Response Graphs as a causality-based formalism and later in this thesis we will discuss them in depth. Third and fourth, we briefly present Petri nets and Higher Dimensional Automata – two further causality-based formalisms – and explain why those are not suited for our purpose.

Higher-order Dynamic-Causality ESs

We introduced Higher-order Dynamic-Causality ESs (HDESs) in [16] and intensively study them in this thesis as a causality-based modelling formalism for dynamic processes. HDESs consist of non-repeatable atomic events, a causality relation on those events (defining causal predecessors), and a rule set, which allows for events to alter the dependencies and also the rule set itself. We created a web tool, in which HDESs can be defined, visualised, and executed at <http://hdes.mtv.tu-berlin.de/>.

The HDES model in Figure 2 represents an exemplary agricultural process of planting and harvesting (this is an enhanced version of the example in Figure 1): In order to plant, one needs to plough and water the field. After planting one might harvest. This is only the regular behaviour (and exactly the same as in Figure 1). We extend the model with the possibility of rain; now rain would delete the dependency of watering for planting. A further extension is a potential

pest infestation; then, one is forced to do a pest control before one can harvest. The fact that an event can insert or delete a dependency is represented by an arrow between the event and the dependency. A filled arrowhead indicates an insertion and an empty arrowhead with a reversed orientation a deletion. Initially absent dependencies which may be added dynamically are denoted dashed. In this example, we modelled the exceptional behaviour (triggered by rain or pest infestation) as deletion and insertion of dependencies. We name such a insertion (resp. deletion) of a dependency a first-order rule. An insertion or removal of a first-order rule is called second-order rule, etc. for higher levels of dynamicity. In general HDES allow for arbitrary high order of dynamics and for sets of events to add or delete a rule (static i.e. causal dependencies or dynamic).

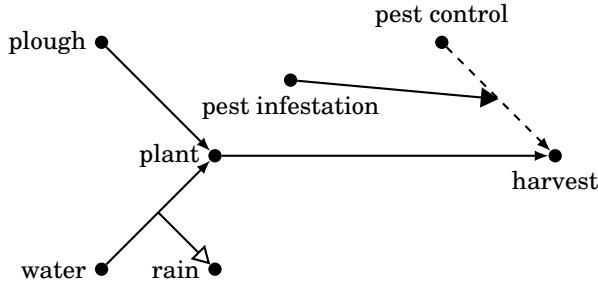


Figure 2: The example of Figure 1 enhanced by exceptional behaviour.

Dynamic Condition Response Graphs

Dynamic Condition Response Graphs (DCR-Graphs) were introduced as an event-based modelling formalism designed for dynamic processes [14]. In contrast to HDESs the dynamicity is obtained by dynamic exclusion and inclusion of events; this means that excluded events are not needed for acceptance or enabling of other events, unless the said events are included again. Another difference to the HDESs approach is the repeatability of events. There are two web tools for DCR-Graphs: a scientific one at <http://tiger.itu.dk> and a commercial one at www.dcrgraphs.net. A formal definition can be found in Section 2.4.

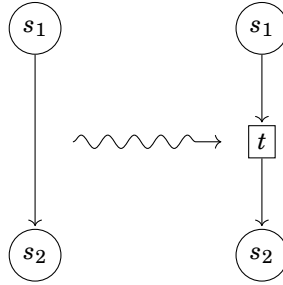


Figure 3: Transition introduction. Between the places s_1 and s_2 a transition t gets introduced.

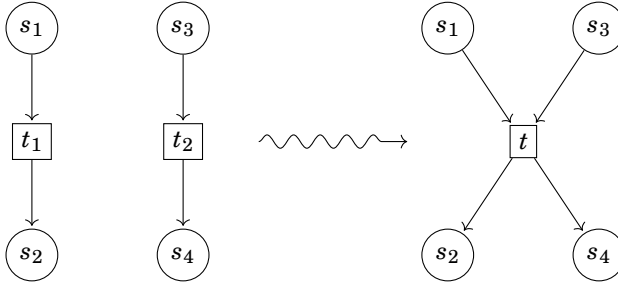


Figure 4: Synchronisation in Petri nets. On the left hand side are two independent transitions t_1 and t_2 . On the right hand side we merged them into the synchronised transition t .

Petri nets

Petri nets are a well-known causality-based formalism. Their origin lies in the PhD thesis of Petri, “Kommunikation mit Automaten”, in 1962 [19].

They are based on the idea of having multiple finite automata in parallel. For steps in the respective automata, now called places, transition in between the states get introduced (as in Figure 3). In order to establish communication (i.e. synchronisation) between automata, different transitions are merged into a single one (as in Figure 4).

Thus, formally a Petri net is a bipartite directed graph (partitioned into places and transitions). A state of the net is a marking, which refers to a state in each

automaton, indicated with tokens on these places. In general the number of tokens per place is not restricted. To perform a transition in a net, there must be at least one token on each pre-place (i.e. places with an edge *to* this transition). As a result of the transition one token of each pre-place gets consumed (i.e. removed) and on each post-place (i.e. places with an edge *from* this transition) a token is added.

Causal dependencies of two actions can be encoded quite naturally in Petri nets by encoding them as transitions with one place in-between. Thus, the dependent action can not take place until the causal predecessor produced a token on its post-place that is also a pre-place of the dependent transition. However, in a huge Petri net it becomes complicated to derive all dependencies. Since in Petri nets transition cycles occur quite naturally, it is not always easy to precisely track causality. Moreover, if there are several interfering cycles, it becomes even more complicated to do so.

Even if we know all the dependencies it is still very complicated to manipulate them, since a transition has to shut down the part of the net with the old behaviour and activate the part with the new behaviour. With this approach, we create an extremely huge static net. Yet we aim for a concise model with built-in dynamics. Thus, Petri nets are generally not well suited for modelling processes in which the causal dependencies evolve during run time.

Higher Dimensional Automata

A Higher Dimensional Automaton (HDA) is a traditional automaton enriched with a notion of concurrency (respectively independence). The HDA were first introduced by Pratt in [20]. Here we follow van Glabbeek [9].

Traditionally, finite state machines do not have any concept of concurrency. Thus, the possible behaviour of the automaton in Figure 5 would be reading the sequence ab or the sequence ba .

The key feature of HDAs is that such situations as in Figure 5 can be defined as concurrent. That means a and b can occur concurrently or sequentially. Such a two-dimensional square can easily be generalised to arbitrary n -dimensional hyper cubes. The resulting formalism is at least as expressive as Petri nets and automata (cf. [9]).

In order to achieve our goal of dynamising a causality-based formalism in order to create concise models of dynamic processes, HDAs are not suited. Three-dimensional cubes (if there are three concurrent events in a state) already are

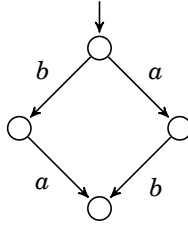


Figure 5: A square (responsively a two-dimensional hyper cube) in a finite state machine.

hard to visualise. This approach becomes even more complicated if there are even more than three concurrent events.

1.1.3 Research Question

Since we are interested in a declarative dynamic causality-based modelling approach, we conducted the above-mentioned case study (more details in Section 5.1) using DCEs. We further theoretically explored the expressive power of DCEs [2]. Based on the incomparability of DCEs and RCEs and also the needs of the case study the following research questions arise:

- Is it possible to generalise DCEs
 - to be at least as expressive as RCEs,
 - and to be capable of expressing the features needed in the case study?
- What is the expressive power of the generalised structure in comparison to itself and to other formal models?

1.2 Overview

In Chapter 2 we present three different event structure formalisms and related causality-based approaches: In Section 2.1 we present a relaxed Prime Event Structures (rPESs), a relaxation of Prime Event Structures (PESs) [24], as the fundamental event structure formalism. In Section 2.2 we present Event Structures

for Resolvable Conflicts (RCESs) [12] as very general event structures, to compare to with respect to expressive power. In Section 2.3 we present Dynamic-Causality Event Structures (DCESS) [2], a dynamisation of the rPESs, and summarise the results on its expressive power. In Section 2.4 we present a dynamic causality-based declarative modelling approach, called the Dynamic Condition Response Graphs (DCR-Graphs).

In Chapter 3 we introduce and analyse Higher-order Dynamic-Causality ESs (HDESs). In Section 3.1 we start with an revised version of the HDES definition from [16]. In Section 3.2 we explain the web tool for working with HDESs. In Section 3.3 we define two concepts that are useful for dealing with HDESs both in designing and proving work: First, the so-called clean-up in Subsection 3.3.1 which allows for deletion of unnecessary rules in a specific state. Second, the synchronisation pattern in Subsection 3.3.2 – a set of rules that enforces an effect if and only if some events occur in the same transition.

In Section 3.4 we provide a different perspectives onto HDESs. First we define an alternative rule update strategy in Subsection 3.4.1. Second in Subsection 3.4.2 we give an alternative representation, namely transition systems. We show that a specific class of transition systems is exactly as expressive as HDESs, independent of the rule update strategy. This implies that both rule update strategies result in the same expressive power. Finally in Subsection 3.4.3 we present a state-independent approach of detecting unnecessary rules (without taking the whole rule set into consideration) and discuss the connection to the above defined clean-up.

The last Section 3.5 of this chapter contains various results with respect to the expressive power of HDESs: In Subsection 3.5.1 we show that the HDES indeed generalise the DCESSs, this is not fully obvious since the semantics differ. In Subsection 3.5.2 we show that both dimensions of the generalisation from DCESSs to HDESs are orthogonal, that means neither set-based dynamics can be generally expressed with higher-order singleton rules, nor the other direction. In Subsection 3.5.3 we recap the result of [16], which relates the expressive power of configuration structures to the one of HDESs. In Subsection 3.5.4 we present some basic results showing that some orders of dynamicity (maximal depth of a rule) yield different expressive power and formulate a conjecture that this holds for any two different levels of dynamicity.

In Chapter 4 we present an encoding for basic DCR-Graphs into HDESs. Therefore, we present the Section 4.1 true-concurrency semantics for DCR-Graphs as in [8]. Next in Subsection 4.2.1 we define the encoding of basic DCR-Graphs into HDESs and in Subsection 4.2.2 prove that this encoding preserves the behaviour.

In Chapter 5 we take a look at future applications of the HDES formalism. To this end, in Section 5.1 we present a case study modelling the treatment process of a patient in a hospital, using the HDES formalism. Based on the results of the case study and other observations, we further discuss in Section 5.2 which features are still needed to be able to apply HDESs in real-world use cases.

We conclude in Chapter 6. In Section 6.1 we summarise the contribution of this thesis before exploring some interesting avenues for future research in Section 6.2.

1.3 Own Publications

In the following, we list our own publications, shortly summarise them and point out for which parts of the publications the author was responsible.

Dynamic-Causality in Event Structures

In [3] and [4] we first considered the insertion and deletion of dependencies separately. Then we analysed Shrinking Causality ES (SES) with only deletion of dependencies, and Growing Causality ES (GES) with only insertion of dependencies, on their own with respect to their expressive power in comparison to other ESs. Thereafter, we combined both approaches and defined and analysed the Dynamic-Causality ES (DCES). My main focus was on the comparisons of the GES to the other ESs, on the comparison of DCES to ES for resolvable conflicts and on the transition definition of the DCES.

Dynamic-Causality in Event Structures (Journal)

In [2] we combined both previous publication in one journal article. Additionally, we came up with a simpler transition definition for SES, GES, and DCES, by adapting the causal dependencies in the transitions if they were dropped or inserted. This idea is based on the definition for Higher-Order Dynamic-Causality ESs (DCESs) in [16]. We also inserted a section on the semantics of ESs since we were using many different ESs and wanted to point out their relationship more clearly. This was the focus of Kirstin Peters.

Higher-Order Dynamics in Event Structures

In [16] we presented the Higher-Order Dynamic-Causality ESs (HDESs) and showed that they are at least as expressive as some configuration structures (a reworked version can be found in Subsection 3.5.3). Sections 2 to 4 were written by myself, whereas the introduction and conclusion were written jointly with Uwe Nestmann.

2 Technical Preliminaries

In this chapter we present three kinds of event structures, which we will later use to build on and to compare to. We also present the related causality-based approach Dynamic Condition Response Graphs. We first introduce relaxed Prime Event Structures (rPESs) (as a relaxation of [24]) as a foundation for the later event structures (ESs). Second we will present the very general Event Structure for Resolvable Conflicts (RCEs) [12]. Finally we present our dynamisation of the rPESs, the Dynamic Causality Event Structures (DCEs) [2], and state an incomparability result between DCEs and RCEs. This chapter is based on our work in [2].

2.1 Prime Event Structures

Relaxed Prime Event Structures (rPESs) consist of a set of events and two relations. One relation describes conflicts and the other causal dependencies. To cover the intuition that events causally depending on an infinite number of other events can never occur, it is required for rPESs to satisfy the axiom of finite causes.

Additionally for the original Prime Event Structures (PES) [24], the enabling relation is assumed to be a partial order, that means it is transitive, reflexive, and anti-symmetric. Furthermore, the concept of conflict heredity is required; that is an event conflicting with a second event is required to conflict with all causal successors of that second event.

If we allow causal dependencies to be added or dropped, it is hard to maintain the conflict heredity as well as transitivity and reflexivity of the enabling relation. Therefore, we drop the partial order property and the axiom of conflict heredity in our definition of rPESs. The same applies for the finite causes property which is covered through finite configurations. Please note that the resulting rPESs has the same expressive power as the original PESs in [24] with respect to finite configurations since the axiom of finite causes is trivially satisfied for finite configurations. The concept of conflict heredity does not influence the expressive

power but ensures that syntactic and semantic conflicts coincide. In order to avoid confusion we call those structures relaxed Prime Event Structures (rPESs).

Definition 2.1. A *relaxed Prime Event Structure (rPES)* is a triple $\pi = (E, \#, \rightarrow)$ where

- E is a set of *events*,
- $\# \subseteq E^2$ is an irreflexive symmetric relation (the *conflict* relation),
- and $\rightarrow \subseteq E^2$ is the *enabling* relation.

The computational state of a process that is modelled as a rPES, is represented by the set of events that have occurred. Given a rPES $\pi = (E, \#, \rightarrow)$ we call such sets $C \subseteq E$ that respect $\#$ and \rightarrow configurations of π .

Definition 2.2 ([24]). Let $\pi = (E, \#, \rightarrow)$ be a rPES. A set of events $C \subseteq E$ is a *configuration* of π if

- it is *conflict-free*, that means for all $e, e' \in C$. $\neg(e\#e')$,
- *downward-closed*, that means for all $e, e' \in E$. $e \rightarrow e' \wedge e' \in C \implies e \in C$,
- and the transitive closure of the enabling relation is *acyclic*, that means $\rightarrow^* \cap C^2$ is free of cycles.

We denote *the set of configurations of π* by $C(\pi)$.

An event e is called impossible in a rPES if it does not occur in any of its configurations. Events can be impossible because of enabling cycles, an overlap between the enabling and the conflict relation, or because of impossible predecessors.

2.2 Event Structures for Resolvable Conflicts

Event Structures for Resolvable Conflicts (RCESs) were introduced in [12] as generalisation of earlier types of ESs to give semantics to general Petri nets. They allow to model the case where a and c cannot occur together until b takes place (i.e. initially a and c are in a conflict until the occurrence of b resolves this conflict, and therefore, the naming). An RCES consists of a set of events and an enabling relation between sets of events. Here, the enabling relation also models conflicts

between events. The behaviour is defined by a transition relation between sets of events that is derived from the enabling relation \vdash .

Definition 2.3 ([12]). An *Event Structure for Resolvable Conflicts (RCES)* is a pair $\rho = (E, \vdash)$ where

- E is a set of *events*
- and $\vdash \subseteq 2^E \times 2^E$ is the *enabling* relation.

In [12] several versions of configurations are defined. Here we consider only configurations which are both reachable and finite. Please note that the enabling relation \vdash can be considered as a witness relation in the following definition. For each subset of the new configuration we need a witness in the old one.

Definition 2.4 ([12]). Let $\rho = (E, \vdash)$ be an RCES and $X, Y \subseteq E$. Then:

$$X \mapsto_\rho Y \iff (X \subsetneq Y \wedge \forall Z \subseteq Y. \exists W \subseteq X. W \vdash Z)$$

We omit the subscript ρ and just write $X \mapsto Y$ if the RCES ρ is obvious from the context. The *set of configurations of ρ* is defined as

$$C(\rho) = \{X \subseteq E \mid \emptyset \mapsto_\rho^* X \wedge X \text{ is finite}\},$$

where \mapsto_ρ^* is the reflexive and transitive closure of \mapsto_ρ .

As an example consider the RCES $\rho = (E, \vdash)$ where $E = \{a, b, c\}$, $\{b\} \vdash \{a, c\}$, and $\emptyset \vdash X$ if and only if $X \neq \{a, c\}$. It models the initial conflict between a and c described above that can be resolved by b .

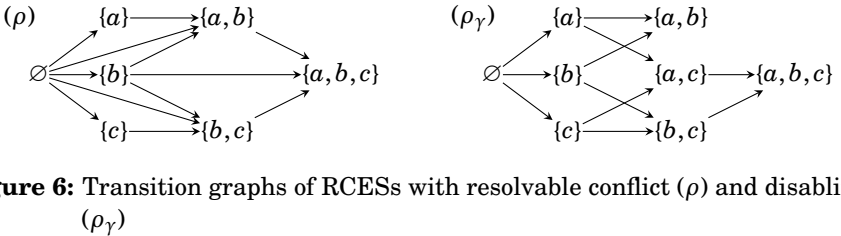


Figure 6: Transition graphs of RCESs with resolvable conflict (ρ) and disabling (ρ_γ)

In Figure 6 (ρ) the respective transition graph is shown (i.e. the nodes are all reachable configurations of ρ and the directed edges represent \mapsto_ρ). There is no transition from $\emptyset \mapsto_\rho \{a, b, c\}$ because of $\{a, c\} \subsetneq \{a, b, c\}$ and $\emptyset \not\vdash \{a, c\}$.

We consider two RCESSs as equivalent if they have the same transition graphs. Since we consider only reachable configurations, the transition equivalence defined below is denoted as reachable transition equivalence in [12].

Definition 2.5 ([12]). Two RCESSs $\rho = (E, \vdash)$ and $\rho' = (E', \vdash')$ are *transition equivalent*, denoted by $\rho \simeq_t \rho'$ if

- they have the same set of events $E = E'$
- and $\mapsto_\rho \cap (C(\rho))^2 = \mapsto_{\rho'} \cap (C(\rho'))^2$ the transition relations on the reachable configurations coincide.

We adapt the notion of transition equivalence to arbitrary types of ESs with a transition relation. Let x and y be two arbitrary types of ESs on that a transition relation is defined. We denote the fact that x and y have the same transition graphs by $x \simeq_t y$. For RCESSs transition equivalence is the most discriminating semantics studied in the literature. We consider two RCESSs as behaviourally equivalent if they have the same transition graphs.

2.3 Dynamic-Causality Event Structures

In [2] we started to dynamise the causal dependencies of events in event structures, meaning we allowed events to insert or to delete dependencies. There we have investigated both approaches first separately and then combined into the Dynamic-Causality ESs (DCESSs). Here we directly present the DCESSs and the most important result for this work: The incomparability of DCESSs and RCESSs. Note that we may omit the conflict relation from this definition because it can be modelled by mutual disabling using adders.

Figure 2 on Page 5 presents an agricultural example DCESS. In the following, we use the events a, c, d , and t , they refer to specific roles the event have: An adder adds a **cause** to a **target** and a **dropper** removes a **cause** from a **target**.

Definition 2.6 ([2]). A *Dynamic-Causality ES (DCESS)* is a quadruple $\Delta = (E, \rightarrow, \triangleright, \blacktriangleright)$ where

- E is a set of events,
- $\rightarrow \subseteq E^2$ the *initial causality* relation,

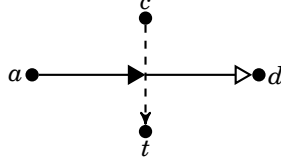


Figure 7: An order-sensitive DCES. The order of the **adder** a and the **dropper** d determines the if the **cause** c is needed for the **target** t .

- $\triangleright \subseteq E^3$ is the *shrinking causality* relation (the triple $(d, c, t) \in \triangleright$ is normally written as $d \triangleright [c \rightarrow t]$ to denote that the dependency $c \rightarrow t$ is absorbed by the dropper d),
- and $\blacktriangleright \subseteq E^3$ is the *growing causality* relation (the triple $(a, c, t) \in \blacktriangleright$ is normally written as $a \blacktriangleright [c \rightarrow t]$ to denote that the dependency $c \rightarrow t$ emerges from the adder a)

such that for all $a, d, c, t \in E$:

- (1) $d \triangleright [c \rightarrow t] \wedge \nexists a \in E. a \blacktriangleright [c \rightarrow t] \implies c \rightarrow t$,
- (2) $d \triangleright [c \rightarrow t] \implies d \notin \{c, t\}$,
- (3) $a \blacktriangleright [c \rightarrow t] \wedge \nexists d \in E. d \triangleright [c \rightarrow t] \implies \neg(c \rightarrow t)$,
- (4) $a \blacktriangleright [c \rightarrow t] \implies a \notin \{c, t\}$,
- (5) $a \blacktriangleright [c \rightarrow t] \implies \neg(a \triangleright [c \rightarrow t])$.

The Conditions (1) and (3) are sanity constraints (i.e. a deletion of a dependency is only allowed if this dependency is initially present or could be added by another event, and vice versa). The Conditions (2) and (4) forbid that an adder or a dropper targets itself. Adding or dropping itself as a cause is obviously not needed since we we add or drop a true condition. Adding or dropping itself as a target is also not needed since events are non-repeatable. If there are droppers and adders for the same causal dependency, we do not specify whether this dependency is contained in the initial causality because the semantics depend on the order in which the droppers and adders occur. Condition (5) prevents that a modifier adds and drops the same cause for the same target.

The order in which droppers and adders occur determines the causes of an event. For example, assume $a \blacktriangleright [c \rightarrow t]$ and $d \triangleright [c \rightarrow t]$ (as depicted in Figure 7), then after first a then d , t does not depend on c , whereas after first d then a , t depends on c . Thus, configurations like $\{a, d\}$ are not expressive enough to represent the state of such a system.

Therefore, a state in a DCES is a pair of a configuration C and a current causality relation \rightarrow_C , which contains the current causal dependencies. Please note that the initial state is the only state with an empty set of events; for non-empty sets of events there may be multiple states. For the next definition we need a little additional notation:

Definition 2.7. Let $\Delta = (E, \rightarrow_{in}, \triangleright, \blacktriangleright)$ be a DCES and $\text{old}, \text{new} \subseteq E$. Then the set of dependencies dropped by events in $(\text{new} \setminus \text{old})$ is defined as

$$\triangleleft_{\text{old}, \text{new}} := \{(c, t) \mid \exists d \in (\text{new} \setminus \text{old}). d \triangleright [c \rightarrow t]\}.$$

And the set of dependencies added by events in $(\text{new} \setminus \text{old})$ is defined as

$$\blacktriangleright_{\text{old}, \text{new}} := \{(c, t) \mid \exists a \in (\text{new} \setminus \text{old}). a \blacktriangleright [c \rightarrow t]\}.$$

The behaviour of a DCES is defined by the transition relation on its reachable states with finite configurations.

Definition 2.8 ([2]). Let $\Delta = (E, \rightarrow_{in}, \triangleright, \blacktriangleright)$ be a DCES and $\text{old}, \text{new} \subseteq E$. Then $(\text{old}, \rightarrow_{\text{old}}) \rightarrow_d (\text{new}, \rightarrow_{\text{new}})$ if:

- (1) $\text{old} \subsetneq \text{new}$
- (2) $\forall t \in \text{new} \setminus \text{old}. \{c \mid (c, t) \in \rightarrow_{\text{old}}\} \subseteq \text{old}$
- (3) $\rightarrow_{\text{new}} = (\rightarrow_{\text{old}} \setminus \triangleleft_{\text{old}, \text{new}}) \cup \blacktriangleright_{\text{old}, \text{new}}$
- (4) $\forall (c, t) \in \blacktriangleright_{\text{old}, \text{new}} \cap \triangleleft_{\text{old}, \text{new}}. (c \in \text{old} \vee t \in \text{old})$
- (5) $\forall t, a \in \text{new} \setminus \text{old}. \forall c \in E. a \blacktriangleright [c \rightarrow t] \implies c \in \text{old}.$

Condition (1) ensures the accumulation of events. Condition (2) ensures that only events which are enabled after old can take place in new. Condition (3) ensures the correct update of the current causality relation with respect to this transition. To keep the theory simple, Condition (4) avoids race conditions; it forbids the occurrence of an adder and a dropper of the same still relevant causal

dependency within one transition. Condition (5) ensures that DCESSs generalises the GESs, it prevents the concurrent occurrence of an adder and its target since they are not independent. One exception is when the target has already occurred. In that case, the adder does not change the target's predecessors.

Definition 2.9 ([2]). Let $\Delta = (E, \rightarrow_{in}, \triangleright, \blacktriangleright)$ and $\Delta' = (E', \rightarrow'_{in}, \triangleright', \blacktriangleright')$ be two DCESSs. We call them *transition equivalent* if they allow for the same transition sequences and write $\Delta \simeq_{ts} \Delta'$.

That means

- for any sequence $\emptyset = X_0, X_1, \dots, X_n$ with $X_i \subsetneq X_{i+1}$ and $X_n \in E \cap E'$,
- we have current causality relations $\rightarrow_{X_0} = \rightarrow_{in}, \rightarrow_{X_1}, \dots, \rightarrow_{X_n}$ such that $(X_i, \rightarrow_{X_i}) \rightarrow_d (X_{i+1}, \rightarrow_{X_{i+1}})$ in Δ ,
- if and only if we have current causality relations $\rightarrow'_{X_0} = \rightarrow'_{in}, \rightarrow'_{X_1}, \dots, \rightarrow'_{X_n}$ such that $(X_i, \rightarrow'_{X_i}) \rightarrow_d (X_{i+1}, \rightarrow'_{X_{i+1}})$ in Δ' .

As visualised by the DCESS in Figure 7, the order of events may be relevant for the behaviour of a DCESS.

DCESSs and RCESSs are incomparable. On the one hand RCESSs can express the disabling of an event c after a conjunction of events a and b as visualised by ρ_γ of Figure 6 on Page 15, whereas DCESSs can only model disabling after single events.

Lemma 2.10 ([2]). *There is no transition-equivalent DCESS to ρ_γ of Figure 6.*

On the other hand RCESSs cannot distinguish the order of occurrences of events as it is done for the order of a and d in the DCESS in Figure 7. Therefore, both structures are incomparable.

Theorem 2.11 ([2]). *DCESSs and RCESSs are incomparable.*

2.4 Dynamic Condition Response Graphs

In 2010 Hildebrand and Mukkamala [14] introduced the Dynamic Condition Response Graphs (DCR-Graphs): “*The model language is inspired by and a conservative generalization of the declarative process matrix model language used by our industrial partner and prime event structures.*” There is a web tool for working with DCR-Graphs at <http://tiger.itu.dk>.

Before introducing the DCR-Graph definition, we present the authors' derivation of. First PESs were extended to Condition Response ESs (CRESs) by introducing an acceptance criterion, that checks if some initially specified or subsequently added response events occurred (or some conflicting events). Thereafter the authors allowed for events to re-occur and also to exclude and re-include events from the structure (therefore, DCR-Graphs are called dynamic). Since DCR-Graphs allow for the re-occurrence of events a finite representation of possibly infinite behaviour is gained. It is also shown that there is a natural inclusion of CRESs into DCR-Graphs.

Hildebrand and Mukkamala first recap the definition of labelled Prime ESs, but in contrast to our approach they allow for infinite event sets, a set of action Act and a labelling function l for the events into the actions. They demand two properties of the causality relation which we omitted, namely conflict heredity and the finite cause property. In order to avoid confusions we use the abbreviation LPES for these labelled PES. All definitions and the proposition in this subsection are originally from [14].

Definition 2.12. A *labelled Prime Event Structure (LPES)* is a 5-tuple $P = (E, \text{Act}, \leq, \#, l)$ where

- E is a (possibly infinite) set of *events*,
- Act is the set of *actions*,
- $\leq \subseteq E \times E$ is the *causality* relation between events which is a partial order,
- $\# \subseteq E \times E$ is a binary *conflict* relation between events which is irreflexive and symmetric,
- $l : E \rightarrow \text{Act}$ is the *labelling function* mapping events to actions.

The causality and conflict relations must satisfy the conditions:

- (1) (*conflict heredity*) $\forall e, e', e'' \in E. e \# e' \wedge e' \leq e'' \implies e \# e''$,
- (2) (*finite cause property*) $\forall e \in E. e \downarrow := \{e' \mid e' < e\}$ is finite.

A *configuration* of P is a set $C \subseteq E$ of events satisfying the conditions:

- (1) *conflict-free*: $\forall e, e' \in C. \neg e \# e'$,
- (2) *downwards-closed*: $\forall e \in C, e' \in E. e' \leq e \implies e' \in C$.

A run ρ of a IPES P is a (possibly infinite) sequence of labelled events $(e_1, l(e_1)), (e_2, l(e_2)), \dots$ such that for all $i \geq 0$ the set $\bigcup_{0 \leq j \leq i} \{e_j\}$ is a configuration. A run $(e_1, l(e_1)), (e_2, l(e_2)), \dots$ is *maximal* if any enabled event eventually happens or becomes in conflict, formally $\forall e \in E, i \leq 0. e \downarrow \subseteq e_i \downarrow \cup \{e_i\} \implies \exists j \geq 0. (e \# e_j \vee e = e_j)$.

Next Hildebrand and Mukkamala extended the IPES to Condition Response ESs by introducing a notion of acceptance. Therefore, they introduced a third relation on the events, namely the response relation and a set of initially needed responses. The semantics are inherited from the IPESs but extended with the acceptance criterion: Any pending response event (initial or added by an event with the response relation) must eventually happen or be in conflict with the current configuration.

Definition 2.13. A labelled Condition Response ES (CRES) over an alphabet Act is a 6-tuple $(E, \text{Re}, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \#, l)$ where

- $(E, \text{Act}, \rightarrow\bullet, \#, l)$ is a IPES, referred to as underlying ES,
- $\bullet\rightarrow \subseteq E \times E$ is the *response* relation between events, satisfying that $\rightarrow\bullet \cup \bullet\rightarrow$ is acyclic,
- $\text{Re} \subseteq E$ is the *set of initial responses*.

The *configuration* C and *run* ρ of a CRES are defined as respectively a configuration and a run of the underlying ES. A run $(e_1, l(e_1)), (e_2, l(e_2)), \dots$ is *accepting* if $\forall e \in E, i \geq 0. e_i \bullet\rightarrow e \implies \exists j \geq 0. (e \# e_j \vee (i < j \wedge e = e_j))$ and $\forall e \in \text{Re}. \exists j \geq 0. (e \# e_j \vee e = e_j)$. In words, any pending response event must eventually happen or be in conflict.

Definition 2.14. A Dynamic Condition Response Graph (DCR-Graph) is the 7-tuple $G := (E, M, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, l)$ where

- E is the set of *events*,
- $M \in \mathcal{M}(G) := 2^E \times 2^E \times 2^E$ is the *marking* and $\mathcal{M}(G)$ is the *set of all markings*,
- Act a *set of actions*,
- $\rightarrow\bullet \subseteq E \times E$ is the *condition* relation,
- $\bullet\rightarrow \subseteq E \times E$ is the *response* relation,
- $\pm : E \times E \rightarrow \{+, \%\}$ defines the *dynamic inclusion/exclusion* relations by $e \rightarrow + e'$ if $\pm(e, e') = +$ and $e \rightarrow \% e'$ if $\pm(e, e') = \%$,

- $l : E \rightarrow \text{Act}$ is a *labelling function*, that maps every event to an action.

The semantics of a DCR-Graph are defined by a transition system on its markings labelled with pairs of events and their labels. A run of a DCR-Graph correspond to a path in the transition system. Such a run is considered to be accepting if no response event is continuously included and pending without it happening (note for finite runs to be accepting this means that there are no included restless events in the final marking).

Definition 2.15. For a DCR-Graph $G = (E, M, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, l)$ the *corresponding labelled transition system* $T(G)$ is the 3-tuple $(\mathcal{M}(G), M, \rightarrow \subseteq \mathcal{M} \times (E \times \text{Act}) \times \mathcal{M})$ with

- $\mathcal{M}(G)$ is the set of markings of G ,
- $M \in \mathcal{M}(G)$ is the initial marking,
- and $\rightarrow \subseteq \mathcal{M} \times (E \times \text{Act}) \times \mathcal{M}$ is the transition relation given by $M' \xrightarrow{(e,a)} M''$

where

- $M' = (\text{Ex}', \text{Re}', \text{In}')$ is the marking before transition,
- $M'' = (\text{Ex}' \cup \{e\}, \text{Re}'', \text{In}'')$ is the marking after the transition,
- $e \in \text{In}'$ and $l(e) = a$,
- $\{e' \in \text{In}' \mid e' \rightarrow\bullet e\} \subseteq \text{Ex}'$,
- $\text{In}'' = (\text{In}' \cup \{e' \mid e \rightarrow + e'\}) \setminus \{e' \mid e \rightarrow \% e'\}$,
- $\text{Re}'' = (\text{Re}' \setminus \{e\}) \cup \{e' \mid e \bullet\rightarrow e'\}$.

