Foundations of computing Volume 7

Youssef Arbach

On the Foundations of Dynamic Coalitions

Modeling Changes and Evolution of Workflows in Healthcare Scenarios

Universitätsverlag der TU Berlin

Youssef Arbach

On the Foundations of Dynamic Coalitions

Modeling Changes and Evolution of Workflows in Healthcare Scenrios

Die Schriftenreihe *Foundations of Computing* der Technischen Universität Berlin wird herausgegeben von:
Prof. Dr. Rolf Niedermeier,
Prof. Dr. Uwe Nestmann,
Prof. Dr. Stephan Kreutzer

Foundations of Computing | 07

Youssef Arbach

On the Foundations of Dynamic Coalitions

Modeling Changes and Evolution of Workflows in Healthcare Scenarios

Universitätsverlag der TU Berlin

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, 2016
http://verlag.tu-berlin.de
Fasanenstr. 88, 10623 Berlin
Tel.: +49 (0)30 314 76131 / Fax: -76133
E-Mail: publikationen@ub.tu-berlin.de

Zugl.: Berlin, Techn. Univ., Diss., 2015
1. Gutachter: Prof. Dr. Uwe Nestmann
2. Gutachter: Prof. Thomas Hildebrandt, PhD
3. Gutachter: Prof. Dr. Mathias Weske
Die Arbeit wurde am 14. September 2015 an der Fakultät IV unter Vorsitz von Prof. Dr. Sabine Glesner erfolgreich verteidigt.

Druck: docupoint GmbH
Satz/Layout: Youssef Arbach

Umschlagfoto:
Allie_Caulfield | https://www.flickr.com/photos/wm_archiv/26315161296 | CC BY 2.0
https://creativecommons.org/licenses/by/2.0/

ISBN 978-3-7983-2856-3 (print)
ISBN 978-3-7983-2857-0 (online)

ISSN 2199-5249 (print)
ISSN 2199-5257 (online)

Zugleich online veröffentlicht auf dem institutionellen Repositorium der Technischen Universität Berlin:
DOI 10.14279/depositonce-5406
http://dx.doi.org/10.14279/depositonce-5406

Zusammenfassung

Dynamische Koalitionen bezeichnen eine temporäre Zusammenarbeit zwischen verschiedenen Entitäten zum Erreichen eines gemeinsamen Ziels. Ein Schlüsselaspekt, der dynamische Koalitionen von statischen Koalitionen unterscheidet ist die *dynamische Mitgliedschaft*, durch die Mitglieder neu hinzukommen oder die Koalition verlassen können, nachdem sie entstanden ist. Diese Arbeit studiert Workflows in dynamischen Koalitionen durch eine Analyse ihrer Eigenschaften, das Herausstellen ihrer einzigartigen Charakteristika und Ähnlichkeiten zu anderen Workflows und durch eine Untersuchung ihrer Beziehung zu dynamischer Mitgliedschaft. In diesem Sinne nutzen wir das formale Model der *Ereignisstrukturen* und erweitern es, um Fallstudien aus der Medizin angemessen zu modellieren. Ereignisstrukturen erlauben sowohl eine generelle Workflow-Modellierung als auch eine Darstellung von Eintritts- und Austrittsereignissen von Mitgliedern.