A run $(e_0, a_0), (e_1, a_1), \dots$ of the transition system is a sequence of labels of a sequence of transitions $M_i \xrightarrow{(e_i, a_i)} M_{i+1}$ where $M_i = (\text{Ex}_i, \text{Re}_i, \text{In}_i)$ and $M_0 = M$. A run is *accepting* if $\forall i \geq 0, e \in \text{Re}_i \cdot \exists j \geq i. (e = e_j \vee e \notin \text{In}_j)$. In words, a run is accepting if no response event is continuously included and pending without it happening.

There is a simple encoding of CRES into DCR-Graphs. First any event should exclude itself as in DCR-Graphs they are re-executable in contrast to CRESs. Second the conflict $e\#e'$ in a CRES can be modelled by mutual exclusion in a DCR-Graph (cf. Figure 8).

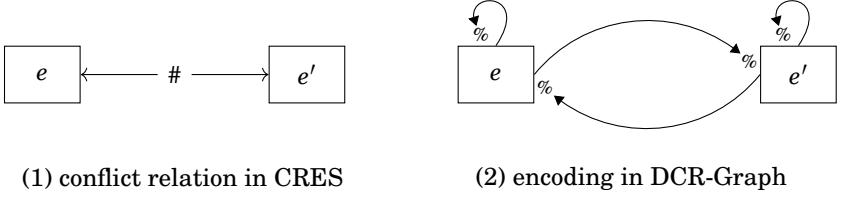


Figure 8: Encoding of conflicting events. (1) the conflicting events e and e' in a CRESs. (2) conflict is modelled as mutual exclusion in DCR-Graphs.

Proposition 2.16. *Let $C = (E, \text{Re}, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \#, l)$ be a CRES and $D = (E, M, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, l)$ be a DCR-Graph with*

- *the initial marking $M = (\emptyset, \text{Re}, E)$*
- *and dynamic exclusion $\pm(e, e') = \%$ if $e = E'$ or $e\#e'$ and undefined otherwise.*

Then C and D have the same runs and accepting runs.

In Chapter 4 we consider a slightly different definition of DCR-Graphs that allows more easily to define true concurrency semantics for DCR-Graphs in order to present an encoding of DCR-Graphs into HDESs.

In Section 5.1 we present a case study once modelled using both DCESs and DCR-Graphs. The focus is on the DCES modelling, however.

3 Higher-Order Dynamics

Based on the incomparability result in Theorem 2.11, we extend the DCEs formalism by two means, set-based dynamics and higher-order dynamics. This will also solve certain shortcomings of the DCEs formalism exhibited in the case study in Section 5.1.

3.1 Higher-Order Dynamic-Causality ESs

Higher-Order Dynamic-Causality ESs (HDESs) are a recent generalisation of DCEs [16]. They allow for higher-order dynamicity (e.g. an adding of an adding) and also for set-based (i.e. conjunctive) modifications (e.g. a dropping after a and b). Figure 9 on Page 26 presents a small example of a second-order adding: After a diagnosis, a treatment becomes possible. Now, a doctor might join the treatment team and nothing changes. But if she read a medical paper before joining, in which an additional test before the treatment was suggested, the joining of the doctor leads to an additional precondition, namely the test for the treatment. This behaviour is modelled with a second order rule, `dr_x_reads_paper ► [dr_x_joins ► [test → treatment]]`, and the zeroth order rule `diagnosis → treatment` (for enhanced readability of the rules we used the notation of the web tool, as in Section 3.2).

Our approach is similar to the definition of DCEs in [2], but instead of updating a causality relation, we use a rule set which is updated after each transition. The following definitions are revised versions of those in [16]. We indicate where we differ from the original version. Again, we omit the conflict relation from our new structures. Now, we define HDES rule sets, further notation is necessary before we are ready to define HDESs:

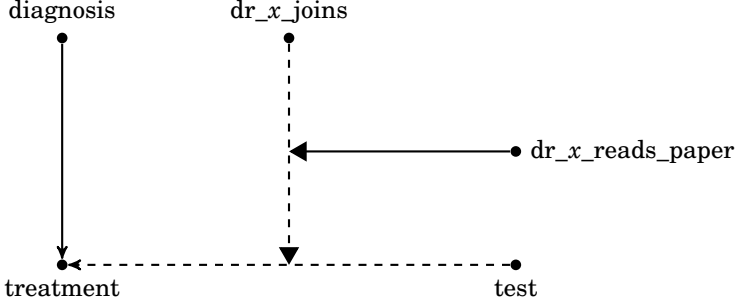


Figure 9: A HDES example with second-order dynamics. **THESIS Example 2** with second-order dynamics: Dashed arrows denote initially absent dependencies or initially absent dynamic rules. Advice for a printed version: Please visit <http://hdes.mtv.tu-berlin.de> and load THESIS Example 2.

Definition 3.1. For a given set of events E , a *HDES rule* is produced by the following grammar, with start symbol \mathbf{S} :

$$\begin{aligned}
\mathbf{S} &\longrightarrow \mathbf{Z} \mid \mathbf{F} \mid \mathbf{H} \\
\mathbf{Z} &\longrightarrow [c \rightarrow t] && c, t \in E \\
\mathbf{F} &\longrightarrow M \blacktriangleright [c \rightarrow t] \mid M \triangleright [c \rightarrow t] && M \subseteq E \wedge (c, t \in E \setminus M) \\
\mathbf{H} &\longrightarrow A \blacktriangleright [\mathbf{R}] \mid D \triangleright [\mathbf{R}] && A, D \subseteq E \\
\mathbf{R} &\longrightarrow \mathbf{H} \mid \mathbf{F}
\end{aligned}$$

A rule is either a *zeroth order rule* created by \mathbf{Z} , or a *first order rule* created by \mathbf{F} , or a *higher order rule* created by \mathbf{H} . A higher order rule adds or drops a rule recursively generated by \mathbf{R} that itself is either a higher order rule or a first order rule.

- For a given rule r , we call the *rank of r* or the *order of r* , written as $\text{rk}(r)$, the number of \blacktriangleright symbols plus the number of \triangleright symbols occurring in r .
- For a rule set R , we call its restriction on the rules of rank zero the *causality relation*, written as \rightarrow_R (or \rightarrow if the rule set is obvious from the context).

- Let $r = M_1 \text{op}_1 [\dots M_k \text{op}_k [c \rightarrow t] \dots]$ with $\text{op}_i \in \{\triangleright, \blacktriangleright\}$ for $1 \leq i \leq k$ be a rule of rank k . Then r_i denotes the *sub-rule*

$$r_i = M_i \text{op}_i [\dots M_k \text{op}_k [c \rightarrow t] \dots]$$

for $1 \leq i \leq k$.

- The set of all rule sets with respect to an event set E is called $\text{RuleSets}(E)$.

The grammar defined above is more complicated than the original in [16], because we want to prevent that the target or the cause are members of the modifier set in first-order rules. This is a generalisation of the condition in DCESSs, that neither adder nor dropper may be the cause or the target, as defined in [2]. Based on the HDES rule sets we can now define Higher-Order Dynamic-Causality ESSs:

Definition 3.2. A *Higher-Order Dynamic-Causality ES (HDES)* is a tuple $H = (E, R)$ where

- where E is an event set
- and $R \in \text{RuleSets}(E)$, that means R is a HDES rule set with respect to E .

The *order of a HDES* $H = (E, R)$ is the maximal order of any rule $r \in R$.

Like in DCESSs, configurations are not expressive enough to capture the behavioural state of a HDES.

Definition 3.3. Let $H = (E, R)$ be a HDES, $C \subseteq E$, $R_C \in \text{RuleSets}(E)$. Then we call

- (C, R_C) a *state of H*
- and $S_0 := (\emptyset, R)$ the *initial state of H* if not explicitly defined differently.

For a state $s = (C, R)$ we define

- *its rule set* as the projection to the rule set $\text{rs}(s) := R$
- and *its configuration* as the projection to the configuration $\text{con}(s) := C$.

Like in the case of DCEs, the behaviour of a HDES is defined by the transition relation between its states, but before we can define such transitions, we define a rule update for a state with respect to a set of events. In the rule update for a transition, we only consider rules whose modifier sets are subsets of the new set of events, but were no subsets of the old set of events in the old state. Dropping rules are executed first (if they are not dropped themselves), and adding rules are executed thereafter. Further, we forbid certain concurrent occurrences of events for an adding rule: If a rule adds a dependency $A \blacktriangleright [c \rightarrow t]$, then t is only allowed to be in the same transition as A if the cause c is element of the old set of events.

Definition 3.4. Let $s_{\text{old}} = (\text{old}, R_{\text{old}})$ be a state of a HDES $H = (E, R)$ and $\text{new} \subseteq E$ with $\text{old} \subsetneq \text{new}$. Then we call

- a rule $M \text{ op}[r]$ *active* denoted as $r \in \text{ac}(s_{\text{old}}, \text{new})$ if $(M \subseteq \text{new}) \wedge (M \not\subseteq \text{old})$,
- otherwise we call it *passive*.
- We call a rule r *independent* denoted as $r \in \text{ind}(\text{old}, R_{\text{old}})$ if there is no active rule r' dropping r .

The *rule update* R_{new} is the result of applying Algorithm 1 to s_{old} and the new set of events. We denote this relation as $s_{\text{old}} \longrightarrow_{\text{new}} R_{\text{new}}$.

The algorithm consists of three parts. In the first part the dropping rules are taken into consideration (Lines 2 to 8) and the independent ones are executed (if active — Line 6) or copied to the new set (if passive — Line 8) step by step. Note that the set of independent rules might change since R_{old} gets modified by the algorithm. In the second part (Lines 11 to 16) each adding rules is executed if active (Line 14), else copied to the new set (Line 16). Finally, all causality rules (rules of the form $[c \rightarrow t]$) are copied to the new set (Line 19) and the result is returned.

Definition 3.5. Let $H = (E, R)$ be a HDES, and $s_{\text{old}} = (\text{old}, R_{\text{old}})$ and $s_{\text{new}} = (\text{new}, R_{\text{new}})$ two of its states. Then $s_{\text{old}} \mapsto_H s_{\text{new}}$ if

- (1) $\text{old} \subsetneq \text{new}$
- (2) $\forall t \in \text{new} \setminus \text{old}. \forall c \in E. c \rightarrow_{R_{\text{old}}} t \implies c \in \text{old}$
- (3) $s_{\text{old}} \longrightarrow_{\text{new}} R_{\text{new}}$
- (4) $\forall r \in R_{\text{old}}. (r = M \blacktriangleright [c \rightarrow t] \wedge (r \in \text{ac}(s_{\text{old}}, \text{new}) \wedge t \in \text{new} \setminus \text{old})) \implies c \in \text{old}$

input : A HDES state $s_{\text{old}} = (\text{old}, R_{\text{old}})$ of a HDES $H = (E, R)$ and a set of events new , with $\text{old} \subsetneq \text{new} \subseteq E$

output: A set of HDES rules R_{new}

```

1  $R_{\text{new}} \leftarrow \emptyset;$  // The output set is constructed step by step.
2 while  $\exists r \in R_{\text{old}}, M \subseteq E. r = M \triangleright [r']$  do // while there are dropping
   rules
3   let  $r = M \triangleright [r'] \in R_{\text{old}} \wedge r \in \text{ind}((\text{old}, R_{\text{old}}));$  // chose a independent
   dropping rule (with respect to the current  $R_{\text{old}}$ )
4    $R_{\text{old}} \leftarrow R_{\text{old}} \setminus \{r\};$  // remove the rule from the old set
5   if  $r \in \text{ac}(s_{\text{old}}, \text{new})$  then // active or passive rule?
6   |  $R_{\text{old}} \leftarrow R_{\text{old}} \setminus \{r'\};$  // drop the target from old set
7   else
8   |  $R_{\text{new}} \leftarrow R_{\text{new}} \cup \{r\};$  // copy the rule to the new set
9   end
10 end
11 foreach  $r = M \blacktriangleright [r'] \in R_{\text{old}}$  do // for each adding rule
12 |  $R_{\text{old}} \leftarrow R_{\text{old}} \setminus \{r\};$  // remove the rule from the old set
13 | if  $r \in \text{ac}(s_{\text{old}}, \text{new})$  then // active adding rule?
14 | |  $R_{\text{new}} \leftarrow R_{\text{new}} \cup \{r'\};$  // add the target to the new set
15 | else
16 | |  $R_{\text{new}} \leftarrow R_{\text{new}} \cup \{r\};$  // add the rule to the new set
17 | end
18 end
19  $R_{\text{new}} \leftarrow R_{\text{new}} \cup R_{\text{old}};$  // copy the causal rules to the new set
20 return  $R_{\text{new}};$ 

```

Algorithm 1: The HDES rule set update algorithm

Condition (1) ensures an accumulation of events and Condition (2) the downward closure under the current causality relation. Condition (3) ensures that all rules, whose modifier sets are included in new but not in old are executed, and the new rule set is adjusted. The last Condition (4) is a generalisation of the DCES condition which forbids the concurrency of an adder and its target if the cause did not occur previously.

The original paper [16] imposed an additional condition that was meant to prevent that one rule is added and dropped by the same modifier in a single transition, but as stated in the paper it forbids certain possibly interesting behaviour. Thus, the condition is dropped.

By this definition, we may add rules which never become active: For example, if we have an active adding rule like $M_1 \blacktriangleright [M_2 \blacktriangleright [r]]$, and M_2 is contained in the current configuration old unified with the modifier set M_1 . Such a behaviour could be prevented by a more strict version of Condition (4). Such rules will be removed by the clean-up (see below in Subsection 3.3.1), certain rules could even be removed initially (cf. redundant rules in Subsection 3.4.3).

To compare two HDESs, we define a proper equivalence based on the transition behaviour. This is a straightforward generalisation of the equivalence definition for DCESs in Definition 2.9.

Definition 3.6. Let $H = (E, R_\emptyset)$ and $H' = (E', R'_\emptyset)$ be two HDESs. We call them *transition-equivalent* if they allow for the same transition sequences with respect to the events and write $H \simeq_{\text{ts}} H'$. That means

- for any sequence $\emptyset = X_0, X_1, \dots, X_n$ with $X_i \subsetneq X_{i+1}$ and $X_n \in E \cap E'$,
- we have rule sets $R_{X_0} = R_\emptyset, R_{X_1}, \dots, R_{X_n}$ such that $(X_i, R_{X_i}) \mapsto_H (X_{i+1}, R_{X_{i+1}})$ in H ,
- if and only if we have rule sets $R'_{X_0} = R'_\emptyset, R'_{X_1}, \dots, R'_{X_n}$ such that $(X_i, R'_{X_i}) \mapsto_{H'} (X_{i+1}, R'_{X_{i+1}})$ in H' .

This equivalence definition abstracts from the non-observable internal behaviour. For example, we do not want to distinguish between two structures if they only differ in the names of some impossible events.

3.2 The Web Tool

We¹ have developed a web tool (<http://hdes.mtv.tu-berlin.de/>) with the purpose of simulating and exploring the behaviour of HDESs (and also DCESs). Example 3.1 shows the web tool version (THESIS Example 2) of the HDES from Figure 9. Advice for a printed version: Whenever you read THESIS Example X, please load THESIS Example X in the web tool.

3.2.1 The Syntax of the Web Tool

The first line `@hdes` declares that we define a HDES. We could have also used `@dces`

```

1 @hdes
2
3 diagnosis(x=50, y=50)
4 dr_x_joins(x=200, y=50)
5 dr_x_reads_paper(x=350, y=125)
6 test(x=350, y=200)
7 treatment(x=50, y=200)
8
9 diagnosis → treatment
10 dr_x_reads_paper ►[dr_x_joins ►[test → treatment]]
11
12 @c "An example HDES with second-order adding."
```

Example 3.1: HDES example from Figure 9.

to define a DCES (or even `@rpes` for rPESs). The next block contains positioning information for visualising the ES. For example `diagnosis(x=50, y=50)` means that the event `diagnosis` will be initially positioned 50 pixel to the right and 50 pixel below the upper left corner. If one omits such declarations, the tool tries to position the events automatically. Dependencies are denoted as `c --> t`, the addition of a dependency as `a +> [c --> t]` (higher order adding works the same way as in the above example `dr_x_reads_paper +> [dr_x_joins +> [test --> treatment]]`), and the deletion is denoted as `d |> [c --> t]`. There can be

¹ that means mostly our student assistant Benjamin Bisping

arbitrary finite sets as modifiers (adders and droppers), for example $\{x, y, z\} \rightarrow [c \rightarrow t]$, separated by commas and encapsulated with curly brackets, and comments can be stated as `@c "This is a comment."`

Note that the notation in this thesis $a \triangleright [c \rightarrow t]$ and $d \triangleright [c \rightarrow t]$ is slightly different to the pure ASCII notation used in the web tool $a \rightarrow [c \rightarrow t]$ and $d \rightarrow [c \rightarrow t]$.

The keyword `@addThenDrop` applies the alternative rule update strategy as defined below in Subsection 3.4.1.




3.2.2 The GUI of the Web Tool

The GUI of the web tool consists of three main parts: The menu bar on the top, the text window on the left, and the visualisation window on the right. Using the menu bar one can load predefined examples (e.g. all examples of this thesis) or local files, export the current structure as a textual representation or as SVG file, apply the encodings of this thesis (see Subsection 3.4.2 for a kind of normalisation of HDESs, Subsection 3.5.1 for embedding DCES into HDES, and Chapter 4 for simulating basic DCR-Graphs with HDES), and switch between straight and bended arrows (the later ones are helpful if arrows are displayed on top of each other like in $c \rightarrow t$ and $c \triangleright [c \rightarrow t]$ see [THESIS Example 3](#)) or activate (respectively deactivate) an automatic rule clean-up after each transition (see Subsection 3.3.1) in the settings.

In the second row, one can switch between the editing (Edit) and the execution (Run) mode. Depending on the mode, the tool shows different buttons in the third row. In editing mode, the first button allows one to move events, the second one to create or rename events, the third one to insert dependencies, the fourth one to introduce binary conflicts (in rPESs), the fifth button to insert an adding, the sixth one to insert a dropping, and the last one is for event refinement, which is currently under development.

In the upper left corner of the visualisation window is a small button to toggle the text window (if the text window is hidden, this button is on the left border with the line numbers).

In execution mode, the text window contains the current rule set with respect to the selected rule update strategy (Definition 3.4 or Definition 3.20). The history of already performed steps is in `@hdesCfgr`. Initially it is empty. One can click on enabled events like `diagnosis` depicted as big green discs (disable events have big grey discs and executed ones smaller black discs). After the execution, the event will appear in the left window `@hdesCfgr "{diagnosis}"` and the rule set

will get updated with respect to this transition. One can also select more than one event by holding  and drawing a frame with a pressed left mouse button, or by hold  and adding events by left clicking them. . A button labelled **Parallel Step** will appear. If the selected events are concurrently enabled, the button is green and one can press it to perform the concurrent step if not this button is greyed out. Using the arrow buttons in the third row, one can undo and redo transition steps. Using the  button, one can perform a clean-up step on the current rule set, which will remove certain superfluous rules (see Subsection 3.3.1). In the Transition System menu, one can create a graphviz file (at <http://www.graphviz.org/> a web tool for those can be found) or create a list of all possible paths in the transition system.

Remark 3.7. The web tool was extremely useful for testing purposes when designing HDES encodings (as in Subsection 3.4.2 and Section 4.2) since we could check if the encoding did what it was supposed to do. This made it possible to identify many mistakes before proving.

3.3 Concepts for Working with HDESs

In the following section, we introduce two tools: a clean-up of a state (in Subsection 3.3.1), which removes certain obsolete rules, and the synchronisation pattern (in Subsection 3.3.2), with which we enforce certain effects if events occur simultaneously.

3.3.1 Clean-Up

Here, we define a rule set clean-up with respect to a specific state (i.e. we remove rules from the rule set which do not influence the observable behaviour). The idea is to remove all rules which will clearly never have any observable effect.

In a state (C, R_C) any rule $r = M_1 \text{ op}_1 [\dots M_k \text{ op}_k [c \rightarrow t] \dots] \in R_C$ can be removed from the rule set R_C if there is a $1 \leq i \leq k$ with $M_i \subseteq C$. This will not affect the behaviour, since r will have no influence on enabled events or constrain concurrency. We can also remove any rule $M_1 \text{ op}_1 [\dots M_k \text{ op}_k [c \rightarrow t] \dots] \in R_C$ – even the causality rule $c \rightarrow t$ itself, if $c \in C$ or $t \in C$. This dependency is meaningless if the target or the cause this dependency has already occurred. Adding or dropping it has no observable effect.

Definition 3.8. Let $H = (E, R)$ be a HDES and $s = (C, R_C)$ a state of H . The *clean-up of s* is the state $\text{cu}(s) := (C, R'_C)$, where

- R'_C can be obtained from R_C as follows:
 - Drop every rule $r = M_1 \text{ op}_1 [\dots M_k \text{ op}_k [c \rightarrow t] \dots] \in R_C$
if there is a $1 \leq i \leq k$ with $M_i \subseteq C$,
 - **or** $c \in C$,
 - **or** $t \in C$
 - **else** add it to R'_C .
- Note, that for $k = 0$ the rule r reduces to $c \rightarrow t$.

Now, we can state the claim that the clean-up does not change the behaviour.

Lemma 3.9. *Let $H = (E, R)$ be a HDES and $s = (C, R_C)$ a state of H and $s' := \text{cu}(s)$ be the clean-up of s . Then $H_s := (E, R_C)$ with initial configuration C and $H_{s'} := (E, \text{rs}(s'))$ with initial configuration C are transition-equivalent.*

We show this equivalence via induction over the rank of the deleted rule.

Proof. Let r be a rule removed by the clean-up with $\text{rk}(r) \leq 1$. We have $r = c \rightarrow t$ with $c \in C$ or $t \in C$. In both cases, the dependency does not constrain any future transitions (by Definition 3.4), or $r = M \blacktriangleright [c \rightarrow t]$ or $r = M \triangleright [c \rightarrow t]$. If $M \subseteq C$, by Definition 3.4, r never becomes active in H_s . It cannot change the rule set. On the other hand, an adding rule could only influence the behaviour by restricting concurrency if it was active. Thus, deleting r does not change the observable behaviour and $H_s \simeq_{\text{ts}} H_{s'}$. If $M \not\subseteq C$, it follows $c \in C$ or $t \in C$. But then, this rule will only add or drop a meaningless dependency and can be deleted.

Let now r be a rule removed in the clean-up with $n := \text{rk}(r) > 1$ such that for all $1 \leq i \leq k$. $M_i \not\subseteq C$. It follows $c \in C$ or $t \in C$, but then this rule will finally only add or drop a meaningless dependency and can be deleted.

For the remainder of the proof we may assume there is a $1 \leq i \leq k$. $M_i \subseteq C$. Let now r be a rule removed in the clean-up with $n := \text{rk}(r) > 1$ if $r = M \triangleright [r']$ and r never becomes active (or active but always not independent), it can obviously be removed. Assume there is a transition sequence in H_s such that r gets executed. Then, the rule r' will get removed (if it exists), but since r got deleted in the clean-up, there is a modifier set M_i with $i > 1$ contained in C (otherwise r would not be active since then $M_1 \in C$). Thus, M_i is also a modifier of r' , and therefore,

r' and any rule which could add r' would also be deleted in the clean-up. By induction and since $\text{rk}(r') < \text{rk}(r)$, we also know that the removal of r' does not change the behaviour. The only difference would be when those rules get deleted. Also in this case $H_s \simeq_{\text{ts}} H_{s'}$.

Let finally r be a rule removed in the clean-up with $n := \text{rk}(r) > 1$ and $r = M \blacktriangleright [r']$, and assume there is a transition sequence in H_s such that r gets executed (if none exists, we clearly have $H_s \simeq_{\text{ts}} H_{s'}$). So in H_s the rule r would add a rule r' such that there is a modifier in r' which is already contained in C and $\text{rk}(r') < \text{rk}(r)$. Thus by induction, r' does not change the behaviour and there is no need to add it, and therefore, $H_s \simeq_{\text{ts}} H_{s'}$. \square

Up to now, we have removed irrelevant rules. The following lemma shows that in certain situations there are irrelevant events in the configuration of the current state which can be removed.

Lemma 3.10. *Let $H = (E, R)$ be a HDES and let $s = (C, R_C)$ be a state of H , let $D \subseteq C$ such that*

- *no $d \in D$ occurs in any rule of R_C*
- *neither as modifier*
- *or as part of the targeted dependency.*

Then the HDES $H' := (E \setminus D, R_C)$ allows initially for exactly the same transition sequences as H in the state s , where the initial state of H' is $(C \setminus D, R_C)$.

Proof. By Definition 3.5 (resp. by Definition 3.21), the events in D are considered at two points. First, in order to check if events are enabled (i.e. all predecessor have occurred), and second in the rule update to check if a rule is active or not. But since no $d \in D$ occurs in any modifier set (or targeted dependency) of R_C , nor in any causality rule, it is possible to reduce the structure to a HDES over $E \setminus D$ with an initial state $(C \setminus D, R_C)$. \square

Corollary 3.11. *In the special case $D = C$, the reduction according to Lemma 3.8 yields a HDES over the reduced event set $E \setminus C$ with initial state (\emptyset, R_C) .*

Proof. Immediate by Lemma 3.10. \square

Note, that in Subsection 3.4.3 a state independent approach for detecting superfluous rules is stated. That approach will detect potentially less rules to be superfluous, but can do it before the system runs.

On the one hand, there are rules that will be detected as superfluous but will never get cleaned up (cf. Lemma 3.45 and Example 3.4). On the other hand, there are non-redundant rules with respect to the definitions in Subsection 3.4.3 which will eventually get cleaned up (cf. Lemma 3.46).

3.3.2 Synchronisation Pattern

In this subsection we define a synchronisation pattern (i.e. a set of rules such that there is an effect if certain events occur concurrently, but not if they occur in different transitions). We also define an extended synchronisation pattern, which works nicely with the clean-up (cf. Definition 3.8).

In the following, we define a set of five rules which insert or delete an arbitrary rule r depending on op ($\text{op} = \triangleright$ would delete and $\text{op} = \blacktriangleright$ would insert) if x and y occur concurrently, but have no effect if x and y occur in different transitions.

Definition 3.12. Let $\text{op} \in \{\triangleright, \blacktriangleright\}$, let $x, y \in E$ with $x \neq y$, and let r be a HDES rule over E . Then we define the *synchronisation pattern* consisting of the following five rules

$$\{x, y\} \text{op} r, \quad (3.1)$$

$$x \triangleright [\{x, y\} \text{op} r], \quad (3.2)$$

$$y \triangleright [\{x, y\} \text{op} r], \quad (3.3)$$

$$\{x, y\} \triangleright [x \triangleright [\{x, y\} \text{op} r]], \quad (3.4)$$

$$\{x, y\} \triangleright [y \triangleright [\{x, y\} \text{op} r]]. \quad (3.5)$$

The set $\text{Sync}(\{x, y\} \text{op} [r])$ consists by definition of exactly the five Rules (3.1) to (3.5).

Example 3.2 (in the tool **THESIS Example 4**) is a small example HDES using the synchronisation pattern for a and b : If they occur synchronised they enable an event x ; if they occur in the order first a then b they enable the event y and if they occur in the reversed order, they enable the event z .

Now, before we prove the correctness of the synchronisation pattern we need a little further definition. We now define when a rule set R does not affect any rule in a rule set S .

Definition 3.13. Let R and S be two HDES rule sets over E . We say R does not affect S if there is now rule r or sub-rule r_i in R adding or dropping any rule s or sub-rule s_i of S .

```
1  @hdes
2
3  a(x=100, y=100)
4  b(x=300, y=100)
5  x(x=175, y=200)
6  y(x=75, y=200)
7  z(x=275, y=200)
8  x → x
9  y → y
10 z → z
11
12 @c "Synchronisation pattern for a AND b enabling x"
13 {a, b} ▷ [x → x]
14 a ▷ [{a, b} ▷ [x → x]]
15 b ▷ [{a, b} ▷ [x → x]]
16 {a, b} ▷ [a ▷ [{a, b} ▷ [x → x]]]
17 {a, b} ▷ [b ▷ [{a, b} ▷ [x → x]]]
18
19 @c "a adds the enabling of y AFTER b"
20 a ► [b▷ [y → y]]
21
22 @c "b adds the enabling of z AFTER a"
23 b ► [a ▷ [z → z]]
```

Example 3.2: HDES example for the synchronisation pattern as defined in Definition 3.12. Concurrently a and b enable x . In the order first a then b the enable y , and in the reversed order they enable z .

The next two lemmata state that the synchronisation pattern does what it was designed to do.

Lemma 3.14. *Let $H = (E, \text{Sync}(\{x, y\} \triangleright [r]) \cup R)$ such that*

- $r \in R$
- *and R does not affect the rules of the synchronisation pattern (as defined in Definitions 3.12 and 3.13),*
- *and r is not activated by x , y , or both of them.*

Then

- *the concurrent occurrence of x and y removes r from the current rule set of H ,*
- *but if only one occurs (or both but one after the other) then r remains in the current rule set of H*
- *if not removed by any other rule in R .*

Proof. Let H as defined in Lemma 3.14. Let us first assume without loss of generality x occurs but y does not occurs concurrently. Then, by Definition 3.4 only Rule (3.2) becomes active, and since it is independent it drops Rule (3.1). Thus, we have no rule in the synchronisation pattern anymore, which could drop r .

Let us now assume x and y occur concurrently. Then all five rules of the synchronisation pattern become active (cf. Definition 3.4), but only the Rules (3.4) and (3.5) are independent and so they drop the Rules (3.2) and (3.3). Thus, Rule (3.1) becomes independent and drops r as claimed. \square

Lemma 3.15. *Let $H = (E, \text{Sync}(\{x, y\} \blacktriangleright [\cup])R)$ such that R does not affect the rules of the synchronisation pattern (as defined in Definitions 3.12 and 3.13).*

Then,

- *the concurrent occurrence of x and y adds r to the current rule set of H ,*
- *but if only one occurs (or both but one after the other) then r is not added to the the current rule set of H*
- *if there were no other active rules in R adding r .*

Proof. Like in the proof of Lemma 3.14, this is an immediate consequence of Definition 3.4. \square

One possible downside of the above definition of the synchronisation pattern $\text{Sync}(\{x, y\} \text{op}[r])$ is that if only one event x or y occurs, there are rules of no further use, which cannot be removed by the clean-up (e.g. if only x occurs, consider the Rules (3.3) and (3.5). They have no further use but are not removed in a clean-up).

To circumvent this situation, we define the extended synchronisation pattern, which works nicely with the clean-up, but consists of nine instead of five rules.

Definition 3.16. Let $\text{op} \in \{\triangleright, \blacktriangleright\}$, let $x, y \in E$ with $x \neq y$, and let r be a HDES rule over E . Then we define the *extended synchronisation pattern* consisting of the following nine rules

$$\{x, y\} \text{op} r, \quad (3.6)$$

$$x \triangleright [\{x, y\} \text{op} r], \quad (3.7)$$

$$y \triangleright [\{x, y\} \text{op} r], \quad (3.8)$$

$$y \triangleright [x \triangleright [\{x, y\} \text{op} r]], \quad (3.9)$$

$$x \triangleright [y \triangleright [\{x, y\} \text{op} r]], \quad (3.10)$$

$$x \triangleright [y \triangleright [x \triangleright [\{x, y\} \text{op} r]]], \quad (3.11)$$

$$y \triangleright [x \triangleright [y \triangleright [\{x, y\} \text{op} r]]], \quad (3.12)$$

$$\{x, y\} \triangleright [x \triangleright [y \triangleright [x \triangleright [\{x, y\} \text{op} r]]]], \quad (3.13)$$

$$\{x, y\} \triangleright [y \triangleright [x \triangleright [y \triangleright [\{x, y\} \text{op} r]]]], \quad (3.14)$$

The set containing exactly the Rules (3.6) to (3.14) is defined as $\text{eSync}(\{x, y\} \text{op}[r])$.

Example 3.3 (in the tool **THESIS Example 5**) shows an example HDES using the extended synchronisation pattern for a and b : If they occur synchronised they enable an event x , if they occur in the order a then b they enable the event y and if they occur in the reversed order they enable the event z .

```

1  @hdes
2  a(x=100, y=100)
3  b(x=300, y=100)
4  x(x=175, y=200)
5  y(x=75, y=200)
6  z(x=275, y=200)
7  x → x
8  y → y
9  z → z
10
11 @c "Extended synchronisation pattern for a AND b enabling
    x"
12 {a, b} ▷ [x → x]
13 a ▷ [{a, b} ▷ [x → x]]
14 b ▷ [{a, b} ▷ [x → x]]
15 a ▷ [b ▷ [{a, b} ▷ [x → x]]]
16 b ▷ [a ▷ [{a, b} ▷ [x → x]]]
17 a ▷ [{b} ▷ [a ▷ [{a, b} ▷ [x → x]]]]
18 b ▷ [{a} ▷ [b ▷ [{a, b} ▷ [x → x]]]]
19 {a,b} ▷ [a ▷ [{b} ▷ [a ▷ [{a, b} ▷ [x → x]]]]]
20 {a,b} ▷ [b ▷ [{a} ▷ [b ▷ [{a, b} ▷ [x → x]]]]]
21
22 @c "a adds the enabling of y AFTER b"
23 a ▶ [b▷ [y → y]]
24
25 @c "b adds the enabling of z AFTER a"
26 b ▶ [a ▷ [z → z]]

```

Example 3.3: HDES example for the extended synchronisation pattern as defined in Definition 3.16

The next two lemmata state that the extended synchronisation pattern does what it was designed to do.