Zu diesem Zweck erweitern wir Ereignisstrukturen zuerst um *Dynamische Kausalität*, um die dynamische Natur von dynamische Koalitionen abzubilden. Dynamische Kausalität erlaubt bestimmten Ereignissen die kausalen Abhängigkeiten anderer Ereignisse in der selben Struktur zu verändern. Danach untersuchen wir die Ausdrucksstärke der resultierenden Ereignisstrukturen und zeigen, dass sie nur eine spezifische Art der Veränderung abbilden, die sogenannten geplanten Veränderungen. Als Zweites präsentieren wir *evolutionäre Ereignisstrukturen* um ad-hoc- und unvorhergesehene Veränderungen zu unterstützen, die zur Modellierung der Fallstudien benötigt werden. Evolotionäre Ereignisstrukturen verbinden verschiedene Ereignisstrukturen durch einer Relation, welche eine Veränderung einer Ereignisstruktur während eines Ablaufs erlaubt. Wir ziehen verschiedene Ansätze der Modellevolution in Betracht und untersuchen ihre Äquivalenzen. Des Weiteren zeigen wir, dass in unserer Betrachtung der Evolution dynamischer Koalitionen die Geschichte eines Workflows erhalten bleiben muss und wir ermöglichen das Extrahieren von Veränderungen einer Evolution, um *Prozesslernen* zu unterstützen. Drittens: Um die Ziele von dynamischen Koalitionen abzubilden, fügen wir den evolutionären Ereignisstrukturen ein Ziel hinzu, repräsentiert durch eine Menge von erreichbaren Ereignissen. Diese Erweiterungen erlauben es, sowohl die Änderungen und Evolutionen, die von Mitgliedern verursacht werden, als auch die Beiträge der Mitglieder zur Zielerreichung durch deren Eintritts- und Austrittsereignisse zu untersuchen. Schlussendlich stellen wir viele Modellierungseigenschaften der dynamichen Koalitionen dar, welche von den Fallstudien aus der Medizin benötigt werden und unabhängig von der Natur der dynamische Koalitionen sind, wie z.B. Zeitmessung. Wir untersuchen die Literatur zu Ereignisstrukturen bezüglich Unterstützung für solche Eigenschaften und stellen fest, dass das Konzept *Priorität* in Ereignisstrukturen fehlt. Daher fügen wir Priorität zu verschiedenen Ereignisstrukturen aus der Literatur hinzu. Des Weiteren untersuchen wir die Beziehungen von Priorität zu konjunktiver Kausalität, disjunktiver Kausalität, kausaler Uneindeutigkeit und verschiedenen Formen von Konflikten.

Im Vergleich zu *adaptiven Workflows*, welche sich mit der Evolution von Workflows beschäftigen, die als Reaktion auf Veränderungen entsteht, wie z.B. Änderungen in der Geschäftsumgebung oder Ausnahmen, zeigt diese Arbeit, dass Workflows in dynamischen Koalitionen nicht nur *adaptiv* sondern auch *zielorientiert* sind. Außerdem fügt sie einen zusätzlichen Auslöser für Evolution in Workflows hinzu, welcher ausschließlich dynamischen Koalitionen zu eigen ist: das Hinzukommen neuer Mitglieder, welche zur Zielerreichung der dynamischen Koalition beitragen.

Zuletzt trägt diese Arbeit bei, die Lücke in der Modellierung zwischen Theorie und Domänenexperten zu schließen, in dem sie eine Schritt-für-Schritt Modellierung ermöglicht, welche regelmäßig in der Medizin und anderen Bereichen angewandt wird.

Abstract

Dynamic Coalitions denote a temporary collaboration between different entities to achieve a common goal. A key feature that distinguishes Dynamic Coalitions from static coalitions is *Dynamic Membership*, where new members can join and others can leave after a coalition is set. This thesis studies workflows in Dynamic Coalitions, by analyzing their features, highlighting their unique characteristics and similarities to other workflows, and investigating their relation with *Dynamic Membership*. For this purpose, we use the formal model of *Event Structures* and extend it to faithfully model scenarios taken as use cases from healthcare. Event Structures allow for workflow modeling in general, and for modeling *Dynamic Membership* in Dynamic Coalitions as well through capturing the *join* and *leave* events of members.

To this end, we first extend Event Structures with *Dynamic Causality* to address the dynamic nature of Dynamic Coalitions. *Dynamic Causality* allows some events to change the causal dependencies of other events in a structure. Then, we study the expressive power of the resulting Event Structures and show that they contribute only to a specific kind of changes in workflows, namely the pre-planned changes. Second, we present *Evolving Event Structures* in order to support ad-hoc and unforeseen changes in workflows, as required by the use cases. *Evolving Event Structures* connect different Event Structures with an *evolution* relation which allows for changing an Event Structure during a system run. We consider different approaches to model evolution and study their relation. Furthermore, we show that the history of a workflow should be preserved in our case of evolution in Dynamic Coalitions, and we allow for extracting changes from an evolution to support *Process Learning*. Third, to capture the goals of Dynamic Coalitions, we equip *Evolving Event Structures* with constraints concerning the reachability of a set of events that represents a goal. The former extensions allow for examining the changes and evolutions caused by members, and examining members' contributions to goal satisfaction, through their *join* and *leave* events. Finally, we highlight many modeling features posed as requirements by the domain of our Dynamic-Coalition use cases, namely the healthcare, which are independent from the nature of Dynamic Coalitions, e.g. timing. We examine the literature of Event Structures for supporting such features, and we identify that the notion of *Priority* is missing in Event Structures. To this end, we add *Priority* to various kinds of Event Structures from the literature. Furthermore, we study the relation between priority on one side, and conjunctive causality, disjunctive causality, causal ambiguity and various kinds of conflict on the other side.

Comparing to *Adaptive Workflows*, which are concerned with evolutions of workflows that occur as a response to changes, e.g. changes in the business environment or exceptions, this thesis shows that Dynamic-Coalition workflows are not only *Adaptive* but also *Goal-Oriented*. Besides, it adds one extra trigger for evolution in workflows—that is unique to Dynamic Coalitions—namely the join of new members who contribute to goal satisfaction in a Dynamic Coalition.

Finally the thesis contributes to bridging the gap in modeling between theory and domain experts by supporting step-by-step modeling applied regularly in healthcare and other domains.

Dedication

To my father who worked hard for me to reach the doctoral study... that was a lifetime.

To mom with all her love, and the great words

To *Bouran*, who has been more than a sister, and to the days we spent together

To *Ivan* my sister, the little angel, with her dreams that kept me alive

To *Elias* my brother, the little prince, with his nobility

To my wife *Hala*, the greatest friend, with her patience and support, and to the little girl we are waiting for... I love you both

To all friends and relatives who have always wished me the best

To Syria, the home of all times, which was going through hard times while I was proceeding with my doctoral study

Acknowledgements

I would like to express my special appreciation and thanks to my advisor Prof. Dr. Uwe Nestmann. He has been a tremendous mentor for me. I would like to thank him for encouraging my research and for allowing me to grow as a research scientist. He has always been positive. I am proud to know you Uwe.

I would also like to thank my committee members, Prof. Mathias Weske, Prof. Thomas Hildebrandt and Prof. Sabine Glesner for serving as my committee members, and for the brilliant comments and suggestions they gave. Thanks to you.

A special thank to Prof. Wolfgang Reisig for his advices with all his experience. A great thank to Dr. Mehmet Gövercin for his hospitality and support in healthcare.

Thanks to all members of the *Models and Theory of Distributed Systems*, among of whom I mention Dr. Kirstin Peters and David Karcher, who supported me intensively in the last years, especially in publications.

I would like to thank the *SOAMED* Research Training Group for offering me the chance and support to do this research over three years. *SOAMED* gave me the chance to dedicate all my time to research.

At last but not at least, I would like to thank my great friend, Nadim Sarrouh, for his support and the nice moments we had together, and to thank all members of *SOAMED* for the friendly atmosphere we had.

Contents

1. Introduction and Motivation

1.1. Introduction

Collaboration is a major concept in human life. The advance in technology and networking allowed organizations and individuals to build quicker collaborations to reach certain goals. Additionally, it allowed technical entities e.g. robots and sensors to build collaboration networks and coalitions. The term *Dynamic Coalitions (DCs)* denotes a temporary collaboration between different entities to achieve a common goal [68]. DCs can be seen in daily life, e.g. in economy between different organizations to penetrate a new market, in military such as the NATO, in healthcare between multiple doctors to treat a patient, in catastrophes between different institutes after an earthquake, etc.

The notion of a DC has evolved over decades, and varied between different fields and studies under various names. *Cooperative Game Theory*, for instance, searches the possibility of building coalitions among players in order to strengthen players' positions in a game [67]. *Distributed Problem Solving* [34] focuses on the idea of how multiple problem solvers may work optimally to solve a problem, which none of them can solve individually. *Virtual Organizations* emerged for Grid Computing [26] to build large-scale resource sharing environment to achieve any given goal. Recently, the term DC appeared to emphasize on *dynamicity*.