Lemma 3.17. *Let $H = (E, \text{eSync}(\{x, y\} \triangleright [r]) \cup R)$ such that*

- $r \in R$
- *and R does not affect the rules of the extended synchronisation pattern (as defined in Definitions 3.13 and 3.16),*
- *and r is not activated by x or y or both of them.*

Then,

- *the concurrent occurrence of x and y removes r from the current rule set of H ,*
- *but if only one occurs (or both but after each other) then r remains in the current rule set of H .*
- *And after any occurrence of x or y , a clean-up removes all remaining rules of the extended synchronisation pattern.*

Proof. Let H be as defined in Lemma 3.17. Let us first assume that, without loss of generality, x occurs but y does not occur concurrently. Then, by Definition 3.4 only the Rules (3.7), (3.10) and (3.11) becomes active, and since they are independent they drop the Rules (3.6), (3.8) and (3.9). Since x occurred, a clean-up with respect to x removes the Rules (3.12) to (3.14). Thus, we have no rule in the extended synchronisation pattern anymore, which could drop r .

Let us now assume x and y occur concurrently, then all nine rules of the extended synchronisation pattern become active (cf. Definition 3.4), but only the Rules (3.13) and (3.14) are independent, and so, they drop the Rules (3.11) and (3.12). Now, the two Rules (3.9) and (3.10) become independent and drop the two Rules (3.7) and (3.8). Thus, Rule (3.6) becomes independent and drops r as claimed. Since all rules of the extended synchronisation have become active, none is in the next rule set (in this case the clean-up is not necessary). \square

Lemma 3.18. *Let $H = (E, \text{eSync}(\{x, y\} \blacktriangleright [r]) \cup R)$ such that R does not affect the rules of the extended synchronisation pattern (as defined in Definitions 3.13 and 3.16).*

Then,

- *the concurrent occurrence of x and y adds r to the current rule set of H ,*
- *but if only one occurs (or both but after each other) then r is not added to the the current rule set of H .*
- *And after any occurrence of x or y , a clean-up removes all remaining rules of the extended synchronisation pattern.*

Proof. Like in the proof of Lemma 3.17, this is an immediate consequence of Definition 3.4. □

Both synchronisations have the same effect, but the extended one interacts nicely with the clean-up, and it might therefore be easier for some proofs. For purposes of modelling and understanding, on the other hand, the basic one is to be preferred, since it only consists of five instead of nine rules. Moreover, the maximal rank of any rule is two less than in the extended pattern +9.

Remark 3.19. Please note that those patterns can be easily lifted from two distinct events $x \neq y$ to two distinct sets $X \neq Y$ with $X, Y \subseteq E$.

3.4 Alternative Approaches

In this section we first present an alternative rule update strategy (in Subsection 3.4.1), second in Subsection 3.4.2 we present a representation of HDESs as transition systems and use this representation to show that both rule update strategies have the same expressive power. Finally, in Subsection 3.4.3, we present an approach to initially remove certain superfluous rules.

3.4.1 Alternative Rule Update Strategy

The approach chosen in Definition 3.4 above (to first evaluate the dropping rules and thereafter the adding rules) seems a little arbitrary. But, if we want to archive a deterministic rule update, which we do because we want to model deterministic real-world workflows, many approaches can be discarded (like take a random active rule). The key decision is whether to first execute the dropping or the adding rules. There are more complex strategies, e.g. an ordered list of rules, but then much simplicity in the models would be lost.

In the following, we define a different rule update strategy by first executing the adding rules and then the dropping rules. Please note that we also allow for

newly added rules to be dropped in the same transition; this is similar to the dropping of active adding rules in the upper approach).

Later, when studying the expressive power, we show that both approaches are equivalent (i.e. a HDES with one rule update strategy can be translated to one with the other one yielding the same transition sequences).

It is also straightforward to see that both synchronisation patterns (cf. Definitions 3.12 and 3.16) work as defined and proved above with respect to this alternative update strategy (even in a set-based variant).

Definition 3.20. Let $s_{\text{old}} = (\text{old}, R_{\text{old}})$ be a state of a HDES $H = (E, R)$ and $\text{new} \subseteq E$ with $\text{old} \subsetneq \text{new}$. Then

- the *rule update* (*AD*, *Adding Dropping*) R_{new} is the result of applying Algorithm 2 to s_{old} and the new set of events.
- We denote this relation as $s_{\text{old}} \xrightarrow{\text{AD}_{\text{new}}} R_{\text{new}}$.

The Algorithm 2 consists of three parts, like the Algorithm 1 above. However, firstly, the adding rules are considered (Lines 3 to 8) and executed (if active — Line 6) or copied to the new set (if passive — Line 8). Secondly (Lines 12 to 18) the dropping rules are considered, and step by step the independent rules are executed, (if active — Line 16) or copied to the new set (if passive — Line 18). Note that the independence of a rule might change since R_{old} gets modified by the algorithm. Finally, all causality rules (rules of the form $[c \rightarrow t]$) and remaining adding rules are copied to the new set (Line 21). We use $\text{ac}(s_{\text{old}}, \text{new})$ and $\text{ind}((\text{old}, R_{\text{old}}))$ as in Definition 3.4 for the set of active rules in a state and independent rules with respect to a configuration and rule set.

For this new rule update strategy, we also get a new transition relation based on it. The following definition is, except for the rule update, exactly the same as Definition 3.5.

Definition 3.21. Let $H = (E, R)$ be HDES, and $s_{\text{old}} = (\text{old}, R_{\text{old}})$ and $s_{\text{new}} = (\text{new}, R_{\text{new}})$ two of its states. Then $s_{\text{old}} \xrightarrow{\text{alt}_H} s_{\text{new}}$ if

- (1) $\text{old} \subsetneq \text{new}$
- (2) $\forall t \in \text{new} \setminus \text{old}. \forall c \in E. c \rightarrow_{R_{\text{old}}} t \implies c \in \text{old}$
- (3) $s_{\text{old}} \xrightarrow{\text{alt}_{\text{new}}} R_{\text{new}}$
- (4) $\forall r \in R_c. (r = M \blacktriangleright [c \rightarrow t] \wedge (r \in \text{ac}(s_{\text{old}}, \text{new}) \wedge t \in \text{new} \setminus \text{old}) \implies c \in \text{old}$

input : A HDES state $s_{\text{old}} = (\text{old}, R_{\text{old}})$ of a HDES $H = (E, R)$ and a set of events new , with $\text{old} \subsetneq \text{new} \subseteq E$

output: A set of HDES rules R_{new}

```

1  $R_{\text{new}} \leftarrow \emptyset;$  // The output set is constructed step by step.
2  $R' \leftarrow \emptyset;$  // This is an intermediate rule set, for dealing with adding
   rules.
3 foreach  $r = M \blacktriangleright [r'] \in R_{\text{old}}$  do // for each adding rule
4    $R_{\text{old}} \leftarrow R_{\text{old}} \setminus \{r\};$  // remove the rule from the old set
5   if  $r \in \text{ac}(s_{\text{old}}, \text{new})$  then // active adding rule?
6      $R' \leftarrow R' \cup \{r'\};$  // add the target to the intermediate set
7   else
8      $R' \leftarrow R' \cup \{r\};$  // add the rule to the intermediate set
9   end
10 end
11  $R_{\text{old}} \leftarrow R_{\text{old}} \cup R';$  // Insert the newly added and the passive adding rule
   to the old set such that the dropping rules can influence them.
12 while  $\exists r \in R_{\text{old}}, M \subseteq E. r = M \triangleright [r']$  do // while there are dropping rules
13   let  $r = M \triangleright [r'] \in R_{\text{old}} \wedge r \in \text{ind}(\text{old}, R_{\text{old}});$  // chose a independent
   dropping rule (wrt. the current  $R_{\text{old}}$ )
14    $R_{\text{old}} \leftarrow R_{\text{old}} \setminus \{r\};$  // remove the rule from the old set
15   if  $r \in \text{ac}(s_{\text{old}}, \text{new})$  then // active or passive rule?
16      $R_{\text{old}} \leftarrow R_{\text{old}} \setminus \{r'\};$  // drop the target from old set
17   else
18      $R_{\text{new}} \leftarrow R_{\text{new}} \cup \{r\};$  // copy the rule to the new set
19   end
20 end
21  $R_{\text{new}} \leftarrow R_{\text{new}} \cup R_{\text{old}};$  // copy the causality and adding rules to the new
   set
22 return  $R_{\text{new}};$ 

```

Algorithm 2: An alternative HDES rule set update algorithm

3.4.2 Event Transition Systems

In this subsection, we define Event Transition Systems (ETs), as a special kind of Labelled Transition System (LTS) additionally fulfilling certain further properties and equipped with one fixed event set E as superset for all state labels, meaning each label is a set of events.

Thereafter, we firstly show that we obtain such ETs for both rule update strategies of HDESs. Secondly, we show that for each ET and both rule update strategies a HDES can be constructed with exactly this ET as a semantic model.

Thus, we show that the rule update strategy does not change the expressive power of HDESs (at least for the two strategies considered above). We have further characterised the expressive power of HDESs by the ETs.

We start by giving a standard definition for LTSs where we consider labelled states and one explicit initial state. In contrast to Debois et al. in [8], we do not use asynchronous transition systems (since our transition might contain more than one event and thus the independence is encoded in the transition definition). We further do not label the transitions, but the states.

Definition 3.22. A *Labelled Transition System (LTS)* is a 5-tuple $L = (S, s_0, \rightarrow, \Lambda, \lambda)$ where

- S is the set of states,
- s_0 is the initial state,
- $\rightarrow \subseteq S \times S$ the transition relation
(with $s \rightarrow t$ indicating a transition from s to t),
- Λ is a set of state labels,
- and $\lambda : S \rightarrow \Lambda$ is a total function labelling the states.

In order to define the additional properties a LTSs must fulfil to be an ETs, we need further definitions. First, we define the the set of enabled event sets $\text{En}(s)$ for a state s of a LTSs with labels in the power set of an event set E .

Definition 3.23. Let E be a set of events, $L = (S, s_0, \rightarrow, \Lambda, \lambda)$ a LTS with

- $\Lambda = 2^E$
- and $s \in S$ with $\lambda(s) = X$.

Then the *set of enabled sets of events in the state s* is defined as

$$\text{En}_L(s) := \{D \subseteq E \setminus X \mid (D \neq \emptyset) \wedge \exists t \in S. (s \rightarrow t) \wedge (\lambda(t) = D \cup X)\},$$

if the LTS L is unambiguous we omit the subscript and write $\text{En}(s)$ instead.

Secondly, in the same setting as in the previous definition, we define the notion of parallelism $A \parallel_s B$ and conflict $A \nparallel_s B$, for two enabled event sets $A, B \in \text{En}(s)$ in a state s .

Definition 3.24. Let E be a set of events, $L = (S, s_0, \rightarrow, \Lambda, \lambda)$ a LTS with

- $\Lambda = 2^E$,
- $s \in S$ with $\lambda(s) = X$,
- $A, B \in \text{En}(s)$ with $A \cap B = \emptyset$.

Then

- the sets A and B are *concurrently enabled* in s , also called *enabled in parallel*, denoted as $A \parallel_s B$,
- if and only if there is a state t such that there is a transition from the state s to t with $\lambda(t) = X \cup A \cup B$.
- If A and B are not concurrently enabled in s we call them *conflicting*, denoted as $A \nparallel_s B$.

Now, we are ready to define the Event Transition Systems (ETS) as LTSs where $\Lambda = 2^E$ for an event set E and $\lambda(s_0) = \emptyset$ with additional five properties that are explained right below the definition.

Definition 3.25. An *Event Transition System (ETS)* \mathcal{E} with respect to an event set E is a LTS $L = (S, s_0, \rightarrow, \Lambda, \lambda)$ where

- $\Lambda = 2^E$,
- $\lambda(s_0) = \emptyset$,
- and $s \rightarrow t$, with $\lambda(s) = \text{old}$ and $\lambda(t) = \text{new}$ implies the following properties :

- (1) *Monotonicity*: $\text{old} \subseteq \text{new}$

- (2) *Progress*: $\text{old} \neq \text{new}$
- (3) *Downward closed enabling*:
 $\forall D \in \text{En}(s). \forall D' \subseteq D. D' \neq \emptyset \implies D' \in \text{En}(s)$
- (4) *Weak parallel consistency*:
 $\forall D \in \text{En}(s). \forall D' \subsetneq D. \forall t \in S.$
 $(s \mapsto t) \wedge (\lambda(t) = \text{old} \cup D') \implies \forall d \in D \setminus D'. \{d\} \in \text{En}(t)$
- (5) *Reachability*:
 $\forall s \in S. s_0 \mapsto^* s$, where \mapsto^* is reflexive and transitive closure of the transition relation \mapsto of \mathcal{E} .

Property (1) ensures that we do not lose any event, and Property (2) guarantees that new events occur. Property (3) ensures that each non empty subset D' of an enabled set D is also enabled. In other words, we do not have any forced parallelism where some events may only occur concurrently. Property (4) formulates the condition that the remainder of a set of concurrently enabled events D stays enabled if a non-empty proper subset D' of D occurs. Note, however, that they might not be concurrently enabled any longer. Finally, Property (5) ensures that all states of \mathcal{E} are reachable from the initial state s_0 .

In the following, we define a corresponding LTS for a HDES (independent of the rule update strategy).

Definition 3.26. Let $H = (E, R)$ be a HDES.

Then we define the corresponding LTS $L_H = (S_H, (\emptyset, R), \mapsto_H, 2^E, \pi_1)$ where

- S_H is the set of reachable states H (consisting of a configuration and a current rule set),
- \mapsto_H is the transition relation of H ,
- and π_1 is the projection to the configuration – the first component of a state.

Since only the new rules set is defined differently in the transition Definitions 3.5 and 3.21 of HDESs, Properties (1) to (3) of the Definition 3.25 are independent of the rule update strategy. Property (4) imposes a condition on the new state t after the transition $s \mapsto t$. In this case the rule update strategy could matter, but the prove shows it does not. Finally, since we only consider finite structures, it is sufficient to consider reachable states (Property (5)). But before starting the proof we show that an additional property always holds for ETSs.

Singleton-based conflicts make sure that any conflict between some sets A and B already exists between a subset X of $A \cup B$ and some event $x \in A \cup B$.

Definition 3.27. Let \mathcal{E} be a ETS as in Definition 3.25.

With *singleton-based conflicts* we denote the following property:

$$\forall A, B \in \text{En}(s). A \parallel_s B \implies \exists x \in A \cup B, X \subseteq A \cup B. \{x\} \parallel_s X$$

Lemma 3.28. Let \mathcal{E} be a ETS as in Definition 3.25.

Singleton-based conflicts as in 3.27 is always fulfilled.

Proof. By the definition of conflict for two sets A and B in Definition 3.24, we only demand $A \cup B$ not to be enabled concurrently. Thus, the singleton-based conflicts property is trivially fulfilled: Since A and B are non-empty (because they are enabled), we can just chose an arbitrary element in $A \cup B$ as x and $A \cup B \setminus \{x\}$ as X . Then we have $\{x\} \parallel_s X$, because $\{x\} \cup X = A \cup B$ is not enabled concurrently. \square

Lemma 3.29. Let $H = (E, R)$ be a HDES and $L_H = (S_H, (\emptyset, R), \mapsto_H, 2^E, \pi_1)$ its corresponding transition system as in Definition 3.26.

Then L_H is an ETS with respect to E .

Proof. Since $\Lambda = 2^E$ and $\lambda((\emptyset, R)) = \emptyset$, we have to show Properties (1) to (5) of Definition 3.25. Thus, let $s \rightarrow t$, with $\lambda(s) = \text{old}$ and $\lambda(t) = \text{new}$, Properties (1) and (2) are equivalent to Condition (1) of Definition 3.5. Conflicts are created by Condition (4) since this is the only concurrency-constraining condition of Definition 3.5, but since here all conflicts are singleton-based, as claimed in Lemma 3.28, we are done. If a set of events D is enabled in $(\text{old}, R_{\text{old}})$, then by Condition (2), all causality rules $e \rightarrow d$ for $d \in D$ are fulfilled by old (i.e. $e \in \text{old}$). Therefore, also each non empty subset of D is enabled and Property (3) is satisfied. To show Property (4), we have to show that subsets of enabled sets do not disable events from the enabled super set. In HDESs, the only way of disabling an event d is to add a not-yet happened precondition to d . Assume $D' \triangleright [x \rightarrow d] \in R_{\text{old}}$ and still is there when it gets evaluated, meaning there is no rule dropping it when first evaluating dropping rules, since we want to have the effect. Also, in the case of first evaluating adding rules, there is no rule dropping the effect, $[x \rightarrow d]$ since we want to have this effect with some $x \in E \setminus (\text{old} \cup D')$. With Condition (4) of Definition 3.5, it follows that there is no transition from s to a state t where $\lambda(t) = \text{old} \cup D' \cup \{d\}$. But, since D is enabled in s and so all non-empty subsets of D (by Property (3)), in particular $D' \cup \{d\}$ is enabled in s . This is a contradiction. Therefore, the assumption is false and Property (4) is satisfied. Finally, Property (5) is fulfilled since we only considered the reachable states of H

and thus, we have reachability also for L_H (since the transition relation of both structures coincide by Definition 3.26). \square

In order to define a corresponding rule set for a state in an ETS, we need further tools and notations.

Definition 3.30. Let $\mathcal{E} = (S, s_0, \mapsto, 2^E, \lambda)$ be an ETS. Let $s \in S$ be an arbitrary state of \mathcal{E} . Then

$$\text{maxEn}(s) := \{A \in \text{En}(s) \mid \nexists A' \in \text{En}(s) . A \subsetneq A'\}$$

is the set of inclusion-maximal enabled sets in the state s . With

$$\text{singleEn}(s) := \{e \in E \mid \{e\} \in \text{En}(s)\}$$

we denote the set of all events that are enabled in the state s . For a fixed set of enabled events $X \in \text{En}(S)$, we define

$$\text{conf}_X(s) := \{e \in \text{singleEn}(s) \mid \nexists A \in \text{maxEn}(s) . (e \in A) \wedge (X \subseteq A)\}$$

to be the set of all events that are in the state s conflicting with X . Let $\lambda(s)$ be the set of already occurred events. We define the local behaviour of a state as

$$\begin{aligned} \text{bh}(s) := & \{e \rightarrow e \mid e \in E \setminus (L \cup \text{singleEn}(s))\} \cup \\ & \{A \blacktriangleright [e \rightarrow e] \mid A \in \text{En}(s) \wedge e \in \text{conf}_A(s)\}. \end{aligned}$$

Definition 3.31. A rule context c is

- a function from a rule set R_E over an event set E to R_E ;
- a rule context c is either a hole $[\cdot]$,
- or recursively defined as $M \blacktriangleright [c']$ (respectively $M \triangleright [c']$) where $M \subseteq E$ and c' is a rule context.
- The application of a rule context c on a rule r , denoted as $c[r]$, is to fill the hole $[\cdot]$ in c with a rule r .

Let c be a rule context. Then

- $\text{mod}(c)$ denotes the outermost modifier set of c , that means if $c = M \text{ op } c'$ we have $\text{mod}(c) = M$, otherwise (if $c = [\cdot]$) $\text{mod}(c) = \emptyset$,

- $\text{modi}(c)$ further denotes *the set of all modifier sets of c* , defined as $\text{modi}(c) := \{M\} \cup \text{modi}(c')$ if $c = M \text{ op } c'$, and $\text{modi}(c) := \emptyset$ if $c = [\cdot]$.

Now, we are ready to define a corresponding HDES rule set for a state cur of an ETS \mathcal{E} . In Definition 3.32, we define the rule set $R_{\text{cur}}^{\text{da}}$ with respect to the original rule update definition, and in Definition 3.33 we define the rule set $R_{\text{cur}}^{\text{ad}}$ with respect to the alternative rule update definition.

The main idea of these rule sets is first to encode the local behaviour of the state cur and second to update the set with respect to all possible transitions (i.e. add recursively the behaviour of the successor state and prune all other branches).

Definition 3.32. Let $\mathcal{E} = (S, s_0, \mapsto, 2^E, \lambda)$ be an ETS and

- cur one of its states,
- $\{s_1, \dots, s_n\} = \{s \in S \mid \text{cur} \mapsto s\}$ the possible successor states of cur ,
- and $X_i = \lambda(s_i) \setminus \lambda(\text{cur})$ as in Figure 10 on Page 54.

The *rule set* $R_{\text{cur}}^{\text{da}}$ (the superscript indicates that this rule set is designed for the original rule update strategy (first dropping, then adding), it is formally defined as:

$$R_{\text{cur}}^{\text{da}} := \text{behave}(\text{cur}) \cup \text{upBh}(\text{cur}) \cup \text{addRec}(\text{cur}) \cup \text{prune}(\text{cur}) \quad (3.15)$$

Where the four *big rule sets* are defines as follows

$$\text{behave}(\text{cur}) := \text{bh}(\text{cur}), \quad (3.16)$$

$$\text{upBh}(\text{cur}) := \bigcup_{i=1}^n \text{upBh}(\text{cur}, s_i), \quad (3.17)$$

$$\text{addRec}(\text{cur}) := \bigcup_{i=1}^n \text{addRec}(\text{cur}, s_i), \quad (3.18)$$

$$\text{prune}(\text{cur}) := \bigcup_{i=1}^n \text{prune}(\text{cur}, s_i), \quad (3.19)$$

and the *branch rule sets* (parametrised by the branch s_i) are defined as follows

$$\begin{aligned} \text{upBh}(\text{cur}, s_i) &:= \{X_i \triangleright [r] \mid (r \in \text{bh}(\text{cur}) \setminus \text{bh}(s_i)) \wedge \\ &\quad ((r = X_i \blacktriangleright [e \rightarrow e]) \implies e \in \text{singleEn}(s_i))\}, \end{aligned} \quad (3.20)$$

$$\text{addRec}(\text{cur}, s_i) := \{X_i \blacktriangleright [r] \mid r \in R_{s_i}^{\text{da}}\}, \quad (3.21)$$

$$\begin{aligned} \text{prune}(\text{cur}, s_i) &:= \{\text{pc}[r] \mid r \in (\text{upBh}(\text{cur}, s_i) \cup \\ &\quad \text{addRec}(\text{cur}, s_i)), \text{pc}[\cdot] \in \text{pc}_{s_i}[\cdot]\}. \end{aligned} \quad (3.22)$$

The *pruning context* $\text{pc}_{s_i}[\cdot]$ for the branch s_i in the rule set (3.22) is defined as follows:

$$\begin{aligned} \text{pc}_{s_i}[\cdot]^1 &:= \{X_j \triangleright [\cdot] \mid (X_j \in \text{En}(\text{cur})) \wedge (X_j \neq X_i)\} \\ \text{pc}_{s_i}[\cdot]_c^1 &:= f_{\text{cur}}^*(\text{pc}_{s_i}[\cdot]^1) \\ \text{pc}_{s_i}[\cdot]^{k+1} &:= \{X_j \triangleright [c] \mid (X_j \in \text{En}(\text{cur})) \wedge (X_j \notin \text{modi}(c)) \wedge \\ &\quad (X_j \neq X_i) \wedge (c \in \text{pc}_{s_i}[\cdot]_c^k)\} \cup \text{pc}_{s_i}[\cdot]_c^k \\ \text{pc}_{s_i}[\cdot]_c^{k+1} &:= f_{\text{cur}}^*(\text{pc}_{s_i}[\cdot]^{k+1}) \\ \text{pc}_{s_i}[\cdot] &:= \text{pc}_{s_i}[\cdot]_c^{n-1} \end{aligned}$$

Possibly the above recursion reaches a fix point already before $n - 1$ steps, but by construction we are sure that a fix point is reached after $n - 1$ steps as each iteration adds a different X_j context to the dropping rules. There are only n different values for X_j , and one of them is X_i which has been excluded explicitly.

The *function* f_{cur} over rule context sets is defined as follows

$$f_{\text{cur}}(C) := \{X \triangleright [c] \mid c \in C \wedge X \in \text{En}(\text{cur}) \wedge \text{mod}(c) \subsetneq X\} \cup C,$$

and $f_{\text{cur}}^*(C)$ denotes the fix point of f_{cur} with respect to a rule context set C .

In the next definition, we define the corresponding rule set for the state cur of an ETS \mathcal{E} with respect to the alternative rule update strategy: first adding then dropping. Thus, it differs from the rules in Definition 3.32 since we now first add the rules of the state s_i , reached with the event set X_i , but we also add all rules of states reached with subsets of X_i . Therefore, we need extra rules to tidy up: $\text{rwa}(\text{cur}, s_i)$ (this stands for **r**emove **w**rongly **a**dded) removes rules which were accidentally added by proper subsets of X_i , or disablings added by X_i itself

which should not be in s_i , but only forbid concurrent occurrence of X_i and some e . The second point, the disablings added by X_i , allows for an easier version of the $\text{upBh}(\cdot)$ rules since we only need to drop those rule which belong to the old but not to the new behaviour. On the other hand, we have to ensure that we do not prune a rule of another branch X_j that was also added by X_i . Therefore we introduce the $\text{filter}(\cdot)$ function for pruning contexts and rules, to filter out exactly those cases. All other rules get adjusted to the new rule update strategy and the extended rule set.

Definition 3.33. Let $\mathcal{E} = (S, s_0, \mapsto, 2^E, \lambda)$ be a ETS and

- cur one of its states,
- $\{s_1, \dots, s_n\} = \{s \in S \mid \text{cur} \mapsto s\}$ the possible successor states of cur ,
- and $X_i = \lambda(s_i) \setminus \lambda(\text{cur})$ as in Figure 10 on Page 54.

We define the *rule set* $R_{\text{cur}}^{\text{ad}}$. The superscript indicates that this rule set is designed for the alternative rule update strategy (first adding, then dropping). Note that certain sets are now mutually recursive. That means $\text{rwa}(\text{cur}, s_i)$ will be affected by any $\text{prune}(\text{cur}, s_j)^{\text{ad}}$ with $X_j \subsetneq X_i$ and therefore $\text{prune}(\text{cur}, s_i)^{\text{ad}}$ is affected by $\text{prune}(\text{cur}, s_j)^{\text{ad}}$ with $(X_j \subsetneq X_i)$. As this is fortunately the only such case, we can start constructing those sets starting with the inclusion-minimal enabled sets and then proceed step by step, i.e. never consider a set if not all of its subsets were already considered.

So formally we define:

$$\begin{aligned}
 R_{\text{cur}}^{\text{ad}} := & \text{behave}(\text{cur}) \cup \text{upBh}(\text{cur})^{\text{ad}} \cup \\
 & \text{addRec}(\text{cur})^{\text{ad}} \cup \text{rwa}(\text{cur}) \cup \text{prune}(\text{cur})^{\text{ad}}
 \end{aligned} \tag{3.23}$$

Where the six *big rule sets* are defines as follows

$$\text{behave}(\text{cur}) := \text{bh}(\text{cur}), \quad (3.24)$$

$$\text{upBh}(\text{cur})^{\text{ad}} := \bigcup_{i=1}^n \text{upBh}(\text{cur}, s_i)^{\text{ad}}, \quad (3.25)$$

$$\text{addRec}(\text{cur})^{\text{ad}} := \bigcup_{i=1}^n \text{addRec}(\text{cur}, s_i)^{\text{ad}}, \quad (3.26)$$

$$\text{rwa}(\text{cur}) := \bigcup_{i=1}^n \text{rwa}(\text{cur}, s_i), \quad (3.27)$$

$$\text{prune}(\text{cur})^{\text{ad}} := \bigcup_{i=1}^n \text{prune}(\text{cur}, s_i)^{\text{ad}}, \quad (3.28)$$

and the *branch rule sets* (parametrised by the branch s_i) are defined as follows

$$\begin{aligned} \text{rwa}(\text{cur}, s_i) := \{X_i \triangleright [r] \mid r \in \text{rwaAuxI}(\text{cur}, s_i) \cup \\ \text{rwaAuxII}(\text{cur}, s_i)\}, \end{aligned} \quad (3.29)$$

$$\begin{aligned} \text{rwaAuxI}(\text{cur}, s_i) := \left\{ r \mid \exists r' \in R_{\text{cur}}^{\text{ad}}. (r' = X_j \blacktriangleright [r]) \wedge \right. \\ \left. (X_j \subsetneq X_i) \wedge (X_i \blacktriangleright [r] \notin R_{\text{cur}}^{\text{ad}}) \right\}, \end{aligned} \quad (3.30)$$

$$\begin{aligned} \text{rwaAuxII}(\text{cur}, s_i) := \{e \rightarrow e \mid X_i \blacktriangleright [e \rightarrow e] \in \text{bh}(\text{cur}) \wedge \\ e \in \text{singleEn}(s_i)\}, \end{aligned} \quad (3.31)$$

$$\text{upBh}(\text{cur}, s_i)^{\text{ad}} := \{X_i \triangleright [r] \mid r \in \text{bh}(\text{cur}) \setminus \text{bh}(s_i)\}, \quad (3.32)$$

$$\text{addRec}(\text{cur}, s_i)^{\text{ad}} := \{X_i \blacktriangleright [r] \mid r \in R_{s_i}^{\text{ad}}\}, \quad (3.33)$$

$$\begin{aligned} \text{prune}(\text{cur}, s_i)^{\text{ad}} := \{\text{pc}[r] \mid r \in \text{toBePruned}(\text{cur}, s_i) \wedge \\ \text{pc}[\cdot] \in \text{filter}(\text{pc}_{s_i}[\cdot], r)\}, \end{aligned} \quad (3.34)$$

$$\begin{aligned} \text{toBePruned}(\text{cur}, s_i) := \text{upBh}(\text{cur}, s_i)^{\text{ad}} \cup \text{addRec}(\text{cur}, s_i)^{\text{ad}} \cup \\ \text{rwa}(\text{cur}, s_i). \end{aligned} \quad (3.35)$$

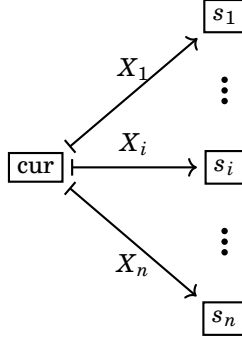


Figure 10: Sketch for Definitions 3.32 and 3.33.

The pruning context used in $\text{prune}(\text{cur}, s_i)^{\text{ad}}$ is the same as for the original rule update strategy and can be found in Definition 3.32 on Page 50. The function $\text{filter}(\text{pc}_{s_i}[\cdot], r)$ used in the same definition is defined as follows with $r = X_i \text{ op } \tilde{r}$:

$$\text{filter}(\text{pc}_{s_i}[\cdot], r) := \begin{cases} \text{pc}_{s_i}[\cdot] \setminus \{X_j \triangleright [\cdot]\}, & \text{if } X_j \blacktriangleright [r] \in \text{addRec}(\text{cur}, s_j) \\ & \text{and } X_i \cap X_j = \emptyset \\ \text{pc}_{s_i}[\cdot], & \text{otherwise} \end{cases} \quad (3.36)$$

We now show that both rule sets defined above yield the same initial local behaviour (i.e. a HDES with one of these rule sets as initial rule set, with respect to the correct event set, and the respective rule update strategy, initially allows for exactly the same transition as \mathcal{E}).

Lemma 3.34. *Let $\mathcal{E} = (S, s_0, \mapsto, 2^E, \lambda)$ be a ETS and*

- $\text{cur} \in S$,
- $E_{\text{cur}} := E \setminus \lambda(\text{cur})$,
- $R_{\text{cur}}^{\text{da}}$ and $R_{\text{cur}}^{\text{ad}}$ the corresponding rule sets for the the state cur of the ETS \mathcal{E} as defined in Definitions 3.32 and 3.33.

Let $H_1 := (E_{\text{cur}}, R_{\text{cur}}^{\text{da}})$ and $H_2 := (E_{\text{cur}}, R_{\text{cur}}^{\text{ad}})$ two HDESs with their corresponding ETSs \mathcal{E}_1 (with initial state s_0^1) and \mathcal{E}_2 (with initial state s_0^2) as defined in Definition 3.26. The following equality is satisfied:

$$\text{En}_{\mathcal{E}}(\text{cur}) = \text{En}_{\mathcal{E}_1}(s_0^1) = \text{En}_{\mathcal{E}_2}(s_0^2)$$

Proof. By the construction of $R_{\text{cur}}^{\text{da}}$ and $R_{\text{cur}}^{\text{ad}}$ in Definitions 3.32 and 3.33 and the HDES transition Definitions 3.5 and 3.21, the only rules that initially influence the enabled sets are in both cases $\text{bh}(\text{cur})$ (cf. Definition 3.30) and $\text{addRec}(\text{cur}, s_i)$ (cf. Rule (3.21)), if some e gets disabled in s_i , since then $[e \rightarrow e] \in \text{bh}(s_i) \subseteq R_{s_i}$ and therefore there is the initial rule $X_i \blacktriangleright [e \rightarrow e]$. By definition, events that are not enabled as singletons in the state cur of \mathcal{E} get initially disabled in H_1 and H_2 . Furthermore, if a set A is conflicting with an enabled event e in the state cur of \mathcal{E} , the concurrent occurrence in H_1 and H_2 gets blocked. Now, we have to consider the rules $X_i \blacktriangleright [e \rightarrow e]$, caused by $\text{addRec}(\text{cur}, s_i)$, they prevent the concurrent occurrence of X_i and e . Since \mathcal{E} is an ETS we have that, if in the state cur the set of events X_i and the event e are concurrently enabled, Property (4) ensures that e cannot get disabled in s_i . Thus, the rule $X_i \blacktriangleright [e \rightarrow e]$ did not block desired behaviour. \square

Before proving that the above construction indeed yields HDESs with the desired behaviour, meaning that not only initial behaviour but also the subsequent behaviour is the same as in \mathcal{E} , we first show that certain properties for the pruning contexts $\text{pc}_{s_i}[\cdot]$ are satisfied. We split the claim into two lemmata with respect to the respective rule update strategy. This makes the following proof shorter and easier to understand.