What gives DCs their dynamic nature is dynamic membership, where members can join and leave after the coalition is set [46, 21]. This is considered a key feature of DCs, which distinguishes DCs from static coalitions. Dynamic membership shows the interaction of a DC with time. Therefore, it describes how a DC has been formed or created.

Many perspectives and problems can be investigated in DCs, e.g. information transformation [53], privacy and access rights [68], time, etc. According to Bryans [16], these different perspectives are called *Dimensions* of Dynamic Coalitions.

1.1.1. Case Study: Clinical Dynamic Coalitions

We investigate workflows that are based on human collaboration in the healthcare sector. Here we consider one example, taken from *Deutsches Herzzentrum Berlin (DHZB)*. This example will be a use case for the rest of this thesis.

> *A cardiac patient had a heart surgery, and was transferred afterwards to the Cardiac Intensive Care Unit (CICU), unconscious. The goal was to warm her up and then to wake her up before she is discharged. While waking her up, the ventilation tube should be disconnected, and severe medication, e.g. circulation support medication, should be stopped before discharging her. The wake-up process was not going well due to a heart failure. As a result, the patient was in need of an external blood pump. The therapy plan was adapted then such that a small surgery was needed to implant the pump. Accordingly, the process of waking up was interrupted, but resumed again. Next morning, during the doctor's visit, the doctor discovered that the patient had a cold-leg problem, which disabled the warm-up process. The doctor decided to call a specialist, namely a surgeon, and to invite her to join. They examined together the patient's situation, and discussed what to do. They decided finally to make an angiography. The result of the angiography showed a closed vessel in the leg, which made them collaborate with a new specialist. The decision of the negotiation was made to adapt the therapy such as that a surgery was to be performed in order to open the closed vessel. Afterwards, the surgery was performed, the warm-up process was enabled again, and the patient was successfully discharged.*

It can be seen as a DC, that was held between different nurses and doctors in CICU. The coalition was dynamic in the sense that a new member, e.g. the surgeon, joined the coalition after it was set. The goal of this DC was to discharge the patient successfully, by proceeding through all the intermediate steps mentioned above, e.g. warm up. Such a DC was temporary and ended by satisfying its goal in sending the patient back.

1.1.2. The Adaptive and Goal-Oriented Nature of Dynamic-Coalition Workflows

The former example can be seen from the perspective of adaptive workflows, too. The initial workflow was to warm the patient up, then wake her up, disconnect the tube and stop severe medication. It changed afterwards to adapt to the heart-failure by adding an extra surgery to implant a blood pump, as a predecessor for the wake-up process. Later on, the workflow changed again to adapt to the cold-leg problem which was an exception for the doctors in CICU. After adding a new surgery to open the closed vessel, doctors proceeded to discharge the patient, and the goal was satisfied.

The adaptation of the workflow to include the surgery as a response to the discovery of the cold-leg was made by the newly joining surgeon. This is an example of how new members in a DC might contribute to changing the workflow of the DC, even though it was caused by another event, that is unforeseen, namely the discovery of the cold leg. Other examples of DCs, like ad-hoc DCs that are formed as a response to an acute need [68], start with initial workflows that would never satisfy their goals, where other members are waited and invited to join and contribute to the work, until the goal is satisfied.

Additionally, DCs are goal-driven [59]. The goal in our use case was to successfully discharge the patient through all the intermediate steps, e.g. warming up, waking up, . . . etc. The initial workflow was designed such that the goal is to be reached at the end. Besides, when the workflow evolved due to exceptions which made the goal unreachable, the evolution itself was made such that the goal became reachable again.

The two features of *adaptivity* and *goal orientation* form the main perspectives through which we address and formalize DC workflows.

1.1.3. The Need for Formal Modeling

Doctors in the healthcare sector emphasize the point that perfect plans, which stay unchanged during the treatment of a patient, do not exist. Rather, hospital processes evolve depending on each case. Besides, the join of new doctors might lift a running process from one scheme to another. The sector would like to organize such workflows so that no undesired scenarios can take place.