Lemma 3.35. *Let $\mathcal{E} = (S, s_0, \mapsto, 2^E, \lambda)$ be a ETS and*

- *cur one of its states,*
- *s_1, \dots, s_n its successor states reachable with the event set X_1, \dots, X_n ,*
- *and $R_{\text{cur}}^{\text{da}}$ be the corresponding rule set for the the state cur of \mathcal{E} as defined in Definition 3.32.*

Then for the HDES $H_1 := (E \setminus \lambda(\text{cur}), R_{\text{cur}}^{\text{da}})$ in the state $(\emptyset, R_{\text{cur}}^{\text{da}})$ the following properties hold, for a transition $(\emptyset, R_{\text{cur}}^{\text{da}}) \mapsto_H (X_i, R_{X_i})$:

- (1) *The set X_i activates no rule $r \in \text{prune}(\text{cur}, s_i)$, except of those, where X_i is the outermost modifier and was added by f_{cur} .*
- (2) *There are no dropping rules in $R_{\text{cur}}^{\text{da}} \setminus \text{prune}(\text{cur}, s_i)^{\text{da}}$ for any rule $r \in \text{prune}(\text{cur}, s_i)$.*
- (3) *In the clean-up of (X_i, R_{X_i}) , the set $\text{prune}(\text{cur}, s_i)$ vanishes.*

- (4) *The set X_i activates certain rules of $\text{prune}(\text{cur}, s_j)$ (for $X_i \neq X_j$) such that in the transition to the state (X_i, R_{X_i}) all rules plugged into $\text{pc}_{s_j}[\cdot]$ to obtain $\text{prune}(\text{cur}, s_j)$ get dropped and the remainder of the pruning context vanishes after a clean-up.*

Proof. By Lemma 3.34 there is a transition in H to any X_i , with $1 \leq i \leq n$. We prove the four properties one by one: For Property (1), by construction of the pruning context, we only used modifier sets $X_j \neq X_i$. Whenever there is a context starting with $X_j \subsetneq X_i$ (which would become active with respect to X_i), another context will be created by the function f_{cur} with X_i as outermost modifier, dropping the first context.

Property (2) follows immediately from Definition 3.32. By Properties (1) and (2), the rules in $\text{prune}(\text{cur}, s_i)$ are not dropped by X_i or executed meaningfully. That means the only rules that might get executed were added by f_{cur} and their effect is to drop certain other rules of $\text{prune}(\text{cur}, s_i)$. The targeted rules, that were plugged into $\text{pc}_{s_i}[\cdot]$ are not affected by execution of the pruning context. The plugged in rules are in $\text{addRec}(\text{cur}, s_i) \cup \text{upBh}(\text{cur}, s_i)$, and therefore, they are all starting with an X_j . Thus each rule in $\text{prune}(\text{cur}, s_i)$ get cleaned up after X_i .

In order to prove Property (4), we take a closer look at $\text{pc}_{s_j}[\cdot]$. By construction, X_i is the outermost modifier of some rule contexts in $\text{pc}_{s_j}[\cdot]_c^{n-1}$. These contexts become active and are independent, thus they drop all contexts in $\text{pc}_{s_j}[\cdot]_c^{n-2}$, except those starting with X_i . Now, by recursion we get that only those contexts in $\text{pc}_{s_j}[\cdot]_c^1$ that start with an X_i do not get dropped. Those rules are then active and independent and drop the rules, that were plugged into them as stated above. On the other hand, the rule contexts in $\text{pc}_{s_j}[\cdot]_c^n$ which do not have $X_i \subseteq X_j$ as outermost modifier are untouched by the rule update, but get removed in the clean-up since they have by construction X_i as an inner modifier. \square

The same as stated in Lemma 3.35 is also fulfilled for the alternative rule update strategy with the appropriate corresponding rule set $R_{\text{cur}}^{\text{ad}}$:

Lemma 3.36. *Let $\mathcal{E} = (S, s_0, \mapsto, 2^E, \lambda)$ be a ETS and*

- *cur one of its states,*
- *s_1, \dots, s_n its successor states reachable with the event set X_1, \dots, X_n ,*
- *and $R_{\text{cur}}^{\text{ad}}$ be the corresponding rule set for the the state cur of \mathcal{E} as defined in Definition 3.33.*

Then for the HDES $H_2 := (E \setminus \lambda(\text{cur}), R_{\text{cur}}^{\text{ad}})$ in the state $(\emptyset, R_{\text{cur}}^{\text{ad}})$ the following properties hold, for a transition $(\emptyset, R_{\text{cur}}^{\text{ad}}) \xrightarrow{\text{H}}^{\text{alt}} (X_i, R_{X_i})$:

- (1) The set X_i does not activate any rule $r \in \text{prune}(\text{cur}, s_i)^{\text{ad}}$, except of those where X_i is the outermost modifier and was added by f_{cur} .
- (2) For no rule $r \in \text{prune}(\text{cur}, s_i)^{\text{ad}}$ exists a dropping rule in the rule set $(R_{\text{cur}}^{\text{ad}} \cup \bigcup_{X_l \subseteq X_i} \text{addRec}(\text{cur}, s_l)^{\text{ad}}) \setminus \text{prune}(\text{cur}, s_i)^{\text{ad}}$,
- (3) In the clean-up of (X_i, R_{X_i}) the set $\text{prune}(\text{cur}, s_i)^{\text{ad}}$ vanishes.
- (4) The set X_i activates certain rules of $\text{prune}(\text{cur}, s_j)^{\text{ad}}$ (for $X_i \neq X_j$) such that after X_i all rules plugged into $\text{pc}_{s_j}[\cdot]$ to obtain $\text{prune}(\text{cur}, s_j)^{\text{ad}}$ get dropped (if they were not also added by X_i) and the remainder of the pruning context vanishes after a clean-up.

Proof. We again prove the four properties one by one: For Property (1), the proof is exactly the same as for Lemma 3.35. By construction of the pruning context, we only used modifier sets $X_j \neq X_i$. Whenever there is a context starting with $X_j \subsetneq X_i$ (which would become active with respect to X_i), there will an other context get created by the function f_{cur} with X_i as outermost modifier dropping the first context.

Property (2) follows immediately from Definition 3.33, as in the proof for Lemma 3.35.

By Properties (1) and (2), the rules in $\text{prune}(\text{cur}, s_i)^{\text{ad}}$ are neither dropped by X_i nor executed meaningfully. In other words, the only rules that might get executed were added by f_{cur} and their effect is to drop certain other rules of $\text{prune}(\text{cur}, s_i)^{\text{ad}}$. The targeted rules which were plugged into $\text{pc}_{s_i}[\cdot]$ are not affected by execution of the pruning context. The plugged-in rules are in $\text{addRec}(\text{cur}, s_i)^{\text{ad}} \cup \text{upBh}(\text{cur}, s_i)^{\text{ad}} \cup \text{urwa}(\text{cur}, s_i)$, and therefore they are all starting with an X_i . Thus, each rule in $\text{prune}(\text{cur}, s_i)^{\text{ad}}$ get cleaned up after X_i , and we are done proving Property (3).

In order to prove Property (4), we take a closer look at $\text{pc}_{s_j}[\cdot]$. By construction, X_i is the outermost modifier of some rule contexts in $\text{pc}_{s_j}[\cdot]_c^{n-1}$. These contexts become active and are independent, and thus they drop all contexts in $\text{pc}_{s_j}[\cdot]_c^{n-2}$, except those starting with X_i . Now, by recursion, we get that only those contexts in $\text{pc}_{s_j}[\cdot]_c^1$ which start with an X_i do not get dropped. Those rules are then active and independent and drop the rules that were plugged into them as stated above.

On the other hand, the rule contexts in $\text{pc}_{s_j}[\cdot]_c^n$ which do not have $X_l \subseteq X_i$ as outermost modifier are untouched by the rule update, but get removed in the clean-up since they have by construction X_i as an inner modifier. \square

Now, we can prove that the correspondence of a rule set is transition-invariant. If we start in a state of an ETS and compute the corresponding rule set, and perform a transition in the corresponding HDES, we get the same rule set after a cleaning up, as we get by making the corresponding transition in the original ETS and computing the corresponding rule set in the successor state afterwards.

Lemma 3.37. *Let $\mathcal{E} = (S, s_0, \mapsto, 2^E, \lambda)$ be an ETS and*

- *cur one of its states,*
- *s_1, \dots, s_n its successor states reachable with the event set X_1, \dots, X_n ,*
- *and $R_{\text{cur}}^{\text{da}}$ the corresponding rule set (with respect to the original transition definition) for the the state cur of \mathcal{E} as defined in Definition 3.32.*
- *The n different event sets X_1, \dots, X_n are initially enabled in the HDES $H_1 := (E \setminus \lambda(\text{cur}), R_{\text{cur}}^{\text{da}})$,*
- *and for all $1 \leq i \leq n$ we have $(\emptyset, R_{\text{cur}}^{\text{da}}) \mapsto_H (X_i, R_{X_i})$ such that the clean-up $\text{cu}(X_i, R_{X_i})$ yields the rule set $R'_{X_i} := \text{rs}(\text{cu}(X_i, R_{X_i}))$.*

Then, we have

- *$R'_{X_i} = R_{s_i}^{\text{da}}$ where the second rule set corresponds to the state s_i*
- *and H_1 in the cleaned-up state $(X_i, R_{s_i}^{\text{da}})$ allows for the same transition sequences as $\tilde{H}_1 := (E \setminus \lambda(s_i), R_{s_i}^{\text{da}})$ with the initial state $(\emptyset, R_{s_i}^{\text{da}})$.*

Proof. By Lemma 3.34, exactly X_1, \dots, X_n are the enabled sets. Let us now choose i without loss of generality. Since X_i is initially enabled in $H_1 := (E \setminus \lambda(\text{cur}), R_{\text{cur}}^{\text{da}})$, we now consider the transition $(\emptyset, R_{\text{cur}}^{\text{da}}) \mapsto_H (X_i, R_{X_i})$ and the clean-up $\text{cu}(X_i, R_{X_i})$ of this state with its rule set $R'_{X_i} := \text{rs}(\text{cu}(X_i, R_{X_i}))$. By Lemma 3.35 Property (4), it follows that all other branches are pruned. By Property (3) of the same Lemma, we have that $\text{prune}(\text{cur}, s_i)^{\text{da}}$ has no meaningful effect and is cleaned up. The only remaining rules are $\text{bh}(\text{cur})$, $\text{upBh}(\text{cur}, s_i)$, and $\text{addRec}(\text{cur}, s_i)$. Thus, we first execute the dropping rules $\text{upBh}(\text{cur}, s_i)$ to remove all undesired local behaviour

(but drop no rule in $\text{addRec}(\text{cur}, s_i)$). Thereafter, we add the remainder of the local behaviour for $\text{bh}(s_i)$ and all other rules in $R_{s_i}^{\text{da}}$ via $\text{addRec}(\text{cur}, s_i)$.

Since by construction no rule in $R_{s_i}^{\text{da}}$ contains any event in X_i , the second claim follows by Corollary 3.11. \square

Lemma 3.38. *Let $\mathcal{E} = (S, s_0, \mapsto, 2^E, \lambda)$ be an ETS and*

- *cur one of its states,*
- *s_1, \dots, s_n its successor states reachable with the event set X_1, \dots, X_n ,*
- *and $R_{\text{cur}}^{\text{ad}}$ the corresponding rule set (with respect to the alternative transition definition) for the the state cur of \mathcal{E} as defined in Definition 3.33.*
- *The n different event sets X_1, \dots, X_n are initially enabled in the HDES $H_2 := (E \setminus \lambda(\text{cur}), R_{\text{cur}}^{\text{ad}})$,*
- *and for all $1 \leq i \leq n$ we have $(\emptyset, R_{\text{cur}}^{\text{ad}}) \xrightarrow{\text{alt}}_{\text{H}} (X_i, R_{X_i})$ such that the clean-up $\text{cu}(X_i, R_{X_i})$ yields the rule set $R'_{X_i} := \text{rs}(\text{cu}(X_i, R_{X_i}))$.*

Then we have

- *$R'_{X_i} = R_{s_i}^{\text{ad}}$, where the right-hand side corresponds to the state s_i*
- *and H_2 in the cleaned-up state $(X_i, R_{s_i}^{\text{ad}})$ allows for the same transition sequences as $\tilde{H}_2 := (E \setminus \lambda(s_i), R_{s_i}^{\text{ad}})$ with the initial state $(\emptyset, R_{s_i}^{\text{ad}})$.*

Proof. By Lemma 3.34, exactly X_1, \dots, X_n are the enabled sets. Let us now choose i without loss of generality. Since X_i is initially enabled in $H_2 := (E \setminus \lambda(\text{cur}), R_{\text{cur}}^{\text{ad}})$, we now consider the transition $(\emptyset, R_{\text{cur}}^{\text{ad}}) \xrightarrow{\text{alt}}_{\text{H}} (X_i, R_{X_i})$ and the clean-up $\text{cu}(X_i, R_{X_i})$ of this state with its rule set $R'_{X_i} := \text{rs}(\text{cu}(X_i, R_{X_i}))$. Since we apply the alternative rule update strategy, we have to consider all adding rules first. Those can be divided into two groups: Rules belonging to $\text{bh}(\text{cur})$, designed to prevent the concurrent occurrence of a set A and a conflicting event e , and rules belonging to any $\text{addRec}(\text{cur}, s_l)^{\text{ad}}$ with $X_l \subseteq X_i$.

After classifying the adding rules, we consider the dropping rules. They are applied on a rule set with the adding rules already executed. First, we consider the rules in $\text{rwa}(\text{cur}, s_i)$: These drop any rule added by $\text{addRec}(\text{cur}, s_l)^{\text{ad}}$ with $X_l \subsetneq X_i$ which was not also added by another rule in $\text{addRec}(\text{cur}, s_i)^{\text{ad}}$. Next, there are dropping rules for $\text{rwa}(\text{cur}, s_i)$ except of those in $\text{prune}(\text{cur})^{\text{ad}}$, but those

are either not active if they belong to $\text{prune}(\text{cur}, s_l)^{\text{ad}}$ with $X_l \not\subseteq X_i$, or they get already dropped by construction in their own pruning context if they belong to $\text{prune}(\text{cur}, s_k)^{\text{ad}}$ with $X_k \subsetneq X_i$.

Furthermore, we consider $\text{upBh}(\text{cur}, s_i)^{\text{ad}}$. These drop the remainder of the old local behaviour in the state cur , and also the disabling that were spuriously added by rules $A \blacktriangleright [e \rightarrow e] \in \text{bh}(\text{cur})$ if e is enabled in s_i .

Finally, by Lemma 3.36 Property (4), all other branches are pruned. By Property (3) of the same Lemma, we have $\text{prune}(\text{cur}, s_i)^{\text{ad}}$ which has no meaningful effect and is cleaned up. To sum it up, we added all rules in $R_{s_i}^{\text{ad}}$ via $\text{addRec}(\text{cur}, s_i)^{\text{ad}}$, and removed thereafter all rules which do not belong into the new rule set.

The second claim follows with Corollary 3.11 since, by construction, no event in X_i occurs in any rule in $R_{s_i}^{\text{ad}}$. \square

The proof that a HDES with either rule update strategies exists for each ETS, such that the corresponding ETS is the same as the original ETS, is based on two steps: The first step uses Lemma 3.34 to show that the encoding has the correct behaviour locally. The second step is based on the two Lemmata 3.37 and 3.38 and states that the corresponding rule set and the transition commute (cf. Figure 11).

Theorem 3.39. *Let $\mathcal{E} = (S, s_0, \mapsto, 2^E, \lambda)$ be a ETS and*

- *$R_{s_0}^{\text{da}}$ and $R_{s_0}^{\text{ad}}$ be the corresponding rule sets for the the initial state s_0 of the ETS \mathcal{E} as defined in Definitions 3.32 and 3.33.*
- *Let $H_1 := (E, R_{s_0}^{\text{da}})$ and $H_2 := (E, R_{s_0}^{\text{ad}})$ two HDESs, \mathcal{E}_{H_1} and \mathcal{E}_{H_2} their corresponding transitions systems as in Definition 3.26.*

Then $\mathcal{E}, \mathcal{E}_{H_1}$ and \mathcal{E}_{H_2} are isomorphic, meaning that they only differ in the names of the states.

Proof. By Lemma 3.34, we initially have the same transitions and by Lemmata 3.37 and 3.38, we have the transition invariance of the corresponding rule set property for both rule update strategies: If we first compute the corresponding HDES for a state and then perform a transition in the HDES, we end up in a HDES state that is transition-equivalent to the initial state of the HDES obtained by first performing a transition in the ETS and then computing the corresponding HDES (cf. Figure 11). By induction, we are done since all states of \mathcal{E} are reachable by Property (5) of Definition 3.25. \square

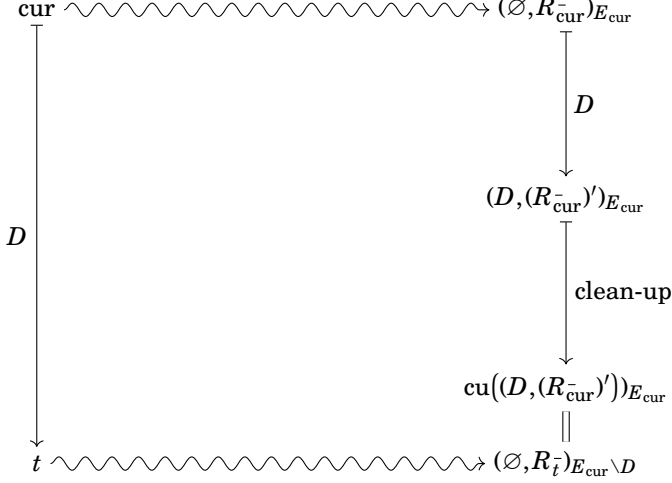


Figure 11: Sketch for the proof of Theorem 3.39. Here, D denotes an enabled set in the state cur of an ETS leading to the successor state t . Furthermore, the subscript E at a HDES state $(C, R_C)_E$ denotes the underlying event set. The underscore in R_{cur}^- and R_t^- is a wildcard for the respective rule update strategy. Obviously, it must be instantiated consistently.

Remark 3.40. The web tool can compute the rule sets $R_{\emptyset}^{\text{ad}}$ and $R_{\emptyset}^{\text{da}}$ of the corresponding ETS for the current HDES. The alternative rule update strategy can be selected with the keyword `@addThenDrop`. Note that our tool can only process very small HDESs with at most four events and not too many initially enabled ones – otherwise it has memory issues. For example, the simple HDES consisting of three events a, b and c and only one rule $a \blacktriangleright [b \rightarrow c]$ already explodes to 113 rules. In order to perform this translation, select the button “HDES Semantical Normalization” in the Translate menu.

3.4.3 Rule Set Reduction

Motivated by the huge rule set with many redundant rules obtained by the construction above, we define two criteria for rules to be redundant in this subsection. Redundancy in this context means that if the rules are removed from

any rule set, the behaviour of the HDES does not change. The first criterion is independent with respect to the rule update strategy. The second, however, depends on it which is why it is given separately for each strategy. Note that this reduction is state-independent in contrast to the clean-up in Subsection 3.3.1. It does not take any other rules in consideration.

Formally, redundancy is defined as follows:

Definition 3.41. A rule r is *redundant* for a specific update strategy if it could be removed from the rule set of any HDES H without changing the behaviour of H with respect to that strategy.

If r is redundant for both update strategies, we call it *redundant* without specifying a strategy.

Lemma 3.42 (Criterion 1). *Let $r = M_1 \text{op}_1[\dots[M_n \text{op}_n[c \rightarrow t]]\dots]$. If there is a $k \in \mathbb{N}^+$ with*

$$1 \leq k \leq n-1 \text{ such that } c \text{ or } t \text{ is contained in } M_k,$$

the rule r is redundant.

Proof. If c or t are contained in any M_i with $i < n$, then neither the dependency $c \rightarrow t$, nor its absence, nor a possible adding of the dependency $M_n \blacktriangleright [c \rightarrow t]$ has an effect (see Definitions 3.5 and 3.21). \square

The second criterion is different for the two rule update strategies: The first variant is for the original strategy (as in Definition 3.5), while the second variant is for the alternative strategy (as in Definition 3.21).

Lemma 3.43 (Criterion 2). *Let $r = M_1 \text{op}_1[\dots[M_n \text{op}_n[c \rightarrow t]]\dots]$. If there are $k, m \in \mathbb{N}^+$ with*

- $1 \leq k \leq n-1$ such that $\text{op}_k = \blacktriangleright$
- and $k < m \leq n$ such that $M_m \subseteq \bigcup_{i=1}^k M_i$,

the rule r is redundant for the original rule update strategy.

Proof. The possible effect of the rule r – after some steps such that M_k becomes the outermost modifier set – is to add via M_k the rule $\tilde{r} = M_{k+1} \text{op}_{k+1}[\dots[M_n \text{op}_n[c \rightarrow t]]\dots]$. However, this rule has no observable effect. Firstly, it can never add or drop its targeted dependency because the set M_m is already contained in $\bigcup_{i=1}^k M_i$

and never becomes active after M_k . Secondly, for $m = n$, we get $M_m \blacktriangleright [c \rightarrow t]$, which could restrict some concurrent occurrences of events, but since all events in M_m already happened, this restriction cannot be observed. Thus, r is redundant for the original rule update strategy. \square

Lemma 3.44 (Criterion 2 ad). *Let $r = M_1 \text{op}_1 [\dots [M_n \text{op}_n [c \rightarrow t]] \dots]$. If there is a $k, l, m \in \mathbb{N}^+$ with*

- $1 \leq k \leq n - 1$ such that $\text{op}_k = \blacktriangleright$,
- $k < l < n - 1$, with $\text{op}_l = \blacktriangleright$,
- and $l \leq m \leq n - 1$, such that $M_m \subseteq \bigcup_{i=1}^k M_i$,

the rule r is redundant for the alternative rule update strategy.

Proof. The proof is the same as in Lemma 3.43. \square

The idea of the above redundancy criteria is as follows: If, in a system run of a HDES with the redundant rule r , M_1, \dots, M_k become active step by step, the remainder of the rule is meaningless. The following Lemma states that if we apply a clean-up after a transition sequence such that M_1, \dots, M_k become active step by step, we end up in exactly the same state, with exactly the same rule set as if we initially removed r from the rule set, and thereafter do the same transition sequence with a final clean-up step. Please note that it is not always possible for the sets M_1, \dots, M_k to become active step by step since any M_i could contain impossible events.

Lemma 3.45. *Let $H = (E, R)$ be a HDES and*

- $r = M_1 \text{op}_1 [\dots M_n \text{op}_n [c \rightarrow t] \dots] \in R$ an redundant rule for any rule update strategy,
- M_k and k as in the Criteria for the redundancy of r ,
- $(\varnothing, R) =: s_1 \mapsto \dots \mapsto s_n := (X, R_x)$ be a transition sequence in H (with respect to the same strategy for that r is redundant),
- such that there is a subsequence s_{i_1}, \dots, s_{i_k} where in the transition to s_{i_j} the sub-rule r_j would be active,
- the sub-rule is defined as $r_j := M_j \text{op}_j [\dots M_m \text{op}_m [c \rightarrow t] \dots]$ (as in Definition 3.1),

- and let $\text{cu}(s_n) =: (X, R'_X)$ be the clean-up of s_n .

Then

- the HDESs H and $H' := (E, R \setminus \{r\})$ allow for the same transition sequences (with respect to the same strategy for that r is redundant),
- that means there is a transition sequence $(\emptyset, R \setminus \{r\}) := t_1 \mapsto \dots \mapsto t_n$ in H' ,
- such that for each $1 \leq i \leq n$ the equation $\text{con}(s_i) = \text{con}(t_i)$ is satisfied,
- and the clean-up of its final state $\text{cu}(t_n)$ is exactly $\text{cu}(s_n)$.

Proof. H' allows for the same transition sequence since r is redundant (cf. Definition 3.41).

Assume there is a rule $s \in \text{rs}(\text{cu}(t_n)) \setminus \text{rs}(\text{cu}(s_n))$, this means s was dropped by r , or some r_i with $i \leq k$. Therefore, it follows $s = r_j$ with $j \leq k + 1$, but then we have a contradiction to $s \in \text{rs}(\text{cu}(t_n))$ since the clean-up removes s .

Assume there is a rule $s \in \text{rs}(\text{cu}(s_n)) \setminus \text{rs}(\text{cu}(t_n))$, this means s was inserted (or its dropping deleted) by r , or some r_i with $i \leq k$. If s was inserted, it has the form $s = r_j$ with $j \leq k + 1$. On the other hand, if its dropping was deleted this was done by some r_i with $i < k$ (since $\text{op}_k = \blacktriangleright$). Now again, $s = r_j$ with $j \leq k + 1$, and thus, we have a contradiction to $s \in \text{rs}(\text{cu}(s_n))$ since the clean-up removes s . \square

There are some HDES with redundant rules that never get cleaned up **THEESIS Example 6:**

On the other hand, it is clear that there are rules which are not redundant but are cleaned up after a specific history. The following lemma states one example.

Lemma 3.46. *The rule $a \blacktriangleright [b \blacktriangleright [c \rightarrow c]]$ is cleaned up after $\{b\}$, but is not spotted by any redundancy criterion in this subsection.*

Proof. Immediate by the definitions. \square

3.5 On the Expressive Power of HDESs

In this section, we study the expressive power of HDESs with respect to other event and configuration structures. We also show that both generalisations, set-based modifiers and higher-order dynamics, from DCEs to HDESs are perpendicular with respect to their expressive power.

All proofs in this section only consider the original rule update strategy. If we have general results that do not consider specific levels of dynamicity, they are also fulfilled with the alternative rule update strategy (by Theorem 3.39 of Subsection 3.4.2). However, any result like Subsection 3.5.4 would need another proof with respect to the alternative rule update strategy.

3.5.1 Encoding DCEs in HDESs

We start this subsection with the expected result that HDESs are indeed a generalisation of DCEs, meaning that for each DCE Δ , there exists a transition-equivalent HDES H_Δ . Note that this is not completely obvious as the transition semantics differ: In a DCE, it is not possible for an adder and a dropper of the same dependency (which is still relevant) to occur concurrently (Definition 2.8 Condition (4)), whereas, in the HDES setting, this would be allowed and, depending on the rule update strategy (Definitions 3.4 and 3.20), would lead to an absent or present dependency.

Let us consider the DCE in Figure 7 on Page 17 (THESIS Example 7.1). There are adder a and dropper d for an initially absent dependency between cause c and target t .

In Example 3.5 (THESIS Example 7.2), we have a transition-equivalent HDES; we need a fresh impossible event, here called i_{ct} , since it belongs to a possible dependency between c and t , which is used to prevent concurrency between a and d using the rules $a \blacktriangleright [i_{ct} \rightarrow d]$ and $a \blacktriangleright [a \blacktriangleright [i_{ct} \rightarrow d]]$. Note that the blocking rule is dropped by c and t via the last two rules $c \blacktriangleright [a \blacktriangleright [i_{ct} \rightarrow d]]$ and $t \blacktriangleright [a \blacktriangleright [i_{ct} \rightarrow d]]$.

This approach is fully compositional and can be applied to each situation in a DCE where there is an adder and a dropper for the same dependency.

Definition 3.47. Let $\Delta = (E, \rightarrow, \triangleright, \blacktriangleright)$ be a DCE and

- $P := \{(c, t) \in E^2 \mid \exists d \in E. d \triangleright [c \rightarrow t] \wedge \exists a \in E. a \blacktriangleright [c \rightarrow t]\}$
the set of all pairs of events such that there is at least one adder and at least one dropper for the dependency $c \rightarrow t$ in Δ ,
- and $A_{c,t} := \{a \in E \mid a \blacktriangleright [c \rightarrow t]\}$ and $D_{c,t} := \{d \in E \mid d \triangleright [c \rightarrow t]\}$
for each $(c, t) \in P$ be the set of all adders (resp. droppers)
for a given dependency with at least one adder and at least one dropper.

Now, we define the HDES encoding of Δ as the HDES $H_\Delta = (E', R)$ where

```

1  @hdes
2  c(x=50, y=50)
3  b(x=150, y=250)
4  a(x=50, y=150)
5  t(x=250, y=50)
6
7  a ▶ [b ▶ [{a,b} ▶ [c → t]]]
8  b → b
9
10 @c "Example for a redundant rule that never gets cleaned
    up."

```

Example 3.4: A HDES with a redundant rule that never gets cleaned up.

```

1  @hdes
2  a(x=100, y=100)
3  d(x=300, y=100)
4  c(x=200, y=30)
5  t(x=200, y=170)
6  i_ct(x=200, y=240)
7
8  i_ct → i_ct
9  a ▶ [c → t]
10 d ▷ [c → t]
11
12 a ▶ [i_ct → d]
13 a ▷ [a ▶ [i_ct → d]]
14 c ▷ [a ▶ [i_ct → d]]
15 t ▷ [a ▶ [i_ct → d]]
16
17 @c "The transition-equivalent HDESs"
18 @c "for THESIS Example 7.1: Order-sensitive DCES."

```

Example 3.5: A transition-equivalent HDES encoding for the DCES in Figure 7.

- $E' = E \cup \{i_{ct} \mid (c, t) \in P\}$ with i_{ct} as fresh events and
- $R = \{c \rightarrow t \mid (c, t) \in \rightarrow\} \cup \{d \triangleright [c \rightarrow t] \mid (d, c, t) \in \triangleright\} \cup \{a \blacktriangleright [c \rightarrow t] \mid (a, c, t) \in \blacktriangleright\} \cup \bigcup_{(c,t) \in P} \{i_{c,t} \rightarrow i_{c,t}\} \cup \bigcup_{(c,t) \in P} R_{c,t}$
- and the rule sets $R_{c,t}$ are defined as

$$R_{c,t} := \{a \blacktriangleright [i_{c,t} \rightarrow d], x \triangleright [a \blacktriangleright [i_{c,t} \rightarrow d]] \mid a \in A_{c,t}, d \in D_{c,t}, x \in A_{c,t} \cup \{c, t\}\}.$$

Theorem 3.48. *Let $\Delta = (E, \rightarrow, \triangleright, \blacktriangleright)$ be a DCES and HDES $H_\Delta = (E', R)$ as defined in Definition 3.47.*

Then Δ and H_Δ are transition-equivalent.

Proof. The proof is straightforward by applying the respective transition Definitions 2.8 and 3.5. \square

Remark 3.49. We also implemented this encoding in the web tool. It can be applied with the button “DCES to HDES” in the Translate menu.

3.5.2 Orthogonal Extensions

In this subsection, we show that both dimensions of generalisation from DCESs to HDESSs, namely set-based dynamics and higher-order dynamics, are orthogonal (i.e. there are restricted HDESSs that use only one generalisation such that there is no restricted HDES which uses the other so that both are transition-equivalent). Two simple examples fulfilling these properties are depicted in Figure 12. They can also be found in the web tool (THESES Example 8 and THESES Example 9).

Lemma 3.50. *Let $H_{sb} = (\{a, b, c, t\}, \{\{a, b\} \blacktriangleright [c \rightarrow t]\})$ be a HDES.*

There is no HDES H_{ho} with only singletons as modifier sets such that $H_{sb} \simeq_{ts} H_{ho}$.

Proof. Assume there would be a HDES $H_{ho} = (E, R)$ with only singletons as modifier sets such that $H_{sb} \simeq_{ts} H_{ho}$. Since in H_{sb} the sets $\{a, t\}$ and $\{b, t\}$ are initially concurrently enabled, they also must be initially concurrently enabled in H_{sb} . Thus, the rule set R of H_{ho} cannot contain $a \blacktriangleright [c \rightarrow t]$ or $b \blacktriangleright [c \rightarrow t]$. But since a and b are concurrently enabled in H_{sb} and will add $c \rightarrow t$, we have a contradiction to the assumption $H_{sb} \simeq_{ts} H_{ho}$ since neither a nor b can add the dependency alone, nor can they together as in H_{ho} only singletons are allowed as modifiers. \square

Lemma 3.51. *Let $H_{ho} = (\{a, b, c, t\}, \{a \blacktriangleright [b \blacktriangleright [c \rightarrow t]]\})$ be a HDES.*

There is no HDES H_{sb} with at most first-order dynamics such that $H_{sb} \simeq_{ts} H_{ho}$.



Figure 12: Example HDES for Subsection 3.5.2. (1) a set-based adding. (2) a second-order adding.

Proof. Consider the transition sequence in the HDES H_{h_0} from the initial state (\emptyset, R_\emptyset) via $(\{a\}, R_{\{a\}})$ to $(\{a, b\}, R_{\{a, b\}})$. By Definition 3.4, we have $R_{\{a, b\}} = \{c \rightarrow t\}$ (but $c \rightarrow t \notin R_{\{a\}}$). Assume there is a HDES H_{sb} with at most first-order dynamics such that $H_{sb} \simeq_{ts} H_{h_0}$. By assumption, we have the transition sequence from the initial state (\emptyset, R_\emptyset) via $(\{a\}, R_{\{a\}})$ to $(\{a, b\}, R_{\{a, b\}})$ in H_{sb} where $c \rightarrow t \in R_{\{a, b\}}$ (but $c \rightarrow t \notin R_{\{a\}}$). By Definition 3.4, it follows that the dependency $c \rightarrow t$ was added by $b \blacktriangleright [c \rightarrow t]$ or $\{a, b\} \blacktriangleright [c \rightarrow t]$, which already were in R_\emptyset since we do not allow for higher-order dynamics in H_{sb} . Let us now consider the direct transition (\emptyset, R_\emptyset) to $(\{a, b\}, R_{\{a, b\}})$ in the HDES H_{h_0} (respectively to $(\{b\}, R_{\{b\}})$). Here we have $c \rightarrow t \notin R_{\{a, b\}}$ (respectively $c \rightarrow t \notin R_{\{b\}}$). However, since without loss of generality $b \blacktriangleright [c \rightarrow t] \in R_\emptyset$ of the HDES H_{sb} , the direct transition (\emptyset, R_\emptyset) to $(\{b\}, R_{\{b\}})$ in H_{sb} will result in $c \rightarrow t \in R_{\{a, b\}}$ (since possibly interfering dropping rules will be executed first, the causal dependency will be present). Thus, we have different observable behaviours (since t is enabled, after the initial transition (\emptyset, R_\emptyset) to $(\{b\}, R_{\{b\}})$, in H_{h_0} but not in H_{sb}). Therefore, we have $H_{sb} \not\simeq_{ts} H_{h_0}$. \square

3.5.3 From Configuration Structures to HDESs

In this section we present an encoding of RCES into a HDES respecting transition equivalence. This section is a reworked version of [16]. We show the result for the even more general class of configuration structures [11]. Here, we recap their definition:

Definition 3.52. Let E be a set.

- We call a pair (E, \mathcal{C}) with $\mathcal{C} \subseteq 2^E$ a *configuration structure*.

- For x, y in \mathcal{C} we write $x \rightarrow_{\mathcal{C}} y$ if

$$(x \subseteq y) \wedge (\forall Z. (x \subseteq Z \subseteq y \Rightarrow Z \in \mathcal{C})).$$

- The relation $\rightarrow_{\mathcal{C}}$ is called the *step-transition* relation.

For the proof, we need further notation.

Definition 3.53. Let $\tau = (E, \mathcal{C})$ be a configuration structure, $D \subseteq E$, and $F \subseteq E$. We denote with

- $\text{En}_D(\tau) \subseteq E \setminus D$ the set of events which are enabled in D ,
in a formal way $e \in \text{En}_D(\tau) \iff \exists D' \supseteq D. D \rightarrow_{\tau} D' \wedge e \in D' \setminus D$.
- $\text{Dis}_{\tau}(D) \subseteq E \setminus D'$ the set of disabled (or not enabled) events,
more formally $\text{Dis}_{\tau}(D) := D' \setminus (D \cup \text{En}_D(\tau))$.
- $\text{Con}_{\tau}(D, F) \subseteq E \setminus D$ the set of events which can be concurrent with F in D ,
in a formal way $e \in \text{Con}_{\tau}(D, F) \iff \exists D' \supseteq D. D \rightarrow_{\tau} D' \wedge F \cup \{e\} \subseteq D' \setminus D$.

Now, we can formulate and prove our result. The HDESs are strictly more expressive than any configuration structure. For any RCES, the transition graph is a configuration structure. The HDESs will be strictly more expressive than the RCESs. It was shown in [3] that there are DCESs whose behaviour cannot be simulated by any RCES. Now, we show that for each RCES there is a transition-equivalent HDES. Both results together (plus the above inclusion of DCESs in HDESs in Subsection 3.5.1) yield the strict inclusion.

Definition 3.54. Let $\tau = (E, \mathcal{C})$ be a configuration structure.

We define the *HDES encoding* of τ as $\text{HDES}(\tau) := (E, R)$:

- For each $e \in \text{Dis}_{\tau}(\emptyset)$, we add the rule $e \rightarrow e$ to R .
- Let $F \subseteq \text{En}_{\emptyset}(\tau)$ with $\emptyset \rightarrow_{\tau} F$
we add the rule $F \blacktriangleright [e \rightarrow e]$ for all $e \in (\text{En}_{\emptyset}(\tau) \setminus \text{Con}_{\tau}(\emptyset, F))$ to R .
- Let $C \subseteq E$ be a non-empty configuration of τ ,
we add the rule $C \blacktriangleright [e \rightarrow e]$ to R if $e \in \text{Dis}_{\tau}(C)$.
- If $e \in \text{En}_C(\tau)$, we add the rule $C \triangleright [e \rightarrow e]$.
- For each $D \subsetneq C$ and for each $e \in \text{Dis}_{\tau}(D)$,
we add the rule $C \triangleright [D \blacktriangleright [e \rightarrow e]]$ to R .

- For all $F \subseteq \text{En}_C(\tau)$ with $C \mapsto_\tau C', F \subseteq C'$ and for all $e \in (\text{En}_C(\tau) \setminus \text{Con}_\tau(C, F))$, we add the rule $C \blacktriangleright [F \blacktriangleright [e \rightarrow e]]$ to R .
- For all $D \subsetneq C$ and for all $A \subseteq \text{En}_D(\tau)$ with $D \mapsto_\tau D', A \subseteq D'$ and for all $e \in (\text{En}_D(\tau) \setminus \text{Con}_\tau(D, A))$, we add the rule $C \triangleright [D \blacktriangleright [A \blacktriangleright [e \rightarrow e]]]$ to R .

We have unique states for each configuration in this translation from configuration structures to HDES, meaning that for each reachable set of events there exists exactly one rule set and therefore one state.

Definition 3.55. Let τ be a configuration structure

- $\Delta = (E, R)$ its HDES encoding,
- and $C \neq \emptyset$ a non-empty configuration of τ .
- We call a rule set R_C the *corresponding rule set to C* if it can be obtained from R in the following way:

- (1) For any $D \subsetneq C$, remove any dropping rule $D \triangleright [r]$.
- (2) For any dropping rule $C \triangleright [r]$, remove the rule and its target r .
- (3) For any adding rule $C \blacktriangleright [r]$, remove the rule but add the target r .

Note that by this algorithm and the construction of R in the translation, each rule $D \text{ op } [r]$ with $D \subseteq C$ is dropped from R_C and each rule $E \text{ op } [r]$ with $E \not\subseteq C$ is copied to R_C . There are the causality rules $e \rightarrow e$ in R_C , for which $e \in \text{Dis}_\tau(C)$ and the concurrency restricting rules $F \blacktriangleright [e \rightarrow e]$ if $F \subseteq \text{En}_C(\tau)$ with $C \mapsto_\tau C', F \subseteq C'$, and $e \in (\text{En}_C(\tau) \setminus \text{Con}_\tau(C, F))$.

We now show that in each reachable state, the rule set corresponds to the configuration, and even for any transition starting in a state where the rule set corresponds to the configuration, this will hold in the resulting state.

We first show that, starting with the initial rule set, we reach only states with corresponding rule sets in one step.

Lemma 3.56. Let $\tau = (E, \mathcal{C})$ be a configuration structure,

- $H := \text{HDES}(\tau)$ its HDES-encoding,
- and $(\emptyset, R) \mapsto_C R_C$ be a rule update.

Then, R_C corresponds to C .

Proof. Since $(\emptyset, R) \rightarrow_C R_C$ is a rule update as in Definition 3.4, all active independent dropping rules are considered. These are all rules of the form $D \triangleright [r]$ for $D \subseteq C$. Note that by construction, r is no dropping rule. These dropping rules drop all former added or initial causality rules (e.g. $e \rightarrow e$) and concurrency-restricting rules (e.g. $F \blacktriangleright [e \rightarrow e]$) by construction. After the dropping rules, we consider the active adding rules. There are two types, causality-adding rules (e.g. $C \blacktriangleright [e \rightarrow e]$) and concurrency-restricting rules (e.g. $C \blacktriangleright [F \blacktriangleright [e \rightarrow e]]$). Finally, we copy the newly added causality rules to the new rule set. This is exactly the same as in Definition 3.55 because for each rule removed $D \triangleright [r]$ in step (Property (1)), there is a rule $C \triangleright [r]$ which will be executed in step (Property (2)). Thus, R_C corresponds to C . \square

We second show that, starting from a state with a corresponding rule set, we can reach only states with corresponding rule sets in one step. The proof is almost the same as above.

Lemma 3.57. *Let $\tau = (E, \mathcal{C})$ be a configuration structure,*

- $H := \text{HDES}(\tau)$ its HDES-encoding,
- and $(C, R_C) \rightarrow_{C'} R'_C$ be a rule update where R_C corresponds to C .

Then, we have R'_C corresponds to C' .

Proof. Since $(C, R_C) \rightarrow_{C'} R'_C$ is a rule update as in Definition 3.4, all active independent dropping-rules are considered. These are all rules of the form $D \triangleright [r]$ for $C' \subseteq D \subseteq C$ (no smaller modifier sets are possible because R_C is a corresponding rule set). Note that by construction, r is no dropping rule. These dropping rules drop all former added causality rules (e.g. $e \rightarrow e$) and concurrency-restricting rules (e.g. $F \blacktriangleright [e \rightarrow e]$). After the dropping rules we consider the active adding-rules. There are two types of adding rules, causality-adding rules (e.g. $C' \blacktriangleright [e \rightarrow e]$) and concurrency-restricting rules (e.g. $C' \blacktriangleright [F \blacktriangleright [e \rightarrow e]]$). Finally, we copy the newly added causality rules to the new rule set. This is exactly the same as in Definition 3.55 because for each removed rule $D \triangleright [r]$ in step (Property (1)), there is a rule $C \triangleright [r]$ which will be executed in step (Property (2)). Thus, R'_C corresponds to C' . \square

Lemma 3.58. *Let $\tau = (E, \mathcal{C})$ be a configuration structure,*

- $H := \text{HDES}(\tau)$ its *HDES-encoding*,
- and (C, R_C) and (D, R_D) two reachable states of H .

Then, we have the implication $(C = D) \implies (R_C = R_D)$.

Proof. Because both states are reachable, the claim follows from the previous two Lemmata 3.56 and 3.57 that both rule sets correspond to the configurations. If these configurations are the same, then clearly the corresponding rule sets are the same as well. \square

Since the rule sets are unique for each reachable configuration (they are the corresponding ones), the above lemma justifies to speak about configurations of H and transitions in between them. In order to compare H to the original configuration-structure τ , it is therefore sufficient to show that both are transition-equivalent.

Lemma 3.59. *Let $\tau = (E, \mathcal{C})$ be a configuration structure and let $H := \text{HDES}(\tau)$ be its HDES-encoding.*

Then, we have the equivalence $\emptyset \mapsto_{\tau} C \iff (\emptyset, R) \mapsto_H (C, R_C)$.

Proof. Let $\emptyset \mapsto_{\tau} C$. By Definition 3.53, it follows that $e \in \text{En}_{\emptyset}(\tau)$ for each $e \in C$ and that $C \subseteq \text{Con}_{\tau}(\emptyset, C)$. Thus, by construction in Definition 3.54, there is no rule $e \rightarrow e$ for any $e \in C$ in the rule set R . Furthermore, there is no rule $D \blacktriangleright [e \rightarrow e]$ for any $c \in C$ and $D \subseteq C$. We now show that all conditions of the transition Definition 3.5 hold. Condition (1) is clearly satisfied because $\emptyset \subseteq C$. The only initial causality rules are deactivating rules like $e \rightarrow e$ which do not occur for events in C by construction. Therefore, Condition (2) is satisfied. Next, Condition (3) constrains only the rule update and is fulfilled by assumption. Regarding the last Condition (4), let us first assume there is a rule $M \blacktriangleright [e \rightarrow e]$ such that $M \subseteq C$ and $e \in C \setminus M$. The event e is either disabled in the configuration M or the event e is initially not allowed to be concurrent with M (by construction of the rule set). However, since we have $\emptyset \mapsto_{\tau} C$ and this implies $\emptyset \mapsto_{\tau} M \cup \{e\}$ and $M \mapsto_{\tau} M \cup \{e\}$ because τ is a configuration structure, we have that no rule $M \blacktriangleright [e \rightarrow e]$ with $M \subseteq C$ and $e \in C \setminus M$ is in R . Therefore, the property is fulfilled. We have shown that all conditions of the transition Definition 3.5 hold. Thus, we have $(\emptyset, R) \mapsto_H (C, R_C)$.

To show the equivalence, we assume $\emptyset \not\mapsto_{\tau} C$. Then, by Definitions 2.4 and 3.53, there are $D \subsetneq C$ and $e \in C \setminus D$ such that $e \notin \text{Con}_{\tau}(\emptyset, D)$. By construction (cf.

Definition 3.54), there is a rule $D \blacktriangleright [e \rightarrow e]$ in R . Thus, by Condition (4) of the transition Definition 3.5, it follows that $\emptyset \not\vdash_H C$. \square

Lemma 3.60. *Let $\tau = (E, \mathcal{C})$ be a configuration structure,*

- $H := \text{HDES}(\tau)$ its HDES-encoding,
- $C \in \mathcal{C}$ a configuration,
- and R_C the rule set corresponding to C .

Then, we have the equivalence $C \rightarrow_\tau C' \iff (C, R_C) \rightarrow_H (C', R_{C'})$.

Proof. Let $C \rightarrow_\tau C'$. By Definition 3.53, it follows that $e \in \text{En}_C(\tau)$ for each $e \in C'$ and $C' \subseteq \text{Con}_\tau(C, C')$. Thus, by construction of R in Definition 3.54 and because of R_C being a rule set corresponding to C by assumption, there is no rule $e \rightarrow e$ for any $e \in C'$ in the rule set R_C . Furthermore, there is no rule $D \blacktriangleright [e \rightarrow e]$ for any $e \in C'$ and $D \subseteq C'$. We now show that all conditions of the transition Definition 3.5 hold: Condition (1) is clearly satisfied because $C \subseteq C'$. The only causality rules are by construction of R and because R_C is corresponding to C deactivating rules like $e \rightarrow e$, which do not occur for events in C' (because they are enabled). Therefore, Condition (2) holds. Next, Condition (3) constrains only the rule update and is fulfilled by assumption. Regarding the last Condition (4), let us first assume there is a rule $M \blacktriangleright [e \rightarrow e]$ such that $M \subseteq C'$ and $e \in C' \setminus M$. Then the event e is either disabled in the configuration M or the event e may not be concurrent with M in C (by construction of the rule set). However, since we have $C \rightarrow_\tau C'$ and this implies $C \rightarrow_\tau M \cup \{e\}$ and $M \rightarrow_\tau M \cup \{e\}$ because τ is a configuration structure, we have that no rule $M \blacktriangleright [e \rightarrow e]$ with $M \subseteq C'$ and $e \in C' \setminus M$ is in R_C , and therefore, the property holds. We have shown that all conditions of the transition Definition 3.5 are satisfied. Thus, we have $(C, R_C) \rightarrow_H (C', R_{C'})$.

To show the equivalence, we assume $C \not\vdash_\tau C'$. Then there are, by Definitions 2.4 and 3.53, $D \subsetneq C'$ and $e \in C' \setminus D$ such that $e \notin \text{Con}_\tau(C, D)$. By construction (cf. Definition 3.54), there is a rule $D \blacktriangleright [e \rightarrow e]$ in R_C . Thus, by Condition (4) of the transition Definition 3.5 it follows that $(C, R_C) \not\vdash_H (C', R_{C'})$. \square

Theorem 3.61. *Let $\tau = (E, \mathcal{C})$ be a configuration structure.*

Then, we have τ and its HDES-encoding $H := \text{HDES}(\tau)$ are transition-equivalent.

Proof. By induction with Lemmata 3.59 and 3.60. \square

3.5.4 Expressive Power with Respect to the Level of Dynamics

In the previous Subsection 3.5.3 we have shown that third-order dynamics is sufficient to model RCESs. Now, we will show that first-order dynamics is not enough. Thus, we show that there is a gain in expressive power from first-order dynamics to third-order dynamics. Actually we show that the gap is already between first- and second-order dynamicity. We will present an RCES ρ_H (see Definition 3.62) which cannot be modelled with first- but with second-order dynamics. It can be described as follows: An event a disables another event d but this disabling can be resolved (also pre-emptively) by the occurrence of the event b or the event c .

Definition 3.62. Let $\rho_H = (\{a, b, c, d\}, \vdash)$ be the RCES where

$$X \vdash Y \iff (X = \emptyset \wedge \{a, d\} \not\subseteq Y) \vee (X \in \{\{b\}, \{c\}, \{d\}\} \wedge \{a, d\} \subseteq Y).$$

It is straightforward to see that ρ_H has the behaviour claimed above: The event a disables the event d but this is resolved by b or c . If there is a transition in ρ_H to a configuration containing a and d , then the previous configuration must contain b , c , or d (by Definition 2.4).

Lemma 3.63. *There is no HDES H_1 with rules of at most rank one, meaning causality rules and first-order adding or dropping rules, such that H_1 and ρ_H from the above Definition 3.62 are transition-equivalent.*

Proof. Assume there is a HDES $H_1 = (E, R)$ with rules of rank at most one such that H_1 and ρ_H are transition-equivalent. Since in ρ_δ we have $\emptyset \rightarrow_{rc} \{a\} \not\vdash_{rc} \{a, d\}$ but $\emptyset \rightarrow_{rc} d$, there must be some $x \in E \setminus \{a\}$ such that $a \blacktriangleright [x \rightarrow d]$ since adding a cause that did not happen is the only way to disable an event in HDESs. Consider the trace $\emptyset \rightarrow_{rc} \{b\} \rightarrow_{rc} \{a, b\} \rightarrow_{rc} \{a, b, d\}$ in ρ_H . In order to have the same behaviour in H_1 , it follows that the above rule $a \blacktriangleright [x \rightarrow d]$ must be instantiated with $x = b$ since otherwise d would be disabled in H_1 after $\emptyset \rightarrow_H \{b\} \rightarrow_H \{a, b\}$. Completely analogously, it follows that $x = c$ if we consider the trace $\emptyset \rightarrow_{rc} \{c\} \rightarrow_{rc} \{a, c\} \rightarrow_{rc} \{a, c, d\}$, so we have a contradiction. Thus, there is no HDES with at most first-order dynamics that is transition-equivalent to ρ_δ . \square

If we combine the above Lemma 3.63 with the Theorem 3.61 of the previous Subsection 3.5.3, we get the following Corollary:

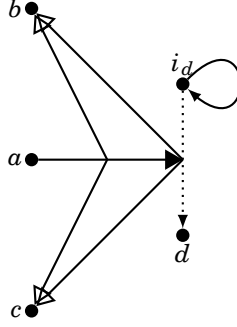


Figure 13: A transition-equivalent HDES H_{ρ_H} for ρ_H (as in Definition 3.62).

Corollary 3.64. *There is a gain in the expressive power of HDESs if we consider third-order dynamics instead of first-order dynamics.*

Proof. Since there is an RCES that cannot be simulated in HDESs with at most first-order dynamics (Lemma 3.63) but for each RCES there is a transition-equivalent HDES with third-order dynamics (Theorem 3.61). It follows that HDES with at most third-order dynamics are strictly more expressive than HDES with at most first-order dynamics. \square

For the RCES ρ_H defined above (as in Definition 3.62) there is a HDES with at most second-order dynamics which is transition-equivalent (cf. Figure 13 and THESIS Example 10).

Lemma 3.65. *The HDES H_{ρ_H} , as in Figure 13 (THESIS Example 10), is transition-equivalent to the RCES ρ_H as in Definition 3.62.*

Proof. The event a disables d by adding the impossible i_d as a precondition for d . However, the events b (or c) resolve this disabling by dropping the possible dependency $i_d \rightarrow d$ and the adding capability of a for this dependency. Note that d can obviously precede a but they are not allowed concurrently if they are not preceded by b or c (as defined in transition Definition 3.5 of HDESs). \square

The previous Lemma 3.65 allows for a more precise version of Corollary 3.64. There is even a gap in expressive power between HDES with at most first-order dynamics in comparison to HDES with at most second-order dynamics.

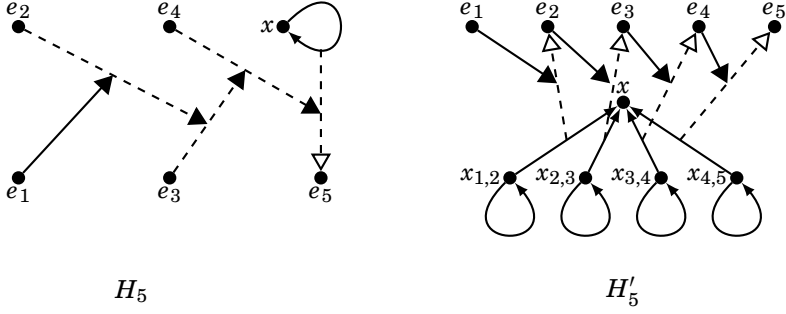


Figure 14: The HDESs H_5 and H'_5 (as in Definitions 3.68 and 3.69).

Lemma 3.66. *There is a gain in the expressive power of HDESs if we consider second-order dynamics instead of first-order dynamics.*

Proof. The proof is immediate by the Lemmata 3.63 and 3.65. □

Now, there is the general question: Does the expressive power of HDES increase with the order of dynamicity (like in Lemma 3.66)? This is still an open problem but we assume it holds for arbitrarily high orders.

Conjecture 3.67. *For $n \in \mathbb{N}^+$,*

- *there is a HDES H_n with n -th-order dynamics*
- *such that there is no transition-equivalent HDES H'_n with at most $(n - 1)$ -th-order dynamics.*

The conjecture is obviously satisfied for $n = 1$. By the above Lemma 3.66, it also is satisfied for $n = 2$. In the following, we will present two HDESs (parametrised by their order of dynamicity) we disproved to fulfil the conjecture.

The first one consists of $n + 1$ events e_1, \dots, e_n and an event x disabled initially. Additionally, we have the rule $e_1 \blacktriangleright [\dots e_{n_1} \blacktriangleright [e_n \triangleright [x \rightarrow x]] \dots]$. In Figure 14 and [THESIS Example 11.1](#), you can find H_5 .

Definition 3.68. Let $n \in \mathbb{N}^+$ and $H_n = (E_n, R_n)$ be the HDES with

- $E_n = \{e_1, \dots, e_n, x\}$
- and $R_n = \{[x \rightarrow x], e_1 \blacktriangleright [\dots e_{n-1} \blacktriangleright [e_n \triangleright [x \rightarrow x]] \dots]]\}$.

For any $n \in \mathbb{N}^+$ this H_n can be collapsed to a second-order structure H'_n which uses additional impossible events where any two consecutive e_i, e_{i+1} drop one impossible cause $x_{i,i+1}$ of the event x . In Figure 14 and THESIS Example 11.2, you can find H'_5 .

Definition 3.69. Let $n \in \mathbb{N}^+$,

- $H_n = (E_n, R_n)$ as in Definition 3.68
- and $H'_n = (E'_n, R'_n)$ be the HDES with
- $E'_n = E_n \cup \{x_{1,2}, \dots, x_{n-1,n}\}$
- and $R'_n = \{[x_{i,i+1} \rightarrow x_{i,i+1}], [x_{i,i+1} \rightarrow x_{i+1}] \mid 1 \leq i \leq n-1\} \cup \{e_i \blacktriangleright [e_{i+1} \triangleright [x_{i,i+1} \rightarrow e_{i+1}]] \mid 1 \leq i \leq n-1\}$.

Lemma 3.70. For any $n \in \mathbb{N}^+$ the HDESs H_n and H'_n , as defined in Definitions 3.68 and 3.69, are transition-sequence-equivalent.

Proof. In both structures, x only becomes enabled if and only if $e_1 \dots e_n$ occur sequentially. There are no further behavioural restrictions for $e_1 \dots e_n$ in either structure. In H'_n , there are additional impossible events. However, since there are no enablers for them, the state transition-equivalence of H_n and H'_n is fulfilled. \square

The reduction of the order of dynamicity from H_n to H'_n is based on the idea of partially enabling, meaning that more and more impossible causes $x_{i,i+1}$ of x are dropped until finally x becomes enabled. Thus, a natural idea for the next structure G_n is to disable an event x after the sequential occurrence of e_1, \dots, e_n since it is not clear, what a partial disabling would look like. In Figure 15 and THESIS Example 12.1, you can find G_5 .

Definition 3.71. Let $n \in \mathbb{N}^+$ and $G_n = (E_n, R_n)$ be the HDES with

- $E_n = \{e_1, \dots, e_n, x\}$
- and $R_n = \{e_1 \blacktriangleright [\dots e_{n-1} \blacktriangleright [e_n \blacktriangleright [x \rightarrow x]] \dots]]\}$.

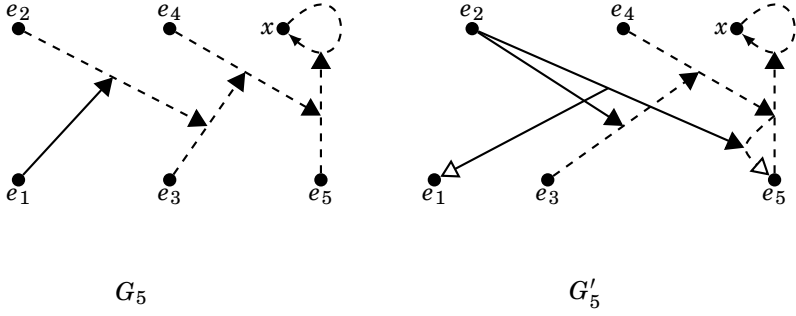


Figure 15: The HDESs G_5 and G'_5 (as in Definitions 3.71 and 3.72).

This G_n can be collapsed for any $n \geq 5$ to a $(n - 1)$ -th-order structure G'_n . The main idea of this reduction is that the potential end effect of e_1 is clear from the beginning: If thereafter e_2, \dots, e_n occur sequentially, x becomes disabled. Therefore, we can model the chain starting with e_2 and add a rule such that e_2 prevents the final effect $e_2 \blacktriangleright [e_n \blacktriangleright [e_n \blacktriangleright [x \rightarrow x]]]$ which is dropped by e_1 , reinstalling the order e_1, \dots, e_n . In Figure 15 and [THESIS Example 12.2](#), you can find G'_5 .

Definition 3.72. Let $n \in \mathbb{N}^+$,

- $G_n = (E_n, R_n)$ as in Definition 3.71
- and $G'_n = (E_n, R'_n)$ be the HDES with
- $R'_n = \{e_2 \blacktriangleright [\dots e_{n-1} \blacktriangleright [e_n \blacktriangleright [x \rightarrow x]] \dots], e_2 \blacktriangleright [e_n \blacktriangleright [e_n \blacktriangleright [x \rightarrow x]]]\} \cup \{e_1 \blacktriangleright [e_2 \blacktriangleright [e_n \blacktriangleright [e_n \blacktriangleright [x \rightarrow x]]]]\}$.

Lemma 3.73. For any $n \in \mathbb{N}^+$ the HDESs G_n and G'_n – as in Definitions 3.71 and 3.72 – are transition-sequence-equivalent.

Proof. In G_n the event x becomes disabled only after the sequential occurrence of e_1, \dots, e_n . Otherwise, there is no restriction on the behaviour. However, exactly the same is true for G'_n . Here, the sequence e_2, \dots, e_n disables x but e_2 creates a delayed trigger ($e_2 \blacktriangleright [e_n \blacktriangleright [e_n \blacktriangleright [x \rightarrow x]]]$) to remove this disabling. The creation of this trigger is removed by e_1 . Since this is a rule of order four any G_n with $n \geq 5$

can be reduced to G'_n with one order of dynamicity less. Thus, the state-transition equivalence of G_n and G'_n is satisfied. \square

As mentioned above, this reduction is based on the fact that it is clear what the potential effect of e_1 might be. The structures considered above are way too static to fulfil the conjecture. In Figure 16 and in [THESIS Example 13](#), you can find the HDES D_2 , a highly dynamic structure (as defined in Definition 3.74): There is a potential dependency between a cause c and a target t which is initially absent but can be added by an adder a and be dropped by a dropper d . This first-order dynamics might also be dropped by the dropper da and dd and reinstalled by the responding adders aa and ad . The structure could easily be generalised to arbitrarily high orders.

Definition 3.74. We first define some list comprehension notation like in Haskell and second a parametrised class of HDESs.

- Let $\{a, d\}^n := \{x \mid x \in \{a, d\}^* \wedge 1 \leq |x| \leq n\}$ be the set of all non-empty strings of length at most n with respect to the alphabet $\{a, d\}$ and
- $s \in \{a, d\}^n$ be a non-empty string.
- We denote with $s = x : xs$ that the first character of s is x and the remaining part is xs .
- Of course xs might be empty (denoted as ε).
- Now, we can define a recursive function which maps strings over $\{a, d\}^n$ to rules over $\{a, d\}^n \cup \{c, t\}$.

$$\begin{aligned}
 f : \{a, d\}^n &\longrightarrow \text{RuleSets}(\{a, d\}^n \cup \{c, t\}) \\
 f(\varepsilon) &\longmapsto c \rightarrow t \\
 f(a : as) &\longrightarrow a : as \blacktriangleright [f(as)] \\
 f(d : ds) &\longmapsto d : ds \triangleright [f(ds)]
 \end{aligned}$$

Let $n \in \mathbb{N}^+$ and

- $D_n = (E_n, R_n)$ be a HDES with
- $E_n := \{a, d\}^n \cup \{c, t\}$
- and $R_n := \{f(e) \mid e \in E_n\}$.

You can find D_2 in Figure 16 and in THESIS Example 13.

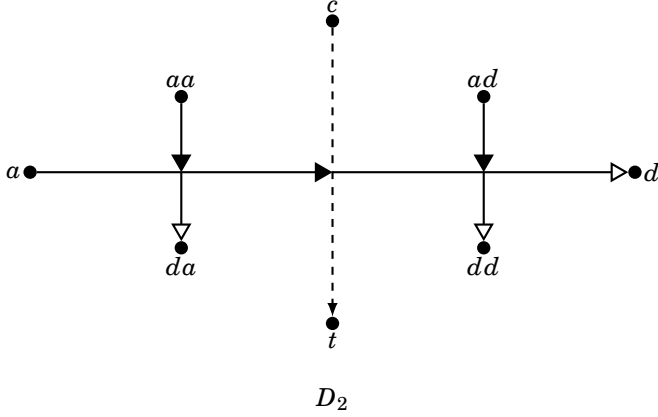


Figure 16: The HDESs D_2 (as in Definition 3.74).

We assume that the HDES D_n defined above (Definition 3.74) indeed fulfils the hierarchy Conjecture 3.67 (i.e. there is no HDES D'_n with order of dynamicity less than n such that D_n and D'_n are transition-equivalent).

Conjecture 3.75. *For any $n \in \mathbb{N}^+$ the HDES D_n as defined in Definition 3.74 fulfils the Conjecture 3.67.*

4 Encoding of DCR-Graphs

4.1 Concurrency Semantics

In this section we recap the basic definitions from [15] and [8] in order to compare Dynamic Condition Response Graphs (DCR-Graphs) to HDESSs. DCR-Graphs are like HDESSs a generalisation of Prime Event Structures, but allow for repeated event execution and dynamic exclusion and inclusion of events. As in our dynamic generalisations, a state with more information than a configuration is needed. Here, markings are used. A marking denotes for each event if it was executed at least once, if it is a pending response (needed for an accepting run), and if it currently is included. First we focus on the definitions in [8] for defining a concurrency semantics for basic DCR-Graphs (we omit the labels from the DCR-Graph). Later we discuss certain more elaborate features of DCR-Graphs (cf. [15]).

Definition 4.1 ([8]). A *Dynamic Condition Response Graph (DCR-Graph)* is a 3-tuple $G = (E, M, R)$ where

- E is the set of events (or activities),
- $M = (\text{Ex}, \text{Re}, \text{In}) \in \mathcal{M}(G)$ is the marking for $\mathcal{M}(G) = 2^E \times 2^E \times 2^E$ where Ex is the set of executed, Re the set of restless, and In the set of included events,
- $R = (-\rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%)$ are the condition, response, include, and exclude relation respectively, with each relation is defined on E^2 .

An event is enabled if and only if it is included and all its conditions were previously executed or are currently excluded.

Definition 4.2 ([8]). For an event e of a DCR-Graph G ,

- we say that e is *enabled*, written $G \vdash e$ if and only if $(e \in \text{In}) \wedge ((\text{In} \cap -\rightarrow\bullet e) \subseteq \text{Ex})$.