Formal methods help avoiding ambiguity, and provide a very precise means of communication between modelers and designers, that does not allow for different interpretations. Besides, one basic advantage of formal modeling is verification which checks whether the modeled workflow might have any undesired path during runtime. Based on Event Structures that we apply here to formalize workflows of DCs, this can be done by checking whether a given sequence or set of events represents a valid system run of the given workflow. A system run must obey the defined causality or flow, as well as conflict-freeness and other constraints. Verification mechanisms, e.g. *Model Checking* [33], can be applied to check whether a given property is maintained by all possible runs of a system.

Avoiding undesired system runs becomes even more complex when the workflow itself changes, or evolves, while it is running. In other words, it might be more complex when the rules used to judge system runs do change themselves. Evolution of workflow might lead to a conflict between the history that precedes the evolution, and the regulations embedded in the new workflow, for example. Furthermore, when considering goals, system runs need to proceed in a way they satisfy their desired goals at the end.

Next to basic verification mechanism, other studies could be applied when using qualitative and quantitative extensions of workflow models. Examples of such extensions are timed [43] and stochastic Event Structures [79]. Predictions could be made through means of a probabilistic workflow model [43]. A sequence of events in a timed ES could be checked to respect the timeouts given for events, or the real-time criteria of the system.

1.2. Problem Statement

A workflow of a DC holds and reflects all its unique points and features, such as dynamicity, evolution and goal orientation. Studies on DCs in general address the problem of how members of coalitions are selected, based on resources they offer to the coalition [57, 87, 59], as well as the problem of building structures for coalitions based on the value each agent provides [59, 52]. None of the studies so far addresses DCs from a process perspective, nor shows what the unique features of DC workflows are, nor how they are affected by the membership. This point has been an open question in the literature of DCs [16, 20, 68, 45].

On the other hand, changes in adaptive workflows are usually triggered by changes in the business environment, such as management decisions, or by exceptions that are case-specific [80, 35, 61]. We claim that in case of DC workflows—as our various examples of DCs show—changes can be additionally triggered by membership, through the join and leave of new and old members respectively. This is due to the Distributed-Problem-Solving nature of DCs, where members or agents, do contribute to solve the problem or reach the goal [34]. This might yield in changes of a running process once new members join the coalition. Such an effect of membership on the dynamicity of DC Workflows was not addressed formally before. Here in this work we show it by capturing the changes events lead to in the workflow, and by modeling membership through events of members' join and leave.

The claim that members or agents of coalitions affect their workflow has already been supported by Buhler et al. in [20] who emphasize that agents in collaborations play a role in the adaptive nature of the collaboration process. Another reason for changes in the process of a DC is adaptation while progressing towards the goal, as shown in the use case. This is due to the fact that some DCs are created on-the-fly [46] and never start with a perfect plan, such as those DCs formed as a response to disasters [68].

All the formalisms used to model DCs up to now, e.g. VDM [16] and RAISE [57], were designed originally for classical processes that are already statically defined and allow for no changes during the runtime. A workflow of a DC should be able to change and adapt to changes while it is running until it reaches the desired goal of the DC. On the other hand, the workflow model of a DC should be able to show clearly the effect of members on the dynamicity of a DC and how they contribute to it [16]. To that end, we choose one formalism as an example of such models, namely Event Structures [81], where rules of workflows are statically defined. Then we show what that formalism lacks to faithfully model DC workflows and cover the missing points. This is done by means of providing extensions, which are meant to be generic and applicable to many similar formalisms, e.g. Petri Nets [77].