We define next the effect of the execution of an event, meaning which event is executed (Δe), which events are being included (ΔI), which events are being excluded (ΔX), and which events are being made pending (ΔR).

Definition 4.3 ([8]). The *effect of the execution of an event e on a DCR-Graph G* is given by $\text{EFFECT}(G, e) = (\Delta e, \Delta I, \Delta X, \Delta R)$ where:

- $\Delta e = \{e\}$ the singleton set containing the event being executed,
- $\Delta I = e \longrightarrow +$ the events being included by e ,
- $\Delta X = e \longrightarrow \%$ the events being excluded by e ,
- $\Delta R = e \bullet \longrightarrow$ the events being made pending by e .

When the G is clear from the context, we write δ_e for $\text{EFFECT}(G, e)$. As a next step, we define how the effect is applied to the DCR-Graph.

Definition 4.4 ([8]). The *application of the effect $\delta_e = (\Delta e, \Delta I, \Delta X, \Delta R)$ on a marking $(\text{Ex}, \text{Re}, \text{In})$* is defined as follows:

$$\delta_e \cdot (\text{Ex}, \text{Re}, \text{In}) = (\text{Ex} \cup \Delta e, (\text{Re} \setminus \Delta e) \cup \Delta R, (\text{In} \setminus \Delta X) \cup \Delta I)$$

The *effect δ_e applied on the DCR-Graph $G = (E, M, R)$* is defined as:

$$\delta_e \cdot (E, M, R) = (E, \delta_e \cdot M, R)$$

Note that if an event e both includes and excludes an event f , the event f remains included. On the other hand, if an event e is a response to itself, it is a pending response after its execution.

Now, we can define how these operations are used to execute an event on a DCR-Graph, yielding a new one.

Definition 4.5 ([8]). For a DCR-Graph G and

- an event e , such that $G \vdash e$,
- we define the *result of executing e* as $G \oplus e = \text{EFFECT}(G, e) \cdot G$.

In order to speak about acceptance of DCR-Graphs, we first define the obligations of a DCR-Graph to be the set of its included and pending events.

Definition 4.6 ([8]). Given a DCR-Graph $G = (E, M, R)$ with

- marking $M = (\text{Ex}, \text{Re}, \text{In})$,
- we define *the obligations of G* to be $\text{OBL}(G) = \text{Re} \cap \text{In}$.

Finally, we have all we need to define the notion of finite and infinite executions and when they are accepting:

Definition 4.7 ([8]). For a DCR-Graph G an *execution of G* is

- a (finite or infinite) sequence of 3-tuples $(G_i, e_i, G'_i)_{i \leq k}$ (for $k \in \mathbb{N} \cup \{\omega\}$)
- each comprising a DCR-Graph, an event and another DCR-Graph such that $G = G_0$ and, for $i < k$, we have $G'_i = G_{i+1}$;
- moreover for, $i \leq k$, we have $G_i \vdash e_i$ and $G'_i = G_i \oplus e_i$.
- We say the execution is *accepting* if, for $i \leq k$, and for all $e \in \text{OBL}(G_i)$ there is a $j \geq i$ with either $e_j = e$ or $e \notin \text{OBL}(G'_j)$.
- With $\text{EXE}(G)$ (resp. $\text{ACC}(G)$) we denote *the set of all executions (resp. all accepting executions) of G* .
- Finally, we say that a DCR-Graph G' is *reachable* from G if and only if there exists a finite execution of G ending in G' .

Up to now, we have only dealt with interleaving semantics. We proceed by introducing a notion of concurrency.

Definition 4.8 ([8]). We call two distinct events $e \neq f$ of a DCR-Graph G *effect orthogonal* if and only if

- (1) no event included by e is excluded by f and vice versa, and
- (2) e requires a response from g if and only if (f does or $f \neq g$).

We lift this notion to effects themselves, saying δ_e, δ_f of G are orthogonal if and only if e, f are.

Lemma 4.9 ([8]). Let δ_e, δ_f be effects of a DCR-Graph G , and

- let M be a marking for G .

Then the orthogonality of e, f implies the commutativity of δ_f and δ_e :

$$\delta_e \cdot (\delta_f \cdot M) = \delta_f \cdot (\delta_e \cdot M)$$

Definition 4.10 ([8]). We call two distinct events e, f of a DCR-Graph G *cause-orthogonal* if and only if

- (1) neither event is a condition for the other,
- (2) neither event includes or excludes the other, and
- (3) neither event includes or excludes a condition for the other.

Definition 4.11 ([8]). Given a DCR-Graph G , we say that

- events e, f are *independent* if they are both effect- and cause-orthogonal.
- We write I_G for the *independence relation* induced by a DCR-Graph G .

Definition 4.12 ([8]). An *Asynchronous Transition System (ATS)* is

- a 5-tuple $A = (S, s_0, E, \rightarrow, I)$
- comprising a set of *states* S ,
- an *initial state* $s_0 \in S$,
- a set of *events* E ,
- a *transition relation* on the events $\rightarrow \subseteq S \times E \times S$,
- and an irreflexive symmetric *independence relation* I satisfying

- (1) $s \xrightarrow{e} s'$ and $s \xrightarrow{e} s''$ implies $s' = s''$,
- (2) $s \xrightarrow{e} s', s' \xrightarrow{e'} s''$, and eIe' implies $\exists s'''$ such $s \xrightarrow{e'} s'''$, and $s''' \xrightarrow{e} s''$,
- (3) $s \xrightarrow{e} s', s \xrightarrow{e'} s''$, and eIe' implies $\exists s'''$ such $s' \xrightarrow{e'} s'''$, and $s'' \xrightarrow{e} s'''$.

Theorem 4.13 ([8]). *Let G be a DCR-Graph. Then $\mathcal{A}(G)$ is an ATS when equipped with the independence relation I_G .*

Now we define a labelled version of HDESs with an acceptance criterion. For the next definition, we assume that A is no event of the underlying HDES. Otherwise we can apply α -conversation.

The original HDES as in Definition 3.1 get extended with the acceptance event A , a set of event labels L , a labelling function $l : E \rightarrow L$, and a possibly non empty initial configuration C_0 .

Definition 4.14. A *labelled HDES* is a 5-tuple $(E \cup \{A\}, R, l, L, C_0)$ consisting of

- a HDES (E, R) ,
- an additional event $A \notin E$ for acceptance simulation,
- a *labelling function* $l : E \rightarrow L$ mapping events to labels in L ,
- and $C_0 \subseteq E$ is the initial configuration (if omitted assumed to be empty).

A state s of a HDES is called *accepting* if and only if the special event A is enabled in s .

Definition 4.15. Let $H = (E, R, l, L, C_0)$ be a labelled HDES, then we call a state $s = (C, R_C)$ *accepting* if and only if A is enabled in s .

We further extend the DCR-Graphs by adding a counter for each event to the marking. This counter has no behavioural impact.

Definition 4.16. An *extended DCR-Graph* $G' = (E, M', R)$ is

- a regular DCR-Graph $G = (E, M, R)$ with a fourth set in the marking,
- that means $M' = M \times \text{Count}$,
- where Count consists of tuples (e, n) for each $e \in E$
the second parameter $n \in \mathbb{N}$ counts how often e occurs.

Since Count should be updated upon execution of events, we have to slightly modify Definition 4.4 by incrementing the counter of the event e :

Definition 4.17. The *application of the effect* $\delta_e = (\Delta e, \Delta I, \Delta X, \Delta R)$ on a marking $(\text{Ex}, \text{Re}, \text{In}, \text{Count})$ with $(e, n) \in \text{Count}$ is defined as follows:

$$\delta_e \cdot (\text{Ex}, \text{Re}, \text{In}, \text{Count}) = (\text{Ex} \cup \Delta e, (\text{Re} \setminus \Delta e) \cup \Delta R, (\text{In} \setminus \Delta X) \cup \Delta I, \text{Count}')$$

where in the upper equation the set Count' is defined as

$$\text{Count}' = (\text{Count} \setminus \{(e, n)\}) \cup \{(e, n + 1)\}.$$

The *effect* δ_e applied on the DCR-Graph $G = (E, M, R)$ is defined as:

$$\delta_e \cdot (E, M, R) = (E, \delta_e \cdot M, R)$$

Note that this extensions only tracks additional information in the markings which can be used in proofs, but does not influence the behaviour of the DCR-Graph.

4.2 Encoding of DCR-Graphs into HDESs

In Subsection 4.2.1 we start by giving the general idea of the encoding. Thereafter, we explain each rule that we might create (depending on the relations and the marking of the DCR-Graph) step by step. And finally, we have the formal definition for the encoding (see Definition 4.18). In Subsection 4.2.2 we show the formal correctness of our encoding.

In our web tool, we implemented the encoding. In [THESIS Example 14](#) a simple DCR-Graph is defined ($e \bullet \rightarrow f$, $x \rightarrow \% f$, and $n \rightarrow + f$) and can be translated (by selecting DCR to HDES in the translate menu) into a HDES with at most three executions for each event (this can be easily adapted with a suitable n in the parameter `@dcrDepth "n"`). Of course, any other basic DCR-Graph can be entered and translated in the same way.

4.2.1 Explanation and Definition of the Encoding

Before starting, we want to point out that in [8] it is possible for an event to exclude and include the very same other event with the effect that the target is afterwards included. Therefore, we can assume without loss of generality that there is no case like this since this behaviour can be modelled with the inclusion only.

Let $G = (E, M, R)$ be an extended DCR-Graph. We define a labelled HDES $H(G) = (E_G, R_G, l, E, C_G^0)$ – with the initial state is (C_G^0, R_G) – such that both are transition-equivalent.

For each event $e \in E$ we have an infinite chain e_i (for each $i \in \mathbb{N}^+$) of events in E_G (with $e_i \rightarrow e_{i+1}$ for each $i \in \mathbb{N}^+$) to be able to repeat e .

Additionally, there is for each $e \in E$ a blocking event $b^e \in E$ (which is disabled by the rule $b^e \rightarrow b^e$) for the dynamic exclusion of DCR-Graphs.

Finally, there is an event $A \in E_G$ which is used to encode the acceptance criterion of DCR-Graphs into the HDES framework. The idea behind this is to encode pending responses as precondition for A , and defining that $H(G)$ is in an accepting state if and only if there is a possible transition by observing A .

Thus, we have $E_G := \{e_i \mid i \in \mathbb{N}^+ \wedge e \in E\} \cup \{b^e \mid e \in E\} \cup \{A\}$ the labelling function l maps every event to the original one in the DCR-Graph, meaning that $l : E_G \setminus \{A\} \rightarrow E$ with $e_k \mapsto e$ and $b^e \mapsto e$ (this second one only for technical reasons since no b^e will ever occur in any state).

We first discuss the rules induced by the four DCR-Graph relations $R = (\rightarrow \bullet, \bullet \rightarrow, \rightarrow +, \rightarrow \%)$ of G (however, for certain rules it is important to check

if certain events are currently included). Second, we take the marking M of G into account to define the proper initial configuration C_G^0 of $H(G)$ (in order to encode the executed events). Certain additional rules are needed to encode initial excluded or restless events.

Now, we explain all rules we create in the encoding step by step. Later in Definition 4.18 they are listed with the according preconditions.

Since the condition relation is the same as the causal dependency in HDESSs, each $e \rightarrow \bullet f$ translates to $e_1 \rightarrow f_i$ (for all $i \in \mathbb{N}^+$) (Rule (4.3)), but of course, only if e is initially included. By the definition of cause-orthogonality (see Definition 4.10), a condition and its target are not allowed to be concurrent. We have the rules $e_i \blacktriangleright [f_j \rightarrow f_j]$ (for all $i, j \in \mathbb{N}^+$) and $e_i \triangleright [e_i \blacktriangleright [f_j \rightarrow f_j]]$ (for all $i, j \in \mathbb{N}^+$) (Rules (4.1) and (4.2)).

For the exclusion and inclusion relation, we first focus on the interplay with the condition relation. Consider an exclusion $x \rightarrow \% e$ with the additional condition $e \rightarrow \bullet f$. From the above modelling of the condition relation, we have $e_1 \rightarrow f_i$ (for all $i \in \mathbb{N}^+$). If e gets excluded, this condition is not taken into account. Thus, we need a disabling of e and a temporary deletion of the dependency. Therefore, we have $x_i \blacktriangleright [b^e \rightarrow e]$ (for all $i \in \mathbb{N}^+$). This Rule (4.4) models the disabling of e since b^e is permanently disabled. We have $x_i \triangleright [e_1 \rightarrow f_j]$ (for all $i, j \in \mathbb{N}^+$) (Rule (4.5)), deleting the dependency between e and f . By Definition 4.10 of cause-orthogonality, an excluder and any condition are not allowed to be concurrent with the target of the condition. So we have the rules $x_i \blacktriangleright [f_j \rightarrow f_j]$ (for all $i, j \in \mathbb{N}^+$) and $x_i \triangleright [x_i \blacktriangleright [f_j \rightarrow f_j]]$ (for all $i, j \in \mathbb{N}^+$) (Rules (4.6) and (4.7)).

On the other hand, if $n \rightarrow + e$ with some $e \rightarrow \bullet f$, we have $e_1 \rightarrow f_i$ (for all $i \in \mathbb{N}^+$) from the above modelling of the condition relation. We have to drop any dependencies from e to certain blocking events and re-include the dependencies modelling the condition relation where e was a condition. Thus, we have $n_i \triangleright [b^e \rightarrow e]$ (for all $i \in \mathbb{N}^+$) (Rule (4.8), enabling of e) and $n_i \blacktriangleright [e_1 \rightarrow f_j]$ (for all $i, j \in \mathbb{N}^+$) (Rule (4.11), restoration of the dependency between e and f). Furthermore, due to Definition 4.10 of cause-orthogonality, the includer and its target are not allowed to be concurrent (the same is fulfilled for the excluder but this is already enforced as a side effect of the Rule (4.4)). We have the rules $n_i \blacktriangleright [e_j \rightarrow e_j]$ (for all $i, j \in \mathbb{N}^+$) and $n_i \triangleright [n_i \blacktriangleright [e_j \rightarrow e_j]]$ (for all $i, j \in \mathbb{N}^+$) (Rules (4.9) and (4.10)). Also due to the same definition if we have some f with $e \rightarrow \bullet f$ like above, f and n are not allowed to be concurrent. We model it as always with the rules $n_i \blacktriangleright [f_j \rightarrow f_j]$ (for all $i, j \in \mathbb{N}^+$) and $n_i \triangleright [n_i \blacktriangleright [f_j \rightarrow f_j]]$ (for all $i, j \in \mathbb{N}^+$) (Rules (4.12) and (4.13)). If we have $x \rightarrow \% e$ and $n \rightarrow + e$, then by Definition 4.8, both x and n are not effect-orthogonal, and thus, not independent, and therefore, never concurrent.

To model this we use rules of the form $x_i \blacktriangleright [n_j \rightarrow n_j]$ (for all $i, j \in \mathbb{N}^+$) and $x_i \triangleright [x_i \blacktriangleright [n_j \rightarrow n_j]]$ (for all $i, j \in \mathbb{N}^+$) (Rules (4.14) and (4.15)).

The basic idea to model the response relation $e \bullet \rightarrow f$, is to temporarily block the acceptance event A by rules of the form $e_i \blacktriangleright [b^f \rightarrow A]$ (for all $i \in \mathbb{N}^+$) (Rule (4.16)) and remove this blocker with the unblocking rules $f_i \triangleright [b^f \rightarrow A]$ (for all $i \in \mathbb{N}^+$) if f occurs (Rule (4.19)) and f is not requiring a response from itself (i.e. $f \rightarrow \bullet f \notin R$). However, this becomes more sophisticated since f might be dynamically excluded (and re-included and so on). Note that the Rule (4.16) are obsolete if f is initially excluded. With respect to Definition 4.8, if there is no $f \bullet \rightarrow f$, we have to omit the concurrent occurrence of e and f . We add the usual rules $e_i \blacktriangleright [f_j \rightarrow f_j]$ (for all $i, j \in \mathbb{N}^+$) and $e_i \triangleright [e_i \blacktriangleright [f_j \rightarrow f_j]]$ (for all $i, j \in \mathbb{N}^+$) (Rules (4.17) and (4.18)).

If there is an event x with $x \rightarrow \% f$, then after any occurrence of x , the event f is excluded, and therefore, not needed for acceptance (see Definition 4.7) (even if e reoccurs). Thus we have the rules $x_i \triangleright [b^f \rightarrow A]$ (for all $i \in \mathbb{N}^+$) and $x_i \triangleright [e_j \blacktriangleright [b^f \rightarrow A]]$ (for all $i, j \in \mathbb{N}^+$) (Rules (4.20) and (4.21)). Finally, if there is an event n with $n \rightarrow + f$, it is clear that after the concurrent occurrence of e and n , the event f is restless, and included, so we have the rules $\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A]$ (for all $i, j \in \mathbb{N}^+$) (Rule (4.22)). However, since we want to have this effect only on synchronisation of n_i and e_j , we use the extended synchronisation pattern $\text{eSync}(\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A])$ (as in Definition 3.16). Therefore, we have the Rules (4.23) to (4.30).

If n_i and e_j occur interleaved without the excluder x or the response f , the second one inserts the acceptance blocking $n_i \blacktriangleright [e_j \blacktriangleright [b^f \rightarrow A]]$ (for all $i, j \in \mathbb{N}^+$) (Rule (4.31)) and respectively $e_j \blacktriangleright [n_i \blacktriangleright [b^f \rightarrow A]]$ (for all $i, j \in \mathbb{N}^+$) (Rule (4.32)).

The case of first n occurring followed by x and finally e is already handled with Rule (4.21), the dual case where first e occurs, then f , and finally n is dealt by following rules $f_i \triangleright [n_j \blacktriangleright [b^f \rightarrow A]]$ (for all $i, j \in \mathbb{N}^+$) (Rule (4.33)).

Let us now consider the marking $M = (\text{Ex}, \text{Re}, \text{In}, \text{Count})$ of G . First, for all $e \in \text{Ex}$ there is $(e, n) \in \text{Count}$ with $n \geq 1$. We insert $\{e_i \mid i \leq n\}$ into the initial configuration C_0 . For all initially excluded events e , we have the rules $[b^e \rightarrow e_i]$ (for all $i \in \mathbb{N}^+$) (Rule (4.34)). If there is another event f with $e \rightarrow \bullet f$, we have to remove the rules $[e_1 \rightarrow f_i]$ (for all $i \in \mathbb{N}^+$) (Rules (4.3)) since only included conditions are needed.

It becomes a little tricky for initially restless events f . Here, there are two cases: First, there is another event e triggering a response from f (i.e. $e \bullet \rightarrow f$), second, f is only initially restless and no other event can make it restless. In the first case we do not need many rules since they are already around caused by $e \bullet \rightarrow f$. If f is initially restless and included, we have the acceptance blocking

$[b^f \rightarrow A]$ (Rule (4.40)). If there also is an includer n for f , we have the rules $n_i \triangleright [b^f \rightarrow A]$ (for all $i \in \mathbb{N}^+$) Rule (4.35)) since f might get excluded and on re-inclusion, if still restless, the acceptance needs to be blocked. In the first cases, we are done and the following rules coincide with the ones defined above. We can now assume that there is no e with $e \bullet \rightarrow f$. If there is an excluder x for f , we have $x_i \triangleright [b^f \rightarrow A]$ (for all $i \in \mathbb{N}^+$) Rule (4.38)) since we assumed there is no e with $e \bullet \rightarrow f$. Particularly, there is no $f \bullet \rightarrow f$. Therefore, the occurrence of f does not make f restless, and thus, we have that f drops the acceptance blocking $f_{k+1} \triangleright [b^f \rightarrow A]$ (Rule (4.36)), where f_{k+1} is the next occurrence of f based on $(f, k) \in \text{Count}$. Since there is no event making f restless again if f occurs, it will never become restless again, and therefore, it should drop the acceptance blocking deletion and inclusion rules $f_1 \triangleright [x_i \triangleright [b^f \rightarrow A]]$ (for all $i \in \mathbb{N}^+$) and $f_1 \triangleright [n_i \triangleright [b^f \rightarrow A]]$ (for all $i \in \mathbb{N}^+$) (Rules (4.37) and (4.39)).

Definition 4.18. Let $G = (E, M, R)$ be an extended DCR-Graph. We define

- the HDES encoding of G as the labelled HDES $H_G = (E_G, R_G, l, E, C_G^0)$ with the initial state (C_G^0, R_G)
- where $E_G := \{e_i \mid i \in \mathbb{N}^+ \wedge e \in E\} \cup \{b^e \mid e \in E\} \cup \{A\}$
- the labelling function l mapping all events to the original ones in G , that means $l : E_G \setminus (\{b^e \mid e \in E\} \cup \{A\}) \rightarrow E$ with $e_i \mapsto e (\forall i \in \mathbb{N}^+)$.
- For each $e \in E$ there is $[e_i \rightarrow e_{i+1}] \in R_G \quad (\forall i \in \mathbb{N}^+)$ and $[b^e \rightarrow b^e] \in R_G$.
- The other rules depend on the relations R of the DCR-Graph G (or the initial marking M of G) and are defined step by step:

- (1) For each pair in the condition relation $e \dashv\rightarrow f$, we have the rules

$$e_i \triangleright [f_j \rightarrow f_j] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.1)$$

$$e_i \triangleright [e_i \triangleright [f_j \rightarrow f_j]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.2)$$

and if e is initially included, we also have

$$[e_1 \rightarrow f_i] \quad (\forall i \in \mathbb{N}^+). \quad (4.3)$$

- (2) For each pair in the exclusion relation $x \dashv\rightarrow\% e$, we have the rules

$$x_i \triangleright [b^e \rightarrow e_j] \quad (\forall i, j \in \mathbb{N}^+). \quad (4.4)$$

If there is an event f with $e \rightarrow \bullet f$, then for every such f we also have the rules

$$x_i \triangleright [e_1 \rightarrow f_j] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.5)$$

$$x_i \blacktriangleright [f_j \rightarrow f_j] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.6)$$

$$x_i \triangleright [x_i \blacktriangleright [f_j \rightarrow f_j]] \quad (\forall i, j \in \mathbb{N}^+). \quad (4.7)$$

(3) For each pair in the inclusion relation $n \rightarrow + e$, we have the rules

$$n_i \triangleright [b^e \rightarrow e_j] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.8)$$

$$n_i \blacktriangleright [e_j \rightarrow e_j] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.9)$$

$$n_i \triangleright [n_i \blacktriangleright [e_j \rightarrow e_j]] \quad (\forall i, j \in \mathbb{N}^+). \quad (4.10)$$

If there is an event f with $e \rightarrow \bullet f$, then for every such f we also have the rules

$$n_i \blacktriangleright [e_1 \rightarrow f_j] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.11)$$

$$n_i \blacktriangleright [f_j \rightarrow f_j] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.12)$$

$$n_i \triangleright [n_i \blacktriangleright [f_j \rightarrow f_j]] \quad (\forall i, j \in \mathbb{N}^+). \quad (4.13)$$

If there also is an event x with $x \rightarrow \% e$, then for every such x we also add the following

$$x_i \blacktriangleright [n_j \rightarrow n_j] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.14)$$

$$x_i \triangleright [x_i \blacktriangleright [n_j \rightarrow n_j]] \quad (\forall i, j \in \mathbb{N}^+). \quad (4.15)$$

(4) For each pair in the response relation $e \bullet \rightarrow f$ with initially included f , we have the rules

$$e_i \blacktriangleright [b^f \rightarrow A] \quad (\forall i \in \mathbb{N}^+), \quad (4.16)$$

and if there is no $f \rightarrow \bullet f$, then we also have

$$e_i \blacktriangleright [f_j \rightarrow f_j] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.17)$$

$$e_i \triangleright [e_i \blacktriangleright [f_j \rightarrow f_j]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.18)$$

$$f_i \triangleright [b^f \rightarrow A] \quad (\forall i \in \mathbb{N}^+). \quad (4.19)$$

If there is an event x with $x \rightarrow \% f$, then for every such x we also have the rules

$$x_i \triangleright [b^f \rightarrow A] \quad (\forall i \in \mathbb{N}^+), \quad (4.20)$$

$$x_i \triangleright [e_j \blacktriangleright [b^f \rightarrow A]] \quad (\forall i, j \in \mathbb{N}^+). \quad (4.21)$$

If there is an event n with $n \rightarrow + f$, then for every such n we also have the rules of the extended synchronisation pattern

$$\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.22)$$

$$n_i \triangleright [\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.23)$$

$$e_j \triangleright [\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.24)$$

$$e_j \triangleright [n_i \triangleright [\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A]]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.25)$$

$$n_i \triangleright [e_j \triangleright [\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A]]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.26)$$

$$n_i \triangleright [e_j \triangleright [n_i \triangleright [\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A]]]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.27)$$

$$e_j \triangleright [n_i \triangleright [e_j \triangleright [\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A]]]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.28)$$

$$\{n_i, e_j\} \triangleright [n_i \triangleright [e_j \triangleright [n_i \triangleright [\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A]]]]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.29)$$

$$\{n_i, e_j\} \triangleright [e_j \triangleright [n_i \triangleright [e_j \triangleright [\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A]]]]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.30)$$

$$n_i \blacktriangleright [e_j \blacktriangleright [b^f \rightarrow A]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.31)$$

$$e_j \blacktriangleright [n_i \blacktriangleright [b^f \rightarrow A]] \quad (\forall i, j \in \mathbb{N}^+), \quad (4.32)$$

and if there is no $f \rightarrow \bullet f$ then we also have

$$f_i \triangleright [n_j \blacktriangleright [b^f \rightarrow A]] \quad (\forall i, j \in \mathbb{N}^+). \quad (4.33)$$

(5) If the marking M of G is *non-trivial*,

- that means if there are exclude, restless, or executed events
- this gets encoded, in the initial configuration of H_G and in R_G .

- For every initially executed event e , we add $\{e_i \mid i \leq n\}$, where n is the number of occurrences of e (i.e. $(e, n) \in \text{Count}$), to the initial configuration of H_G

$$C_G^0 = \{e_i \mid 1 \leq i \leq k \wedge (e, k) \in \text{Count}\}.$$

- (i) For every initially excluded event e , we add the following rule

$$[b^e \rightarrow e_i], \quad (\forall i \in \mathbb{N}^+). \quad (4.34)$$

- (ii) For every initially restless event f , if there are events n with $n \rightarrow +f$, for each of them we have the rules

$$n_i \blacktriangleright [b^f \rightarrow A] \quad (\forall i \in \mathbb{N}^+). \quad (4.35)$$

If there is no e with $e \bullet \rightarrow f$, we also have

$$f_{k+1} \triangleright [b^f \rightarrow A], \quad (4.36)$$

and if there also are events n with $n \rightarrow +f$, for each of them we additionally have the rules

$$f_{k+1} \triangleright [n_i \blacktriangleright [b^f \rightarrow A]] \quad (\forall i \in \mathbb{N}^+), \quad (4.37)$$

and if there also are events x with $x \rightarrow \%f$, for each of them we additionally have the rules

$$x_i \triangleright [b^f \rightarrow A] \quad (\forall i \in \mathbb{N}^+), \quad (4.38)$$

$$f_{k+1} \triangleright [x_i \triangleright [b^f \rightarrow A]] \quad (\forall i \in \mathbb{N}^+), \quad (4.39)$$

where k is the number of occurrences of f , meaning $((f, k) \in \text{Count})$. If f is initially included, we additionally have the rule

$$[b^f \rightarrow A]. \quad (4.40)$$

- (iii) • Finally, we have to consider all $(e, n) \in \text{Count}$ with $n \geq 1$.
- We remove all rules where e_i (with $i \leq n$) occurs as a modifier, as a cause, or as a target.

- For example, if there is an $f \in E$ with $e \rightarrow \bullet f$, we remove every rule $e_i \blacktriangleright [f_j \rightarrow f_j]$ with $i \leq n$, which has been created by Rule (4.1).
- However, if e_i (with $i \leq n$) only occurs in the causality part of a rule, this rule is removed, for example, if there are $f, x \in E$ with $e \rightarrow \bullet f$ and $x \rightarrow \% e$.
- The rule $x_m \blacktriangleright [e_k \rightarrow e_k]$ which was created by Rules (4.6), is removed from the rule sets.

4.2.2 Correctness of the Encoding

In this subsection we prove the correctness of the HDESSs encoding for basic DCR-Graphs. We first show that the initial behaviour is the same, meaning that the same sets of events are concurrently enabled and the DCR-Graph is initially in an accepting state if and only if the HDESS encoding is in an initially accepting state.

Lemma 4.19. *Let $G = (E, (\text{Ex}, \text{Re}, \text{In}, \text{Count}), R)$ be an extended DCR-Graph*

- *and $H_G = (E_G, R_G, l, E, C_G^0)$ its encoding as defined in Definition 4.18.*

Then, if e is enabled in G (with $(e, k) \in \text{Count}$), e_{k+1} is initially enabled in H_G .

Proof. Let $e \in E$ be initially enabled in G . Then by Definition 4.2, e is included and there is no condition f for e not executed but included. Since e is initially included, we do not have the rules $[b^e \rightarrow e_i] \ (\forall i \in \mathbb{N}^+)$ (4.34). On the other hand, if we have any condition $f \rightarrow \bullet e$, we would have $f_1 \rightarrow e_i \ (\forall i \in \mathbb{N}^+)$ by Rule (4.3) blocking e ; however by assumption, f must be initially executed or excluded. If f is initially executed, we have $f_1 \in C_G^0$ and thus $f_1 \rightarrow e_i \ (\forall i \in \mathbb{N}^+)$ does not block e . However, the rules $f_1 \rightarrow e_i \ (\forall i \in \mathbb{N}^+)$ are not initially included and e is unblocked. Thus, the event e_{k+1} (since $(e, k) \in \text{Count}$, i.e. e already occurred k times) is enabled in H_G in the state (C_G^0, R_G) . \square

Lemma 4.20. *Let $G = (E, M, R)$ be an extended DCR-Graph*

- *and $H_G = (E_G, R_G, l, E, C_G^0)$ its encoding as defined in Definition 4.18.*

Then, if any event $x \neq A$ is initially enabled in H_G , $x = e_i$ and e is enabled in G .

Proof. Since $x \neq A$ is enabled in H_G in the state (C_G^0, R_G) , it follows that $x = e_i$ for some $e \in E$ and $i \in \mathbb{N}^+$ since no b^e can ever be enabled (caused by the rule $[b^e \rightarrow b^e]$ never being dropped) by construction of R_G in Definition 4.18. By the same definition, we have if there are $f \in E$ with $f \rightarrow \bullet e$, then f is excluded. The conditions Rules $f_1 \rightarrow e_i \quad (\forall i \in \mathbb{N}^+)$ (4.3) are not added or f is initially marked as executed and so $f_1 \in C_G^0$ and thus $f_1 \rightarrow e_i \quad (\forall i \in \mathbb{N}^+)$ is not constraining e_i because otherwise e_i would not be enabled in H_G (obviously e is included in G initially because otherwise it would be blocked in (C_G^0, R_G) by the Rules (4.34)). Finally, we have that e is included initially in G and all its conditions are executed or excluded. Thus e is enabled in G . \square

Lemma 4.21. Let $G = (E, M, R)$ be an extended DCR-Graph

- and $H_G = (E_G, R_G, l, E, C_G^0)$ its encoding as defined in Definition 4.18.

Then both have the same enabled events modulo labelling and the acceptance event $A \in E'$.

Proof. Immediate by Lemmata 4.19 and 4.20. \square

Lemma 4.22. Let $G = (E, (Ex, Re, In, Count), R)$ be an extended DCR-Graph

- and $H_G = (E_G, R_G, l, E, C_G^0)$ its encoding as defined in the Definition 4.18.

Then,

- if $X \subseteq E$ is concurrently enabled in G ,
- $X' := \{e_{k+1} \mid (e \in X) \wedge ((e, k) \in \text{Count})\} \subseteq E_G$ is concurrently enabled in H_G in the state (C_G^0, R_G) with $l(X') = X$.

Proof. By Lemma 4.19, we have that each $x \in X'$ is enabled in H_G . The construction in Definition 4.18 reflects exactly the case where concurrency is prevented in DCR-Graphs (in Definitions 4.8 and 4.10): The Rule (4.14) reflects Condition (1) of the effect orthogonality (Definition 4.8) and the Rule (4.17) reflects Condition (2). Regarding the cause-orthogonality (Definition 4.10) the Rule (4.1) reflects Condition (1), the Rules (4.4) and (4.9) reflect Condition (2), and Condition (3) is reflected by the Rules (4.6) and (4.12). These rules are the only ones possibly

constraining concurrency for any $x \in X'$. As they would only forbid behaviour which is already forbidden in the DCR-Graph, the set X' is concurrently enabled in H_G . \square

Lemma 4.23. *Let $G = (E, M, R)$ be an extended DCR-Graph*

- *and $H_G = (E_G, R_G, l, E, C_G^0)$ its encoding as defined in Definition 4.18.*

Then,

- *if any set event $X \not\# A$ is concurrently enabled in H_G in the state (C_G^0, R_G) ,*
- *$l(X)$ is concurrently enabled in G .*

Proof. Analogously to the proof of Lemma 4.22. \square

Lemma 4.24. *Let $G = (E, M, R)$ be an extended DCR-Graph and*

- *$H_G = (E_G, R_G, l, E, C_G^0)$ its encoding with initial state (C_G^0, R_G) as defined in Definition 4.18.*

Then, both have the same sets of concurrently enabled events modulo labelling and the acceptance event $A \in E'$.