Finally, DCs are goal-driven [59]. Accordingly, when their workflows evolve, the evolution itself should be guided by the reachability of their goals. Much work has been done on evolution in processes [60, 61, 80], yet adaptive workflows never address goal-orientation, which has been separately addressed in Goal-Oriented Requirements Engineering [29, 78]. No work has been done on combining both concepts formally and clearly such

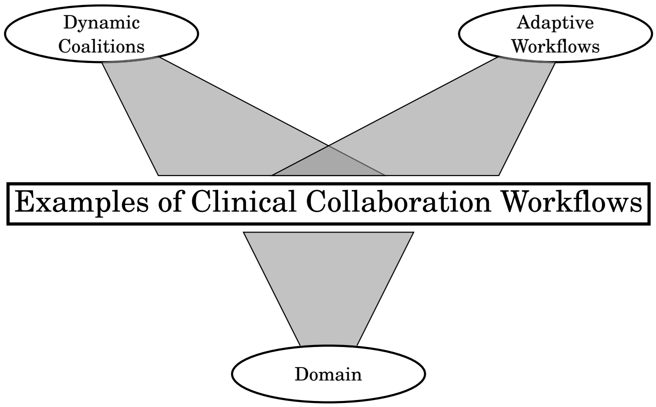


Figure 1.1.: *The three perspectives through which the examples taken from healthcare are studied in this thesis.*

that not only the process of software development is guided by goals, but also the course of evolution is goal-oriented, as we are going to see in our case of DCs.

1.3. Solution Approach

We consider the workflow of a Dynamic Coalition, we give a formal model of it, and highlight its uniqueness. To achieve that:

1) We choose a formalism from the literature, namely *Event Structures (ESs)*, to model workflows in general, and show what it is still missing such that it can be applied in our case to model the evolution or other features of a DC workflow. This will be covered in Chapter 2, which will provide preliminaries, and the reasons behind choosing this formalism, as well as the various kinds of ESs in the literature and their modeling capabilities.

2) Next, we make a study to understand the nature of changes in workflows during their runtime. For that, we choose one ingredient of the formalism we have, namely the causality or flow, and see how it can be adapted by the occurrence of certain events. The result will be a model where changes that might occur are foreseen, and triggered by internal events. So it contributes to only a sub-set of changes that a workflow might face, namely *preplanned changes* of workflows [80], where some other changes in reality as in our use case might be unplanned, i.e. adhoc, and unforeseen at design time [61]. Furthermore, this study is not covering goal-orientation. All such limitations emphasize the need for the next contribution. This study is completely included in Chapter 3.5 along with examples, detailed related work, and an evaluation section with application to the use case.

3) Then, we study the idea of evolution of workflows, which allows for adding new events, dropping old ones as well as changing their flow, in an unplanned way, as a generalization of the last study, that overcomes its limitations. We exhibit the criteria that should be fulfilled through evolution. Furthermore we discuss the idea of goal-orientation, and the various kinds of goals and how they are refined. To the end, we bind evolution with goal-orientation, by showing how evolution should be guided to satisfy the goals set at the beginning, in a way that fits our case of DCs. We use the outcome of the last study of pre-planned changes to infer changes and allow for process learning [66]. By that we provide a holistic approach for modeling dynamicity and changes in processes, with all their different variations. This study is included in Chapter 4 together with a detailed related-work section, and an evaluation section with a full modeling of the cardiac use case.

4) For the sake of completeness, the domain we are considering, namely the clinical domain, emphasizes the need for quantitative and qualitative features in the underlying formalism used to model medical and clinical processes in general. Such features are independent from the concepts of DCs, evolution or goal-orientation. As an example, we mention timing that is needed to model the case that a surgery should be held in two hours maximally. So we add another perspective to our study, namely the domain-specific one, next to the two perspectives of DCs and Adaptive Workflows we have. We examine the major quantitative and qualitative features needed in the healthcare sector, investigate which of them are already covered in the literature regarding ESs, and cover the missing one, namely priority, by adding it to different kinds of ESs. This is done in

Chapter 5 along with examples for different qualitative and quantitative extensions, taken from the healthcare sector.

By that, we provide three extensions for Event Structures: Dynamic Causality, Evolutionary Sequences, and Priority respectively. The extensions and the contributions are generic in the sense that they can be applied to other formalisms, e.g. Petri Nets [77].