Proof. Immediate by Lemmata 4.22 and 4.23. \square

Lemma 4.25. *Let $G = (E, M, R)$ be an extended DCR-Graph*

- *and $H_G = (E_G, R_G, l, E, C_G^0)$ its encoding as defined in Definition 4.18.*

Then, G initially is in an accepting state if and only if H_G initially is in an accepting state.

Proof. Let G be initially accepting. Then, by the Definition 4.7, all initially restless events must be excluded. Therefore, by the construction in Definition 4.18, no rule of the form $[b^e \rightarrow A]$ is in R_G , and therefore by Definition 4.15, H_G is initially accepting. On the other hand, if H_G is initially accepting, then there is no rule of the form $[b^e \rightarrow A]$ in R_G , and thus by construction of the encoding, all initially restless events of G are excluded. Thus, G is accepting. \square

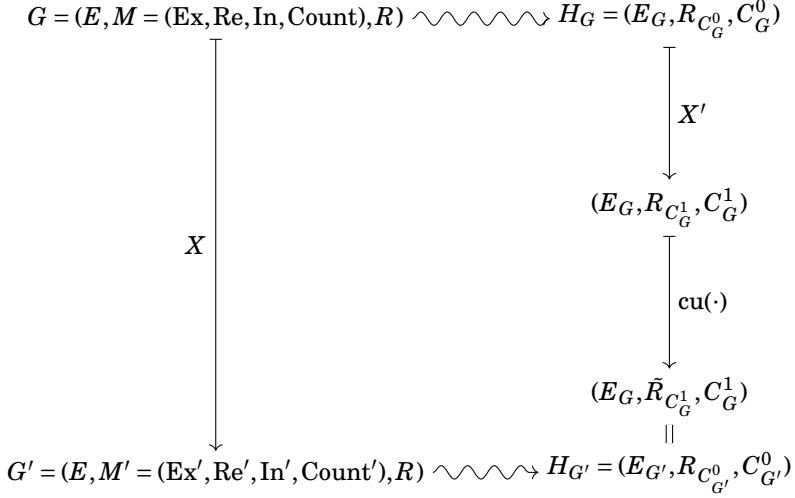


Figure 17: Sketch for the proof of Lemma 4.26

Lemma 4.26. *Let $G = (E, (\text{Ex}, \text{Re}, \text{In}, \text{Count}), R)$ be an extended DCR-Graph*

- *and $H_G = (E_G, R_{C_G^0}, l, E, C_G^0)$ as defined in Definition 4.18.*
- *Let $X \subseteq E$ be concurrently enabled in G .*

Then

- *the encoding $H_{G'} = (E_{G'}, R_{C_{G'}^0}, l, E, C_{G'}^0)$ of $G' := G \oplus X$*
- *is exactly the same as we would obtain after a clean-up,*
- *if X' (as defined in Lemma 4.22) would initially occur in H_G .*

Proof. Figure 17 sketches the proof with all involved structures. In the following we show that the diagram commutes. In Figure 18, a more detailed view on the rule sets $\tilde{R}_{C_G^1}$ and $R_{C_{G'}^0}$ can be found.

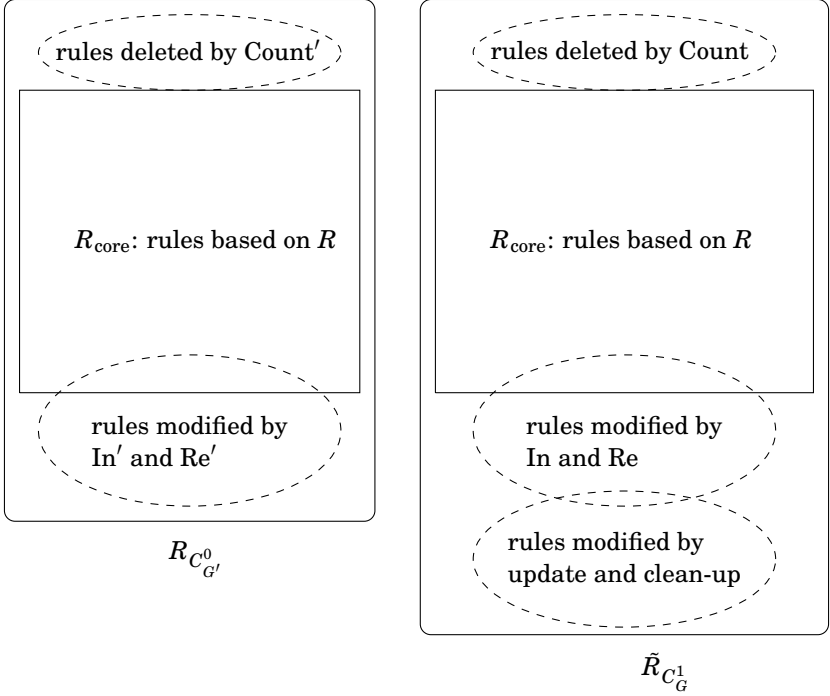


Figure 18: The rule sets $R_{C_{G'}^0}$ and $\tilde{R}_{C_G^1}$ for the proof of Lemma 4.26.

By Lemma 4.22, the set $X' = \{e_{k+1} \mid (e \in X) \wedge ((e, k) \in \text{Count})\}$ is concurrently enabled in (C_G^0, R_G) . There is a transition to a state $(C_G^1, R_{C_G^1})$, where $C_G^1 = C_G^0 \cup X'$ and $(C_G^0, R_G) \rightarrow_{C_G^1} R_{C_G^1}$ (i.e. $R_{C_G^1}^1$ is the rule update with respect to C_G^1).

We first show that $C_G^1 = C_{G'}^0$, and second that the clean-up $\tilde{R}_{C_G^1}$ (as defined in Definition 3.8) of $R_{C_G^1}$ with respect to X' equals $R_{C_{G'}^0}$. By definition, we have $C_{G'}^0 = \{e_i \mid 1 \leq i \leq k \wedge (e, k) \in \text{Count}'\}$. Note that we can obtain Count' from Count and X as

$$\begin{aligned} \text{Count}' = & \{(e, k) \mid (e \notin X) \wedge ((e, k) \in \text{Count})\} \\ & \cup \{(e, k+1) \mid (e \in X) \wedge ((e, k) \in \text{Count})\}, \end{aligned}$$

since we increase the counter of each event $x \in X$ by one (according to Definition 4.17). Therefore, we can split $C_{G'}^0$ into the two parts:

$$\begin{aligned} C_{G'}^0 = & \{e_i \mid 1 \leq i \leq k \wedge (e \notin X) \wedge ((e, k) \in \text{Count})\} \\ & \cup \{e_i \mid 1 \leq i \leq k+1 \wedge (e \in X) \wedge ((e, k) \in \text{Count})\} \end{aligned}$$

Since $X' = \{e_{k+1} \mid (e \in X) \wedge ((e, k) \in \text{Count})\}$, we can split the second set of $C_{G'}^0$ once more and obtain:

$$\begin{aligned} C_{G'}^0 = & \{e_i \mid 1 \leq i \leq k \wedge (e \notin X) \wedge ((e, k) \in \text{Count})\} \\ & \cup \{e_i \mid 1 \leq i \leq k \wedge (e \in X) \wedge ((e, k) \in \text{Count})\} \\ & \cup X' \end{aligned}$$

By merging the first two sets, we obtain the definition of C_G^0 . Therefore, we have

$$C_{G'}^0 = C_G^0 \cup X'.$$

On the other hand, we obviously have

$$\begin{aligned} \text{Count} = & \{(e, k) \mid (e \notin X) \wedge ((e, k) \in \text{Count})\} \\ & \cup \{(e, k) \mid (e \in X) \wedge ((e, k) \in \text{Count})\} \end{aligned}$$

Now we show that the rule sets $R_{C_{G'}^0}$ and $\tilde{R}_{C_G^1}$ (cf. Figure 18), where $\tilde{R}_{C_G^1}$ is the clean-up of $R_{C_G^1}$ with respect to X' , coincide. Both rule sets have a common core, namely R_{core} , from which certain rules already are initially deleted, to be precise those rules, whose modifiers are contained in Count and Count' respectively. On the other hand the core set gets modified depending on the respective marking, meaning based on initial excluded or restless events of M (resp. M').

We have to show that all modifications of R_{core} in $\tilde{R}_{C_G^1}$ are present in $R_{C_{G'}^0}$ and vice versa.

In the case of $\tilde{R}_{C_G^1}$, three modifications took place:

- (1) Every rule with a modifier, cause, or target in e_i , where $(e, i) \in \text{Count}$ got deleted.
- (2) Marking dependent Rules (4.3), (4.16) and (4.34) to (4.40).
- (3) A rule update with respect to X' took place followed by the clean-up.

In the case of $R_{C_{G'}^0}$, only two modifications took place:

- (a) Every rule with a modifier, cause, or target in e_i , where $(e, i) \in \text{Count}'$ got deleted.
- (b) Marking dependent Rules (4.3), (4.16) and (4.34) to (4.40).

We start with $\tilde{R}_{C_G^1}$ and show that each insertion or deletion can be found in $R_{C_{G'}^0}$: Point (1) is handled by (a) of $R_{C_{G'}^0}$ since any rule delete in (1), also gets deleted by (a). As, for all $e \in E$ with $(e, i) \in \text{Count}$ and $(e, j) \in \text{Count}'$, it is satisfied that $i \leq j$ (with equality if $e \notin X$, otherwise we have $i + 1 = j$).

Point (2) is only relevant for the events where the marking changes from M to M' and is addressed below.

Therefore we focus on (3). Let us consider the state (C_G^0, R_G) of H_G . The active rules with respect to X' can be partitioned into three types: First concurrency-constraining rules (like Rules (4.1) and (4.2)), second the rules of the extended synchronisation-pattern (Rules (4.22) to (4.30)), and third possibly marking-changing rules (like Rules (4.4)) which do have an effect on the rule set $R_{C_G^1}$ other than being removed. We first consider all rules of the second type and show how their effect can be found in $R_{C_{G'}^0}$.

If any rule of $\text{eSync}(\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A])$ becomes active, then n_i or e_j belong to X' . If both n_i and e_j are in X' , then by Lemma 3.18, we know that $[b^f \rightarrow A]$ is in $\tilde{R}_{C_G^1}$ but no Rules (4.22) to (4.30). However, then f is included and restless in G' , and therefore, we have $[b^f \rightarrow A]$ is in $R_{C_{G'}^0}$ by Rule (4.40). If, without loss of generality, only n_i is in X' , we again have that all rules in $\text{eSync}(\{n_i, e_j\} \blacktriangleright [b^f \rightarrow A])$ are removed in the clean-up from $\tilde{R}_{C_G^1}$, but we do have no further influence on this rule set (by Lemma 3.18).

Now we take the rules of the third type into account (first those created by the relations of the graph and second those caused by the initial marking).

- $x_i \blacktriangleright [b^e \rightarrow e_j] \quad (\forall i, j \in \mathbb{N}^+)$ (Rule (4.4)): This reflects the blocking of e after the exclusion of e by x . There are the Rules $[b^e \rightarrow e_i]$ for all $i \in \mathbb{N}^+$ in $\tilde{R}_{C_G^1}$. On the other hand, we have $[b^e \rightarrow e_i]$ for all $i \in \mathbb{N}^+$ in $R_{C_{G'}^0}$ (by the Rule (4.34)) since e is excluded in G' because $x \in X$ and $x \rightarrow_{\%} f$.
- $x_i \triangleright [e_1 \rightarrow f_j] \quad (\forall i, j \in \mathbb{N}^+)$ (Rule (4.5)): These rules encode that excluded conditions are not considered. We do not have $[e_1 \rightarrow f_j]$ in $\tilde{R}_{C_G^1}$. However,

as e is excluded in G' because $x \in X$ and $x \rightarrow \% f$, no such rules appears in $R_{C_{G'}^0}$ (Rule (4.3) only creates them if e is included).

- $n_i \triangleright [b^e \rightarrow e_j] \quad (\forall i, j \in \mathbb{N}^+)$ (Rule (4.8)): This reflects the unblocking of e after the inclusion of e by n . Therefore, $[b^e \rightarrow e_j]$ for all $j \in \mathbb{N}^+$ are not in $\tilde{R}_{C_G^1}$ since no concurrent excluder that would reinstall theses rules is allowed (cf. Definition 4.8 Condition (1)). These rules do not appear in $R_{C_{G'}^0}$ since e is included in G' because $n \in X$ and $n \rightarrow + e$ (Rule (4.34) only only creates them if e is excluded).
- $n_i \blacktriangleright [e_1 \rightarrow f_j] \quad (\forall i, j \in \mathbb{N}^+)$ (Rule (4.11)): The inclusion of e reinstalls the condition for f (again e cannot concurrently get excluded cf. Definition 4.8 Condition (1)). We have $[e_1 \rightarrow f_j]$ for all $j \in \mathbb{N}^+$ in $\tilde{R}_{C_G^1}$. As e is included in G' (because $n \in X$ and $n \rightarrow + e$) we have $[e_1 \rightarrow f_j]$ for all $j \in \mathbb{N}^+$ in $R_{C_{G'}^0}$ (by Rule (4.3)) since e is an included condition for f .
- $e_i \blacktriangleright [b^f \rightarrow A] \quad (\forall i \in \mathbb{N}^+)$ (Rule (4.16)): As e makes f restless, and since f is only allowed to be concurrent with e if it makes itself restless, f is restless in G' . On the other hand, f is included in G' since it was included in G . Otherwise, the rule $e_i \blacktriangleright [b^f \rightarrow A]$ for all $i \in \mathbb{N}^+$ would not be in $R_{C_G^0}$, and there is no $x \in X$ with $x \rightarrow \% f$ since this would imply that the $x_j \triangleright [e_i \blacktriangleright [b^f \rightarrow A]]$ for all $i, j \in \mathbb{N}^+$ (Rule (4.21)) are in $R_{C_G^0}$. Where for some x_j one of the rules deletes the rule $e_i \blacktriangleright [b^f \rightarrow A]$ for all $i \in \mathbb{N}^+$ which is not the case since it became active. As an effect, we have $[b^f \rightarrow A]$ in $\tilde{R}_{C_G^1}$. However, since f is included because it was included in G and was not excluded by some $x \in X$, and restless because $e \in X$ and $e \bullet \rightarrow f$, this rule is in $R_{C_{G'}^0}$ (by Rule (4.40)).
- $f_i \triangleright [b^f \rightarrow A] \quad (\forall i \in \mathbb{N}^+)$ (Rule (4.19)): The occurrence of f removes it from the set of restless events (if not $f \bullet \rightarrow f$, but then these rules would not exist). There is no rule $[b^f \rightarrow A]$ in $\tilde{R}_{C_G^1}$. However, since f is not restless in G' (because $f \in X$ and not $f \bullet \rightarrow f$), these rules are not in $R_{C_{G'}^0}$ (Rule (4.40) only creates them if f is included and restless).
- $x_i \triangleright [b^f \rightarrow A] \quad (\forall i \in \mathbb{N}^+)$ (Rule (4.20)): As x excludes f , there is no f needed for an acceptance blocker and thus the blocking rule $[b^f \rightarrow A]$ is not in $\tilde{R}_{C_G^1}$.

It is not in $R_{C_{G'}^0}$ (by Rule (4.40)) since $x \in X$ and $x \longrightarrow \% f$, where f is excluded in G' .

- $x_i \triangleright [e_j \blacktriangleright [b^f \rightarrow A]] \quad (\forall i, j \in \mathbb{N}^+)$ (Rule (4.21)): Since x excludes f , and as long f stays excluded, e making f restless does not mean that f is acceptance blocking. Hence, $e_j \blacktriangleright [b^f \rightarrow A]$ is not in $\tilde{R}_{C_G^1}$ (because by Definition 4.8 Condition (1) there can be no concurrent inclusion of f). However, as f is excluded in G' because $x \in X$ and $x \longrightarrow \% f$, the rules $e_j \blacktriangleright [b^f \rightarrow A]$ are not in $R_{C_{G'}^0}$ (Rule (4.16) only creates them if f is included).
- $n_i \blacktriangleright [e_j \blacktriangleright [b^f \rightarrow A]] \quad (\forall i, j \in \mathbb{N}^+)$ (Rule (4.31)): The includer n allows the trigger e to install the acceptance blocker $[b^f \rightarrow A]$. If only $n_i \in X'$ but $e_j \notin X'$, we have $e_j \blacktriangleright [b^f \rightarrow A]$ for all $j \in \mathbb{N}^+$ in $\tilde{R}_{C_G^1}$. Otherwise, this rule gets removed by the clean-up and the effect is already applied by Rule (4.22). If only $n \in X$, then f is included in G' because $n \longrightarrow + f$. We have $e_j \blacktriangleright [b^f \rightarrow A]$ for all $j \in \mathbb{N}^+$ (Rule (4.16)) in $R_{C_{G'}^0}$.
- $e_j \blacktriangleright [n_i \blacktriangleright [b^f \rightarrow A]] \quad (\forall i, j \in \mathbb{N}^+)$ (Rule (4.32)). The trigger e allows the includer n to install the acceptance blocker $[b^f \rightarrow A]$. If only $e_j \in X'$, but $n_i \notin X'$, we have $n_i \blacktriangleright [b^f \rightarrow A]$ for all $i \in \mathbb{N}^+$ in $\tilde{R}_{C_G^1}$. Otherwise this rule gets removed by the clean-up and the effect is already applied by Rule (4.22). If only $e \in X$, f is restless in G' because $e \bullet \longrightarrow f$. Therefore, we have $n_i \blacktriangleright [b^f \rightarrow A]$ for all $i \in \mathbb{N}^+$ (Rule (4.35)) in $R_{C_{G'}^0}$.
- $f_i \triangleright [n_j \blacktriangleright [b^f \rightarrow A]] \quad (\forall i, j \in \mathbb{N}^+)$ (Rule (4.33)): After an event e making f restless, Rule (4.22) creates rules of the form $n_j \blacktriangleright [b^f \rightarrow A]$ for all $j \in \mathbb{N}^+$. Those rules recreate the acceptance blocker if f got excluded and re-included again. However, if f occurs before an includer, this effect must be negated (since we only create this rule if not $f \bullet \longrightarrow f$). There are no rules $n_j \blacktriangleright [b^f \rightarrow A]$ for all $j \in \mathbb{N}^+$ in $\tilde{R}_{C_G^1}$. Since f is not restless in G' because $f \in X$ and not $f \bullet \longrightarrow f$, there is no such rules in $R_{C_{G'}^0}$ (Rule (4.35) only creates them if f is restless).

Now we look at the rules caused by an initially restless event f .

- $n_i \blacktriangleright [b^f \rightarrow A] \quad (\forall i \in \mathbb{N}^+)$ (Rule (4.35)): These rules add the acceptance blocker $[b^f \rightarrow A]$ to the rule set $\tilde{R}_{C_G^1}$. Since they are in the current rule set, we have $f \notin X$ or $f \bullet \longrightarrow f$. Otherwise $n_i \blacktriangleright [b^f \rightarrow A]$ for all $i \in \mathbb{N}^+$ would have

been deleted by some f_k , either by Rules (4.33) or (4.37). However, since f is restless in G' , the graph G' is not accepting after the includer n and we have the acceptance blocker $b^f \rightarrow A$ (Rule (4.40)) in $R_{C_{G'}^0}$.

- $x_i \triangleright [b^f \rightarrow A]$ ($\forall i \in \mathbb{N}^+$) (Rule (4.38)): These rules delete the acceptance blocker $[b^f \rightarrow A]$ from the rule set $\tilde{R}_{C_G^1}$. However, since f got excluded by x because $x \in X$ and $x \rightarrow \% f$, we have no acceptance blocker in $R_{C_{G'}^0}$ (Rule (4.40) only creates it if f is included).
- $f_{k+1} \triangleright [x_i \triangleright [b^f \rightarrow A]]$ ($\forall i \in \mathbb{N}^+$) (Rule (4.39)): These rules drop the deletion of the acceptance blocker $x_i \triangleright [b^f \rightarrow A]$ for all $i \in \mathbb{N}^+$ (Rule (4.38)) from $\tilde{R}_{C_G^1}$. However, as f is not putting a response on itself (because there is no $e \in E$ with $e \bullet \rightarrow f$, and in particular not $f \bullet \rightarrow f$), there are no $x_i \triangleright [b^f \rightarrow A]$ for all $i \in \mathbb{N}^+$ (Rule (4.38)) in $R_{C_{G'}^0}$ since f is not restless anymore.
- $f_{k+1} \triangleright [n_i \blacktriangleright [b^f \rightarrow A]]$ ($\forall i \in \mathbb{N}^+$) (Rule (4.37)): These rules drop the insertion of the acceptance blocker $n_i \blacktriangleright [b^f \rightarrow A]$ for all $i \in \mathbb{N}^+$ (Rule (4.35)) from $\tilde{R}_{C_G^1}$. However, as f is not putting a response on itself (because there is no $e \in E$ with $e \bullet \rightarrow f$, and in particular not $f \bullet \rightarrow f$), there are no $n_i \blacktriangleright [b^f \rightarrow A]$ for all $i \in \mathbb{N}^+$ (Rule (4.35)) in $R_{C_{G'}^0}$ since f is not restless in G' anymore.
- $f_{k+1} \triangleright [b^f \rightarrow A]$ (Rule (4.36)): These rules delete all the acceptance blocker $[b^f \rightarrow A]$ (Rule (4.40)) from $\tilde{R}_{C_G^1}$. However, since f is not putting a response on itself (because there is no $e \in E$ with $e \bullet \rightarrow f$, and in particular, not $f \bullet \rightarrow f$), there is no $[b^f \rightarrow A]$ (Rule (4.40)) in $R_{C_{G'}^0}$ since f is not restless in G' anymore.

If we now consider the step from G to G' in the extended DCR-Graph, we have certain changes in the marking which modify the core rule set. We now show that these modifications also take place if we consider the rule update and clean-up in the translation.

- (i) e getting included, meaning $e \notin \text{In}$ but $e \in \text{In}'$. There is an $n \in X$ with $n \rightarrow +e$. Now, there are up to three rule patterns which might get included or excluded:
 - $[b^e \rightarrow e_i]$ ($\forall i \in \mathbb{N}^+$) (Rule (4.34)) are in $R_{C_G^0}$ but not in $R_{C_{G'}^0}$ (since they only get created if e is excluded). However, the blocker $[b^e \rightarrow e_i]$

for all $i \in \mathbb{N}^+$ gets removed with $n_i \triangleright [b^e \rightarrow e_j]$ for all $i, j \in \mathbb{N}^+$ (Rules (4.8)) which are in $R_{C_G^0}$ since there is $n \in E$ with $n \rightarrow + e$. And since $n \in X$, there is an l such that $n_l \in X'$. Therefore, $n_l \triangleright [b^e \rightarrow e_j]$ is in $R_{C_G^0}$ and active, and thus the blocker is removed in $\tilde{R}_{C_G^1}$.

- If there is an event f with $e \rightarrow \bullet f$, then by Rule (4.3) $[e_1 \rightarrow f_i]$ for all $i \in \mathbb{N}^+$ are in $R_{C_{G'}^0}$ but not in $R_{C_G^0}$ (as they only get created if e is included). However, the condition constraints $[e_1 \rightarrow f_i]$ for all $i \in \mathbb{N}^+$ get inserted with $n_i \triangleright [e_1 \rightarrow f_j]$ for all $i, j \in \mathbb{N}^+$ (Rule (4.11)) which are in $R_{C_G^0}$ since there is $n \in E$ with $n \rightarrow + e$ and e is excluded in G . And, since $n \in X$, there is an l such that $n_l \in X'$. Therefore, $n_l \triangleright [e_1 \rightarrow f_j]$ is in $R_{C_G^0}$ and active, and thus the condition constraint is inserted into $\tilde{R}_{C_G^1}$.
- If there is an event d with $d \bullet \rightarrow e$, then by Rule (4.16) $d_i \triangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ are in $R_{C_{G'}^0}$ but not in $R_{C_G^0}$ (as they only get created if e is included). However, the blocker of the acceptance $d_i \triangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ gets inserted with $n_i \triangleright [d_j \triangleright [b^e \rightarrow A]]$ for all $i, j \in \mathbb{N}^+$ (Rule (4.31)) which are in $R_{C_G^0}$ since there is $n \in E$ with $n \rightarrow + e$ and e is excluded in G . And since $n \in X$, there is an l such that $n_l \in X'$. Therefore, $n_l \triangleright [d_j \triangleright [b^e \rightarrow A]]$ is in $R_{C_G^0}$ and active, and thus, the condition constraint is inserted into $\tilde{R}_{C_G^1}$.

(ii) e getting excluded, meaning $e \in \text{In}$ but $e \notin \text{In}'$. There is an $x \in X$ with $x \rightarrow \% e$. Now, there are up to three rule patterns which might get included or excluded:

- $[b^e \rightarrow e_i] \ (\forall i \in \mathbb{N}^+)$ (Rule (4.34)) are in $R_{C_{G'}^0}$ but not in $R_{C_G^0}$ (as they only get created if e is excluded). However, the blocker $[b^e \rightarrow e_i]$ for all $i \in \mathbb{N}^+$ gets added with $x_i \triangleright [b^e \rightarrow e_j]$ for all $i, j \in \mathbb{N}^+$ (Rule (4.4)) which are in $R_{C_G^0}$ as there is $x \in E$ with $x \rightarrow \% e$. And, since $x \in X$, there is an l such that $x_l \in X'$. Therefore, $x_l \triangleright [b^e \rightarrow e_j]$ is in $R_{C_G^0}$ and active, and thus, the blocker is added to $\tilde{R}_{C_G^1}$.

- If there is an event f with $e \twoheadrightarrow f$, then by Rule (4.3) $[e_1 \rightarrow f_i]$ for all $i \in \mathbb{N}^+$ are in $R_{C_G^0}$ but not in $R_{C_{G'}^0}$ (as they only get created if e is included). However, the condition constraints $[e_1 \rightarrow f_i]$ for all $i \in \mathbb{N}^+$ get removed with $x_i \triangleright [e_1 \rightarrow f_j]$ for all $i, j \in \mathbb{N}^+$ (Rule (4.5)) which are in $R_{C_G^0}$ as there is $x \in E$ with $x \twoheadrightarrow e$. And, since $x \in X$, there is an l such that $x_l \in X'$. Therefore, $x_l \triangleright [e_1 \rightarrow f_j]$ is in $R_{C_G^0}$ and active, and thus the condition constraint is removed from $\tilde{R}_{C_G^1}$.
- (iii) e getting restless, meaning $e \in \text{Re}'$ but $e \notin \text{Re}$ (there is some $d \in X$ with $d \bullet \rightarrow e$):
 - and e stays included, meaning $e \in \text{In}$ and $e \in \text{In}'$:
 - $[b^e \rightarrow A]$ (Rule (4.40)) is in $R_{C_{G'}^0}$ but not in $R_{C_G^0}$ (as it only gets created if e is restless). However, the acceptance blocker $[b^e \rightarrow A]$ gets inserted with $d_i \blacktriangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ (Rule (4.16)) which are in $R_{C_G^0}$ as there is a $d \in E$ with $d \bullet \rightarrow e$ and e is initially included. And, since $d \in X$, there is an l such that $d_l \in X'$. Therefore, $d_l \blacktriangleright [b^e \rightarrow A]$ is in $R_{C_G^0}$ and active, and the acceptance blocker is added to $\tilde{R}_{C_G^1}$.
 - If there is an event n with $n \longrightarrow + e$, then by Rule (4.35) $n_i \blacktriangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ are in $R_{C_{G'}^0}$ but not in $R_{C_G^0}$ (as they only get created if e is restless). However, the acceptance blocking capability $n_i \blacktriangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ gets added with $d_j \blacktriangleright [n_i \blacktriangleright [b^e \rightarrow A]]$ for all $i, j \in \mathbb{N}^+$ (Rule (4.32)) which are in $R_{C_G^0}$ as there are $d, n \in E$ with $d \bullet \rightarrow e$ and $n \longrightarrow + e$. And, since $e \in X$, there is an l such that $d_l \in X'$. Therefore, $d_l \blacktriangleright [n_i \blacktriangleright [b^e \rightarrow A]]$ is in $R_{C_G^0}$ and active, and thus, acceptance blocking capability is added to $\tilde{R}_{C_G^1}$.
 - and e gets included, meaning $e \notin \text{In}$ and $e \in \text{In}'$, and there are $n, d \in X$ with $n \longrightarrow + e$ and $d \bullet \rightarrow e$:

- $[b^e \rightarrow A]$ (Rule (4.40)) is in $R_{C_{G'}^0}$ but not in $R_{C_G^0}$ (as it only gets created if e is restless). However, the acceptance blocker $[b^e \rightarrow A]$ gets inserted with $\{n_j, d_i\} \blacktriangleright [b^e \rightarrow A]$ for all $i, j \in \mathbb{N}^+$ (Rule (4.22)) which are in $R_{C_G^0}$ as there are $d, n \in E$ with $d \bullet \rightarrow e$ and $n \rightarrow +e$. And, since $d, n \in X$, there are l_d, l_n such that $d_{l_d}, n_{l_n} \in X'$. Therefore $\{n_{l_n}, d_{l_d}\} \blacktriangleright [b^e \rightarrow A]$ are in $R_{C_G^0}$ and active, and the acceptance blocker is added to $\tilde{R}_{C_G^1}$.
- Since there is an event n with $n \rightarrow +e$, then by Rule (4.35) $n_i \blacktriangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ are in $R_{C_{G'}^0}$ but not in $R_{C_G^0}$ (as they only get created if e is restless). However, the acceptance blocking capability $n_i \blacktriangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ gets added with $d_j \blacktriangleright [n_i \blacktriangleright [b^e \rightarrow A]]$ for all $i, j \in \mathbb{N}^+$ (Rule (4.32)) which are in $R_{C_G^0}$ as there are $d, n \in E$ with $d \bullet \rightarrow e$ and $n \rightarrow +e$. And, since $e \in X$, there is an l such that $d_l \in X'$. Therefore, $d_l \blacktriangleright [n_i \blacktriangleright [b^e \rightarrow A]]$ is in $R_{C_G^0}$ and active, and thus, acceptance blocking capability is added to $\tilde{R}_{C_G^1}$.
- and e gets excluded, meaning $e \in \text{In}$ and $e \notin \text{In}'$, and so there are $x, d \in X$ with $x \rightarrow \%e$ and $d \bullet \rightarrow e$:
 - If there is an event n with $n \rightarrow +e$, then by Rule (4.35) $n_i \blacktriangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ are in $R_{C_{G'}^0}$ but not in $R_{C_G^0}$ (as they only get created if e is restless). However, the acceptance blocking capability $n_i \blacktriangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ gets added with $d_j \blacktriangleright [n_i \blacktriangleright [b^e \rightarrow A]]$ for all $i, j \in \mathbb{N}^+$ (Rule (4.32)) which are in $R_{C_G^0}$ as there are $d, n \in E$ with $d \bullet \rightarrow e$ and $n \rightarrow +e$. And, since $e \in X$, there is an l such that $d_l \in X'$. Therefore, $d_l \blacktriangleright [n_i \blacktriangleright [b^e \rightarrow A]]$ is in $R_{C_G^0}$ and active, and thus, acceptance blocking capability is added to $\tilde{R}_{C_G^1}$.
- (iv) e getting removed from pending responses, meaning $e \notin \text{Re}'$ but $e \in \text{Re}$, and there is $e \in X$. Note that by cause orthogonality (cf. Definition 4.10) no parallel includer or excluder of e is permitted:

- $[b^e \rightarrow A]$ (Rule (4.40)) is in $R_{C_G^0}$ but not in $R_{C_{G'}^0}$ (as it only gets created if e is restless). However, the acceptance blocker $[b^e \rightarrow A]$ gets removed with $e_i \triangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ (Rule (4.19), respectively Rule (4.36) if there is no $d \in E$ with $d \bullet \rightarrow e$) which are in $R_{C_G^0}$ as e is initially restless. And since $e \in X$, there is an l such that $e_l \in X'$. Therefore, $e_l \triangleright [b^e \rightarrow A]$ is in $R_{C_G^0}$ and active, and thus, the acceptance blocker is removed from $\tilde{R}_{C_G^1}$.
- If there is an event n with $n \rightarrow + e$, then by Rule (4.35) $n_i \blacktriangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ are in $R_{C_G^0}$ but not in $R_{C_{G'}^0}$ (as they only get created if e is restless). However, the acceptance blocking capability $n_i \blacktriangleright [b^e \rightarrow A]$ for all $i \in \mathbb{N}^+$ gets removed with $e_j \triangleright [n_i \blacktriangleright [b^e \rightarrow A]]$ for all $i, j \in \mathbb{N}^+$ (Rule (4.33), respectively Rule (4.37) if there is no $d \in E$ with $d \bullet \rightarrow e$) which are in $R_{C_G^0}$ as e is initially restless. And, since $e \in X$, there is an l such that $e_l \in X'$. Therefore, $e_l \triangleright [n_i \blacktriangleright [b^e \rightarrow A]]$ is in $R_{C_G^0}$ and active, and thus, acceptance-blocking capability is removed from $\tilde{R}_{C_G^1}$.