5) Finally, DCs can be nested. For instance, assuming that a stroke unit gets involved into the last example, a closer look at its involvement shows another sub-coalition taking place there, e.g. nurses, doctors, therapists, etc. Some of these members join based on an invitation by the doctor, while others leave until the patient is stabilized and then discharged. The same applies for the involvement of the emergency room in the big picture of the stroke example, where other doctors and personnel involve in their turn. The problem of nested DCs is essential when considering and designing the flow of DCs. For instance, the role of a stroke unit might in a DC set to rescue a patient, might be a DC itself run by a set of nurses and doctors. To address this problem, we refer in Chapter 2 to the concept of Action Refinement [63], and show how it can be applied to model nesting in coalitions.

1.4. Contributions

The main contributions of the thesis can be summarized as follows. Detailed contributions are discussed in the evaluation section of each chapter w.r.t. the chapter's topic, and summarized in Chapter 6.

1) We prove that workflows of Dynamic Coalitions are both evolutionary and goal-oriented. We show that the join of a new member in a DC is a possible trigger for the evolution of the DC workflow.

2) We add Dynamic Causality to Event Structures. We study expressiveness of the new causality and show how Dynamic Causality can model other relations e.g. conflict and disabling.

3) We add the notion of Evolving to Event Structures. We combine the dynamics of a system, represented by its system runs, with the dynamics of the system model, represented by its evolution transitions, in one model. Additionally we introduce different approaches to model evolution, and study their equivalence.

4) We add priority to various kinds of Event Structures. We study the relation between priority and other event relations e.g. causality and conflict.

In addition, we develop a tool for Event Structures. The tool can be used to check correctness in evolution, and verify system runs in other ESs defined in this thesis.

1.5. Scope

DC Membership: This work does not contribute to membership of a Dynamic Coalition, since the latter is concerned with selecting members based on what they share, and has been covered in previous works [16, 57]. Besides, the selection process itself does not contribute to the essence of workflow evolution in Dynamic Coalitions. Rather, we abstract membership by capturing events of join and leave, and show how these events might contribute to evolution.

Evolution Guidance: Out of its scope, this work does not contribute to the way of how to change a running workflow such that a goal is satisfied. Instead, it addresses goal satisfaction in a declarative way based on constraints. Suggesting changes and adaptations is left up to the personnel in the medical domain who have the know-how based on experience. Besides, evolution guidance was not emphasized by the doctors we met through the course of this thesis in the *Charité Hospital in Berlin* neither by the doctros in *Deutsches Herzzentrum Berlin (DHZB)*. Rather, the need for checking the consistency between a new change on one hand and the history and goal of a workflow on the other hand was emphasized.

1.6. Tool Support

In the group *MTV* at *TU Berlin* we have developed a tool that supports Event Structures. The tool helps to define structures of various kinds and derive system runs. It supports all kinds of ESs mentioned in this thesis, namely ESs of Chapter 2 (except for Sections 2.8 and 2.9), in addition to all variants of dynamic causality ESs defined in Chapter 3 and prioritized ESs of Chapter 5. The tool has been developed in Scala using Object-Oriented and Functional Programming.

Many notions of system runs have been implemented including traces, configurations, posets, and families of both configurations and posets (cf. Chapter 2 for definitions). The tool provides the ability to check whether any set (or sequence or poset) of events represents a system run in a given ES. Besides it provides the ability to linearize configuartions and posets to obtain traces. To this end, and since disabled events do not belong to any (reachable) system run, the algorithm of deriving and linearizing system runs follows the causality relation of a given ES to avoid disabled events. This enhances performance and avoids state explosion.

Furthermore, the tool provides visualization capabilities for all kinds of ESs supported except for Event Structures for Resalvable Conflict which have no graphical notation [77]. In addition, the tool provides visualization for system runs of all notions and their families. Besides, the tool also provides a console interface. The interface gives the user the ability to define ESs not only through code, but also in runtime i.e. without the need to recompile the code each time a new ES is defined. Therefore, a minimal set of commands have been defined to construct ESs, draw them, and generate their system runs.