(v) e getting executed for the first time, that means $e \notin \text{Ex}$ but $e \in \text{Ex}'$. This is covered by the next point.

- (vi) e getting executed for the $(k+1)$ -th time, meaning $(e, k) \in \text{Count}$ and $(e, k+1) \in \text{Count}'$: As mentioned above we add e_{k+1} to the new initial configuration $C_{G'}^0$, but we remove all rules containing e_{k+1} as a modifier, cause, or target (cf. Definition 4.18 Case (5).(iii)) from $R_{C_{G'}^0}$. This is also done in the rule update and clean-up step for $\tilde{R}_{C_G^1}$. Note since we use the extended synchronisation pattern for $\{n_i, e_{k+1}\} \blacktriangleright [b^f \rightarrow A]$, the occurrence of e_{k+1} (in combination with the clean-up) removes all rules of this pattern. However, this pattern is the only place where e_{k+1} occurs in a modifier set which is not a singleton and therefore easily removed by the clean-up.

Thus, we have $R_{C_{G'}^0} = \tilde{R}_{C_G^1}$, and therefore, $H_{G'} = (E_G, \tilde{R}_{C_G^1}, C_G^1)$. \square

Now we can formulate our result that DCR-Graphs without a labelling on the events (or just the identity) can be encoded into a slightly enhanced HDESs such that on finite executions they behave the same. However, for DCR-Graphs an acceptance criterion for infinite runs exists, but for HDESs we do not even have the notion of infinite runs.

Theorem 4.27. *For each DCR-Graph G (without labelled events) there is a labelled HDES H_G such that for finite execution both yield the same transition system.*

Proof. By interpreting G as an extended DCR-Graph with $(e, 1) \in \text{Count}$ if $e \in \text{Ex}$ and $(e, 0)$ otherwise, we can apply Lemma 4.26 inductively on the transition sequence. \square

There are certain points in the encoding (cf. Definition 4.18) to be commented.

Remark 4.28. There are a few things we would like to point out:

- In the encoding we used a synchronisation pattern, but the DCR-Graphs semantics does not have any dependencies on synchronous executions. However, since DCR-Graphs have an additional memory, the marking, it is easy to see if an event is restless. On the other hand in the HDES encoding, we have to treat the cases of concurrent and interleaved occurrence of n, e differently (for $e \bullet \rightarrow f$ and $n \rightarrow + f$), in order to simulate the restlessness correctly.
- The usage of the extended synchronisation pattern, instead of the basic one, allows for a shorter proof, but adds more complex rules.
- The usage of non-empty initial configurations in the encoding is for simplifying the proof. However, this could be translated to an equivalent structure with an empty initial configuration.
- There are more advanced variants of DCR-Graph. In [15] there is a milestone relation $e \multimap f$ (as long e is restless, f is blocked), nesting (there is a tree-like structure for the events and the relations are inherited top down), and sub-graph spawning (including fresh events). However, only for the variant we considered, there is a defined concurrency semantics.
- In [1] Arbach introduced an evolutionary version of DCEs. There, not only the dependencies between the events might get changed, but also new events can get inserted into the structure. If this approach is lifted to HDESs, this could make the above presented encoding extremely smaller: Instead of the event chains $\{e_i \mid i \in \mathbb{N}^+ \wedge e \in E\}$ we would initially only need $\{e_0 \mid e \in E\}$. For the rules we would initially only need those where all the indices which now range over all naturals, equal zero. On the occurrence of an event e_i , this structure would evolve to a new ES which includes e_{i+1} and all rules which contain e_{i+1} .

In this way, it is possible to get a finite evolution scheme: A finite initial ES and a finite set of rules which get inserted when an event occurs. Such a model would have the huge benefit to only model and visualise the current state, in contrast to now, where every possible future behaviour must be taken into account.

5 Application

In this chapter we first present a medical case study from a bachelor thesis [22] modelling the patient treatment. Therefore we introduce the case study, explain the surgery part in more detail, show an example situation in which we need second-order set-based dynamics. Second, based on this case study, we discuss the requirements on HDESs for further applicability.

5.1 Case Study

In his bachelor thesis Trénous [22] analysed the utility and usability of DCESs to model dynamic processes in a clinical case study. The case of a patient at the German Heart Institute Berlin (DHZB) suffering from a coronary heart disease and a mitral valve insufficiency is modelled. Since many complications occurred, the treatment was dynamically adapted many times. Thus, it is considered a good indicator for how well DCESs are suited to model dynamic processes. Later on, Gräffe used DCR-Graphs in his bachelor thesis [13] to model the same case study.

5.1.1 The Surgery

The following summary is based on the description of the case study in Section 2.2 and the surgery in Section 4.2.1 of [22]:

A chain of complications lead to dynamic adjustments of the treatment process. Before the start of the surgery the patient is connected to a heart-lung-machine (HML) which replaces the functionality of the heart and the lung for the duration of the surgery. In order to prevent blood clotting the anticoagulant heparin must be administered during an HML treatment. After a normal surgery, the patient would be disconnected from the HML, the heart would be reanimated with the help of an intravenous adrenaline injection, and heparin would be discontinued.

In this particular case, the aorta of the patient was accidentally pierced during the surgery leading to severe pulmonary bleeding and acute pulmonary failure.

As a life saving measure the patient was connected to a veno-arterial extra-corporeal membrane oxygenation (VA-ECMO), a machine that does the work of heart and lung of the patient. This VA-ECMO replaces the HML since it is better suited for a prolonged treatment. In order to mimic the functionality of the heart, the blood is pumped through an external machine that enriches it with oxygen. To prevent blood clotting the anticoagulant heparin must be administered during a VA-ECMO (as before with the HML) treatment but in this case it cannot be stopped after the surgery.

When the heart function of the patient was reasonably well, the VA-ECMO was replaced by a veno-venous extra-corporeal membrane oxygenation (VV-ECMO). A VV-ECMO only oxygenates the blood. The heart is again responsible for the pumping. In order to handle the increased workload the heart was supported with an increased dose of adrenaline.

Thereafter, an even further deterioration of the lung functions was detected and an x-ray examination indicated an accumulation of blood in the thorax (hemothorax). After further treatment and examination it was assumed that the hemothorax was a complication caused by heparin. Since the patient was relying on VV-ECMO, the heparin could not be stopped.

At this point we stop the summary of the case study. In [22] it continues with some further complications but eventually the patient was transferred from the DHZB to a regular hospital.

5.1.2 Some Modelling

We focus on the modelling of the contraindication of hemothorax and heparin and the interplay with the VV-ECMO treatment:

Let us first consider the interaction of heparin and a hemothorax. The hemothorax is an accumulation of blood in the thorax and heparin prevents the blood clotting. Thus, if a patient is prescribed heparin and suffers from a hemothorax normally the heparin treatment should be stopped before further steps are taken. In DCES this contraindication cannot be modelled nicely (i.e. without duplicating some events) since we only have single events as modifiers for dependencies. In Figure 19 which was created with our web tool (cf. Section 3.2) there is a model with set-based dynamics: The event `startHeparin` is a precondition for `stopHeparin`. We modelled that `startHeparine` and `hemothorax` together insert the dependency from `stopHeparin` to the remainder (of course this still is a HDES).

The situation becomes even more sophisticated if an acute respiratory distress syndrome enforces a treatment with an ECMO which requires the patient to

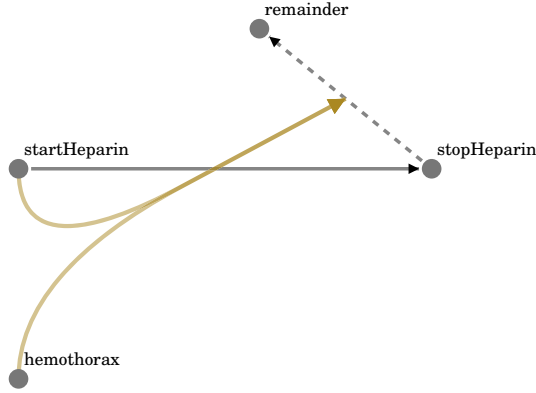


Figure 19: Example with set-based dynamics. A dynamic ES with set-based dynamics from the case study modelling the contraindication of hemothorax and heparin.

get heparin in order to prevent the blood clotting. Thus, the acute respiratory distress syndrome should disable the dynamics modelled above until a respiratory stabilisation is achieved. In this situation an ECMO treatment is no longer needed and the heparin treatment can be stopped.

In Figure 20 we have modelled the extended situation. We had to use not only the set-based dynamics, but also second-order dynamics while both of them are not part of the DCES formalism (of course they are HDES features). An example for the second-order dynamics is the following rule:

`aRDS ▷ [{startHeparin, hemothorax} ► [stopHeparin → remainder]]`

Where aRDS is an abbreviation for acute respiratory distress syndrome (For enhanced readability of the rules we use the notation of the web tool, as in Section 3.2). In the said figure it can be found as the line from the set-based adding to aRDS (the empty arrow tip at the start of the line denotes that the rule is absorbed, that means deleted, by aRDS). We have to consider the case where startHeparin and hemothorax both precede the aRDS, then the dependency stopHeparin → remainder is already inserted, and thus it must be deleted by aRDS. The respiratory stabilization reinserts the set-based adding, and if both stopHeparin and startHeparin preceded it, it also inserts the dependency stopHeparin → remainder. Example 5.1 lists all dependencies and dynamic rules from Figure 20 since the figure is rather complicated.

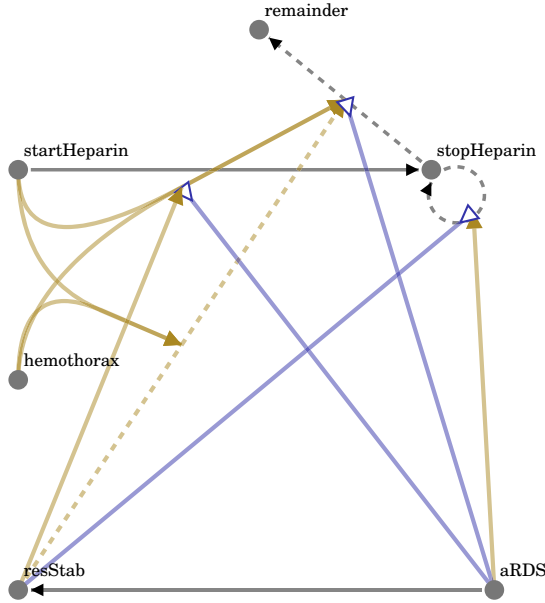


Figure 20: Example with set-based and second-order dynamics. A dynamic ES with set-based and second-order dynamics derived from the case study. This can be found in the web tool (cf. Section 3.2) at [THESIS Example 1](#).

5.1.3 Conclusion

In the above model we have seen that we need HDESs since DCEs are not expressive enough for this dynamic process. We needed some notion of higher-order adding and dropping and set-based dynamics. Please note that the event remainder in the above modelling is a place holder for many other events. We implicitly used grouping of events which is up to now neither part of DCEs nor the generalised HDES formalism. In the next section, we illustrate, not only based on this case study, which features are still missing in the HDES formalism for being applicable to real-world processes. As mentioned above, the same case

```

1 heparin+ → heparin-
2 aRDS → resStab
3 {heparin+, hemothorax} ► [heparin- → rem]
4 {hemothorax, heparin+} ► [resStab ► [heparin- → rem]]
5 aRDS ▷ [{heparin+, hemothorax} ► [heparin- → rem]]
6 aRDS ▷ [heparin- → rem]
7 aRDS ► [heparin- → heparin-]
8 resStab ▷ [heparin- → heparin-]
9 resStab ► [{heparin+, hemothorax} ► [heparin- → rem]]

```

Example 5.1: A list of all dependencies and rules of the ES in Figure 20. In order to keep this readable we use abbreviations: aRDS for acute respiratory distress syndrome, heparin+ and heparin- for startHeparin and stopHeparin, rem for remainder, and resStab for respiratory stabilisation.

study was also modelled with DCR-Graphs in [13] with some other minor issues: *“Despite their limitations, I still found DCR graphs to be suitable to model the case study if the level of abstraction is kept reasonable.”* (page 37) Since both formalisms were more or less capable of modelling the case study, we compared these formalisms in general. In Chapter 4 we showed that any basic DCR-Graph can be encoded into a HDES.

5.2 Requirements for Further Applicability

In order to model bigger case studies certain features are strongly needed for the HDES formalism. We assume that the first three features can be added as syntactic sugar while for the later ones new theory is needed.

5.2.1 Syntactic Sugar

We assume that the following three features – grouping, bundling of dynamics, and urgent events – can be added to the HDES formalism without creating new semantical concepts but as syntactic sugar.

Grouping In many cases it would be really useful to have some grouping of events that can be used in the rules. For example, the group S defined as the set of all events during the surgery together with some rules like $a \blacktriangleright [p \rightarrow S]$ containing this group S . The meaning of this rule is that a precondition p is added to any event in the surgery. Another example for a group is the event remainder in Figure 21.

In this approach grouping is a kind of compressing the rules in a still intuitive way. Since we already can use modifier sets, we only need to adapt the definitions for the causality rules and to add a group list to the HDEs.

Bundling of Dynamics In the modelling of the case study we encountered the situation where we wanted to turn a rule with all of its effects off and possibly later on again. We had the rule $\{\text{startHeparin}, \text{hemothorax}\} \blacktriangleright [\text{stopHeparin} \rightarrow \text{remainder}]$ indicating to stop the heparin treatment in case of a hemothorax (see Figure 19). This was overruled in a case of an acute respiratory distress syndrome (ARDS) until a respiratory stabilisation (see Figure 20) could be achieved. Since the events startHeparin and hemothorax could both happen before ARDS, it is not sufficient to just have the single rule $\text{ARDS} \triangleright [\{\text{startHeparin}, \text{hemothorax}\} \blacktriangleright [\text{stopHeparin} \rightarrow \text{remainder}]]$. We also need the rule $\text{ARDS} \triangleright [\text{stopHeparin} \rightarrow \text{remainder}]$ and, on the re-insertion of the rule, we should also be careful if we should re-insert the dependency or not. In Figure 21 the dash-dotted arrows denote the deletion and insertion of the rule $\{\text{startHeparin}, \text{hemothorax}\} \blacktriangleright [\text{stopHeparin} \rightarrow \text{remainder}]$ and if needed of the dependency $\text{stopHeparin} \rightarrow \text{remainder}$. Since they bundle certain dynamic rules, this is called bundling of dynamics. Achieving this the models will become clearer and easier to understand.

Urgent events (resp. Priorities) The notion of urgent events is needed in various scenarios. For example, in Figure 21 the event stopHeparin becomes urgent after startHeparin and hemothorax . That means it should precede any event in the group remainder. The general scenario derived from the example above: We have a trigger that makes an event urgent with respect to a group (i.e. it should precede any event in this group). We have $t \blacktriangleright [u \rightarrow G]$. That can be translated to the set of rules parametrised by the group $\{t \blacktriangleright [u \rightarrow G] \mid g \in G\}$. In such a way the use of the remainder event in the modelling would not effect the compositionality as it did in [22].

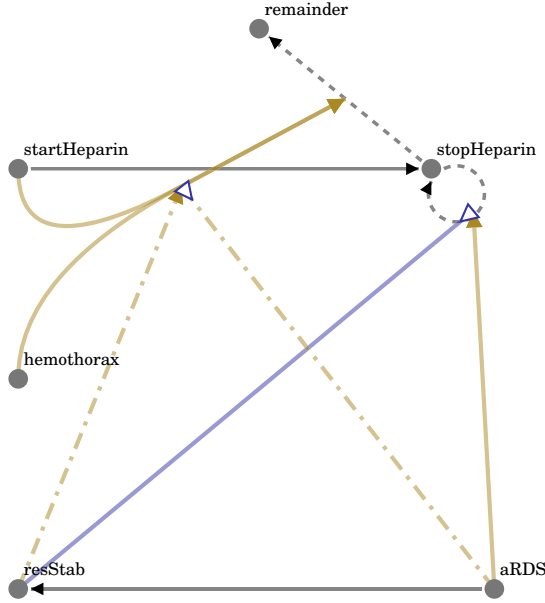


Figure 21: Example for bundling of dynamics. Here the dash-dotted arrows denote the bundling of dynamics explained above.

Urgency can be seen as a special case of priority. In the above formulation the urgent event u would get a higher priority than all those events not in the group G . With a similar approach as for the urgency even the more general notion of priority can be encoded in the base formalism which means that both urgency and priority are only syntactic sugar and no new features.

There is a work in which priority, as a binary relation, was added to various event structures [5]. Whenever there is a pair (e, d) in this priority relation d must precede e if both are concurrently enabled.

5.2.2 Proper Extensions

For the here mentioned two extensions we do not see any possibility to add them as syntactic sugar. We strongly expect that new semantic concepts are needed to implement them to the HDES formalism.

Repeatable events As in the construction of the HDES encoding of a DCR-Graph (cf. Definition 4.18), often repeatable events would make life easier and the model more concise. As mentioned in the last point of Remark 4.28, a similar approach to the evolutionary DCEs as in [1] sounds promising. It is also of interest to us whether it can be achieved with a smaller extension than that since we do not consider arbitrary evolutions but very regular ones. This means up on execution of an event insert a new copy of this event, remove all rules that contain the old event, but re-include them with the new version of this event.

Abstraction/Refinement In order to view a process on a suitable level of abstraction, zooming mechanisms are needed. This means to have some way of zooming out (abstraction) and zooming in (refinement) as an ingredient of HDESs. In the case study, for example we could look at the surgery as an atomic event (very high level), or at a low level model where all actions performed during the surgery by any participant are modelled.

The idea of refinement in traditional ESs is well known. An action can be replaced by a more complicated processes, for example in [10]. This approach is quite intuitive and elegant, yet it is rather complicated to adapt this in our dynamic ESs. Even for the DCEs it is not clear how a modifier could or should be refined. For example, if we have $a \blacktriangleright [c \rightarrow t]$ and want to refine the adder a of the dependency c to t with the very simple process $a_1 \rightarrow a_2$, it is unclear which of these events a_1 and a_2 (or both together) should be the adder in a refined structure. It is also unclear which event should be targeted by the dynamics if we refine t instead of a . It becomes a serious problem if we consider a binary conflict $a \# b$ modelled as mutual disabling $a \blacktriangleright [b \rightarrow b]$ and $b \blacktriangleright [a \rightarrow a]$. Now, a and b are not allowed to happen in parallel, or after each other. If we refine a and b , we must take care of not allowing for both refinements to start and deadlock as they would block each other at some point since in the original model such a behaviour was impossible.

There was some effort on solving these (and even more complex issues) problems in order to define refinements for DCEs. But up to now we are not aware of sufficient results. Thus, we relaxed the requirements a little bit and discussed the notion of grouping in a previous paragraph.

5.2.3 Closing Remarks

There are obviously other dimensions in which the base formalism HDESs can be enhanced such as data or timing. Such extension were also developed for other

formalisms but only after they got a solid base formalism (e.g. timed automata, or timed Petri nets). Since HDESSs are a new approach we do not think they are finalised in their current state. Thus, timing and data are not mentioned in the list above.

For DCR-Graphs some of the previously mentioned features are already developed (as in [15] the milestone relation which can be seen as urgency with respect to a group of events, or quite recently in [7] refinement, timing, and data).

6 Conclusion

In this thesis we presented Higher-Order Dynamic-Causality ESs (HDESs) as a declarative causality-based modelling formalism for highly dynamic processes. In order to obtain the HDESs we generalised the Dynamic-Causality ESs (DCESSs) [2]. We based this generalisation on theoretical needs caused by an incomparability result (as in [2]) to the ESs for resolvable conflicts (RCESSs) of [12] and the practical needs of a case study [22]. After a revised definition of the HDESs (in comparison to the original definition in [16]), we presented all results with respect to the expressive power of HDESs (many previously unpublished) and a useful web tool for simulating and exploring HDES.

Now we summarise the main contributions (in Section 6.1) and we state research questions, which we want to address in the future (in Section 6.2).

6.1 Main Contributions

In this thesis we revisited the HDESs introduced in [16]. We presented an improved version of the HDES definition (in Section 3.1) and the result (in Subsection 3.5.3) that HDESs are strictly more expressive than a specific class of configuration structures (this implies that HDESs are strictly more expressive than the RCESSs mentioned above).

There is a web tool in Section 3.2 in which HDESs (and also DCESSs) can be defined and simulated. This web tool can be reached at <http://hdes.mtv.tu-berlin.de/>.

Further, in Subsection 3.4.2, we introduced the event transition systems (ETSs) a specific kind of labelled transition systems (LTSs). We proved that the ETSs are exactly as expressive as HDESs, which means for any ETS exists a HDESs with the same behaviour, and also for any HDES exists a ETS with the same behaviour. With this result we have simplified future comparison with HDESs since they do not need to be based on the special HDES semantics but on the very general ETSs semantics.

We also used this approach to justify the use of our rule update strategy. We introduced an alternative rule update strategy (in Subsection 3.4.1) and proved that HDESs equipped with this strategy are exactly as expressive as ETSs. With this method we have shown that the expressive power of HDESs does not depend on the choice between the original and the alternative rule update strategy. Other rule update strategies exist which yield different results (e.g. a rule update strategy that simply deletes all rules).

In Section 3.5 we showed various results with respect to the expressive power of HDESs: First, we showed that HDESs are indeed generalising DCEs (in Subsection 3.5.1). Second, we proved that set-based dynamics and higher-order dynamics are orthogonal extensions, which means neither can be expressed by the other (in Subsection 3.5.2). Third, we compared HDESs to transition systems as mentioned above. And fourth, we discussed some results if we fix the level of dynamicity.

In Chapter 4 we showed that any basic DCR-Graph [8] can be encoded into a slightly extended HDESs (i.e. HDESs with an acceptance criterion and a labelling of the events). The other direction is assumed to hold but this hypothesis is not yet proven and subject to future work. One main difference between HDES and DCR-Graphs is the different level of abstraction. In HDES everything is based on dynamic exclusion and inclusion of dependencies, where in DCR-Graphs a lot more basic relations are around and the dynamics are based on exclusion and inclusion of events.

6.2 Open Problems and Future Work

There are quite a few open problems that we would like to address in the future:

In Subsection 3.5.4 we presented Conjecture 3.67 stating that the expressive power of HDESs is increasing monotonously with the order of dynamicity. It still needs to be determined if this conjecture is true. A related question is if there is any bound on the order of dynamicity of a HDES encoding with the other rule update strategy. Currently there is only a bound in the length of the longest path of the corresponding transition system.

In the previous chapter we presented a list of features that are required for HDES to become more applicable. All of them are subject to future research. Another feature we would like to investigate is the naming of rules. Right now rules could be considered to be labelled by themselves and it is only possible to have at most one copy of any rule. With unique names we could have multiple

copies of the same rule with different names where other rules interact with this rule based on the names. It would become possible to do local transformations on a HDES. Until now this is more or less impossible because we do not know how the other rules of the HDES might interact with our modification. If we create new names for the modified rules, we are sure only to have the intended interactions.

In Chapter 4 we presented an encoding of basic DCR-Graphs into slightly extended HDESs. We assume that also an encoding of the extended HDESs into DCR-Graphs exists.

The synchronisation pattern (as in Subsection 3.3.2), the bundling of dynamics, the notions of grouping, urgent events, and repeatable events are design patterns that are useful for modelling with HDES. We would like to further investigate if there are even more patterns. We also are interested if similar patterns can be found and are of potential use for other declarative causality-based modelling formalisms (e.g. DCR-Graphs).

Bibliography

- [1] Youssef Arbach. “On the Foundations of Dynamic Coalitions: Modeling Changes and Evolution of Workflows in Healthcare Scenarios”. PhD thesis. Technische Universität Berlin, 2016. ISBN: 978-3-7983-2857-0.
- [2] Youssef Arbach, David Stefan Karcher, Kirstin Peters, and Uwe Nestmann. “Dynamic Causality in Event Structures”. In: *Logical Methods in Computer Science* Volume 14, Issue 1 (Feb. 2018). DOI: [10.23638/LMCS-14\(1:17\)2018](https://doi.org/10.23638/LMCS-14(1:17)2018).
- [3] Youssef Arbach, David Karcher, Kirstin Peters, and Uwe Nestmann. “Dynamic Causality in Event Structures”. In: *Formal Techniques for Distributed Objects, Components, and Systems: 35th IFIP WG 6.1 International Conference, FORTE 2015, Held as Part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2-4, 2015, Proceedings*. Vol. 9039. LNCS. Springer, 2015, pp. 83–97.
- [4] Youssef Arbach, David Karcher, Kirstin Peters, and Uwe Nestmann. “Dynamic Causality in Event Structures (Tech. Report)”. In: (2015). URL: <http://arxiv.org/abs/1504.00512>.
- [5] Youssef Arbach, Kirstin Peters, and Uwe Nestmann. “Adding Priority to Event Structures”. In: *Proceedings of EXPRESS/SOS*. Ed. by Johannes Borgström and Bas Luttik. Vol. 120. EPTCS. 2013, pp. 17–31. DOI: [10.4204/EPTCS.120](https://doi.org/10.4204/EPTCS.120).
- [6] Ilaria Castellani and Guo Qiang Zhang. “Parallel product of event structures”. In: *DAIMI Report Series* 18.285 (1989).
- [7] Søren Debois and Thomas T. Hildebrandt. “The DCR Workbench: Declarative Choreographies for Collaborative Processes”. English. In: *Behavioural Types: from Theory to Tools*. Ed. by Simon Gay and António Ravara. River Publishers, June 2017, pp. 99–124. ISBN: 978-87-93519-82-4.

- [8] Søren Debois, Thomas T. Hildebrandt, and Tijs Slaats. “Concurrency and Asynchrony in Declarative Workflows”. In: *Business Process Management: 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 – September 3, 2015, Proceedings*. Springer, 2015, pp. 72–89.
- [9] R.J. van Glabbeek. “On the expressiveness of higher dimensional automata”. In: *Theoretical Computer Science* 356.3 (2006). Expressiveness in Concurrency, pp. 265–290. ISSN: 0304-3975. DOI: <https://doi.org/10.1016/j.tcs.2006.02.012>.
- [10] Rob van Glabbeek and Ursula Goltz. “Refinement of actions in causality based models”. In: *Stepwise Refinement of Distributed Systems Models, Formalisms, Correctness: REX Workshop, Mook, The Netherlands May 29 – June 2, 1989 Proceedings*. Springer, 1990, pp. 267–300.
- [11] Rob van Glabbeek and Gordon Plotkin. “Configuration Structures, Event Structures and Petri Nets”. In: *Theoretical Computer Science* 410.41 (2009). Festschrift for Mogens Nielsen’s 60th Birthday, pp. 4111–4159.
- [12] Rob van Glabbeek and Gordon Plotkin. “Event Structures for Resolvable Conflict”. In: *Mathematical Foundations of Computer Science 2004: 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22-27, 2004. Proceedings*. Vol. 3153. LNCS. Springer, 2004, pp. 550–561.
- [13] Tim Malte Gräfe. “On the Utility and Usability of DCR Graphs to Model Dynamic Processes in a Clinical Case Study”. Bachelor Thesis in Computer Science. Technische Universität Berlin, 2016.
- [14] Thomas T. Hildebrandt and Raghava Rao Mukkamala. “Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs”. In: *PLACES*. 2010.
- [15] Thomas T. Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. “Nested Dynamic Condition Response Graphs”. In: *Fundamentals of Software Engineering: 4th IPM International Conference, FSEN 2011, Tehran, Iran, April 20-22, 2011, Revised Selected Papers*. Vol. 7141. LNCS. Springer, 2011, pp. 343–350.
- [16] David S. Karcher and Uwe Nestmann. “Higher-Order Dynamics in Event Structures”. In: *Theoretical Aspects of Computing - ICTAC 2015: 12th International Colloquium, Cali, Colombia, October 29-31, 2015, Proceedings*. Vol. 9399. LNCS. Springer, 2015, pp. 258–271.

-
- [17] Rom Langerak. “Transformations and Semantics for LOTOS”. PhD thesis. Universiteit Twente, 1992.
- [18] Morten Nielsen, Gordon Plotkin, and Glynn Winskel. “Petri nets, event structures and domains”. In: *Semantics of Concurrent Computation: Proceedings of the International Symposium, Evian, France, July 2–4, 1979*. Springer, 1979.
- [19] Carl Adam Petri. “Kommunikation mit Automaten”. PhD thesis. Universität Hamburg, 1962.
- [20] Vaughn Pratt. “Modeling Concurrency with Geometry”. In: *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’91. Orlando, Florida, USA: ACM, 1991, pp. 311–322. ISBN: 0-89791-419-8. DOI: [10.1145/99583.99625](https://doi.org/10.1145/99583.99625).
- [21] Tijs Slaats, Raghava Rao Mukkamala, Thomas T. Hildebrandt, and Morten Marquard. “Exformatics Declarative Case Management Workflows as DCR Graphs”. In: *Business Process Management*. Ed. by Florian Daniel, Jianmin Wang, and Barbara Weber. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 339–354. ISBN: 978-3-642-40176-3.
- [22] Jonay Trénous. “On the Utility and Usability of Event Structures to Model Dynamic Processes in a Clinical Case Study”. Bachelor Thesis in Computer Science. Technische Universität Berlin, 2015.
- [23] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. ISBN: 3540735216.
- [24] Glynn Winskel. “An introduction to event structures”. In: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*. Vol. 354. LNCS. Springer, 1989, pp. 364–397.
- [25] Glynn Winskel. “Events in computation”. PhD thesis. University of Edinburgh, 1980.

Schriftenreihe **Foundations of Computing**

Hrsg.: Prof. Dr. Stephan Kreutzer, Prof. Dr. Uwe Nestmann, Prof. Dr. Rolf Niedermeier

ISSN 2199-5249 (print)

ISSN 2199-5257 (online)

- 01: Bevern, René van: Fixed-Parameter Linear-Time Algorithms for NP-hard Graph and Hypergraph Problems Arising in Industrial Applications.** - 2014. - 225 S.
ISBN 978-3-7983-2705-4 (print) EUR 12,00
ISBN 978-3-7983-2706-1 (online)
- 02: Nichterlein, André: Degree-Constrained Editing of Small-Degree Graphs.** - 2015. - xiv, 225 S.
ISBN 978-3-7983-2761-0 (print) EUR 12,00
ISBN 978-3-7983-2761-7 (online)
- 03: Brederbeck, Robert: Multivariate Complexity Analysis of Team Management Problems.** - 2015. - xix, 228 S.
ISBN 978-3-7983-2764-1 (print) EUR 12,00
ISBN 978-3-7983-2765-8 (online)
- 04: Talmon, Nimrod: Algorithmic Aspects of Manipulation and Anonymization in Social Choice and Social Networks.** - 2016. - xiv, 275 S.
ISBN 978-3-7983-2804-4 (print) EUR 13,00
ISBN 978-3-7983-2805-1 (online)
- 05: Siebertz, Sebastian: Nowhere Dense Classes of Graphs.** Characterisations and Algorithmic Meta-Theorems. - 2016. - xxii, 149 S.
ISBN 978-3-7983-2818-1 (print) EUR 11,00
ISBN 978-3-7983-2819-8 (online)
- 06: Chen, Jiehua: Exploiting Structure in Computationally Hard Voting Problems.** - 2016. - xxi, 255 S.
ISBN 978-3-7983-2825-9 (print) EUR 13,00
ISBN 978-3-7983-2826-6 (online)
- 07: Arbach, Youssef: On the Foundations of dynamic coalitions.** Modeling changes and evolution of workflows in healthcare scenarios - 2016. - xv, 171 S.
ISBN 978-3-7983-2856-3 (print) EUR 12,00
ISBN 978-3-7983-2857-0 (online)
- 08: Sorge, Manuel: Be sparse! Be dense! Be robust!** Elements of parameterized algorithms - 2017. - xvi, 251 S.
ISBN 978-3-7983-2885-3 (print) EUR 13,00
ISBN 978-3-7983-2886-0 (online)
- 09: Dittmann, Christoph: Parity games, separations, and the modal μ -calculus** - 2017. - x, 274 S.
ISBN 978-3-7983-2887-7 (print) EUR 13,00
ISBN 978-3-7983-2888-4 (online)

Event Structures with Higher-Order Dynamics

Event Structures were introduced in 1979 as a formal model to connect the theory of Petri nets and domain theory. Originally they consisted of atomic non-repeatable events, a binary causal dependency relation, and a binary conflict relation between those events. For a long time various extensions of the original formalism were used to define semantics for other structures such as classes of Petri nets and process calculi.

In this thesis the Event Structures (ESs) are considered solely as a declarative modelling tool than as a formalism to define semantics for other structures. In order to model highly dynamic real-world processes (i.e. processes in which occurrences of events may change the dependencies of other events) with a concise model the dynamics must be an inherent part of the modelling formalism.

Therefore, the Higher-Order Dynamic-Causality ESs (HDESSs) were introduced. They consist of a finite set of atomic non-repeatable events, a causal dependency relation between these events, and a rule-based formalism for events to change the dependencies and the existing rules.

This formalism is studied with the respect to its expressive power in comparison to transition systems, some subclasses of HDESSs, and to Dynamic Condition Response Graphs (DCR-Graphs).

ISBN 978-3-7983-2995-9 (print)

ISBN 978-3-7983-2996-6 (online)



ISBN 978-3-7983-2995-9



<http://verlag.tu-berlin.de>