Usability in the thesis: The tool can be used to support all concepts and contributions of this thesis. First, regarding dynamic causality of Chapter 3, the tool supports the ability to verify system runs of Growing, Shrinking and Dynamic Causality ESs. This includes traces, configurations and transitions. Second, regarding Priority of Chapter 5, the tool supports the ability to define prioritized ESs of different kinds, e.g. Prime, Bundle, etc. Finally, regarding Evolution of Chapter 4, the tool supports the ability to verify whether a given ES is considered an evolution of another. This includes checking correctness against the history that precedes the evolution (i.e. history preservation) and the satisfaction of goals.

1.7. Related Work

Many studies have been carried out on the topic of Dynamic Coalitions and all its forms such as Virtual Organizations, Distributed Problem Solving. On the other hand, the topic of workflow evolution has been investigated in Business Process field. Furthermore, goal orientation has been covered in Requirements Engineering. Here we examine the overlap between these studies and our work, by illustrating what each of them covers and highlighting the unique points of our work. Furthermore, each chapter in this thesis includes detailed related work w.r.t. its idea and contribution.

1.7.1. Related Work in Dynamic Coalitions

The following studies have been carried out on different dimensions of DCs; none of them covers workflows of DCs, addresses them, or shows their unique characteristics[1].

Formation and Membership: Zuzek et al. in [87] worked on the formation of virtual organizations through contract negotiation, where statements about resources along with the owning agent, and the actions performed against them were negotiated. A negotiation table with pairs of statements existed, and messages were sent to update the table. In this way the authors showed how virtual organizations can evolve by negotiating what each member would share and which actions to be performed on resources shared by others.

Nami et al. in [57] also described how members are selected in the best way to make up a virtual organization. Others, like Rahwan et al. in [59], worked on finding algorithms for generating structures for coalitions, out of a set of agents, such that certain criteria, like *Worst Case Guarantees*, are maintained. Michalak et al. [52] did the same, but with a distributed algorithm to avoid bottlenecks.

Information Flow and Privacy: Mozolevsky and Fitzgerald [53] worked on the dimension of "Information Transformation" and used digraphs to build a model for the flow of information within a coalition. They defined

[1]Some of the related works in Dynamic Coalitions are discussed in Section 1.7.2 since they contribute to evolution.

the concept of interface, which is a combination of an agent's resource and an access privilege, as a vertex in the digraph. Edges in the digraph were tuples between interfaces, connecting two interfaces with different access rights. They used access rights to capture the information flow, but they did not cover the membership dimension or the phenomenon of coalition formation. The authors provided a graphical modeling of coalition structures depending on digraphs. However, it was very specific to the issue of information flow.

Sarrouh in his thesis [68], as well with Bab in [4], worked on building a modeling framework for privacy-aware DCs, that contains various access control mechanisms. He used Abstract State Machines (ASM) [37] to formalize coalitions with concurrency, and applied four access control mechanisms: IBAC, RBAC, ABAC and TBAC based on the membership dynamicity level. Although his work models the concurrent process of certain DCs, it does not show how workflows of DCs might evolve or be affected by membership dynamicity. Rather it uses a formalism, namely ASM, where rules are defined in a static manner.

User Tools: Bobba et al. in [12] worked on building a tool for Administrating Access Control in DCs. They tried to get closer to the user or designer of coalitions, by allowing access negotiation, review, specification and many other access-control operations.

1.7.2. Related Work in Evolution and Adaptive Workflows

Workflow Management Systems were criticized to lack the flexibility to deal with changes and cope to evolving business environments, as changes are a daily routine which big organizations need to consider as a key for success. Thus evolution was studied in Workflow Systems [65] as a mechanism to enable flexibility and respond to changes in the environment. Many studies have been carried out [80, 61, 65, 71, 60, 35] on changes at the process-type level, as well as the instance level w.r.t. dynamic changes which take place while an instance or a workflow is running.

Correctness criteria [65] where established to ensure that a change leads to no undesired scenarios. Some of the criteria were graph-based [71], such that they ensure some inheritance relation between the original and the new workflow process. Other criteria were trace-based where certain system runs of the old process should be preserved by the new one [80]. Such criteria work on a pair basis, i.e. it is a relation defined between the old and the new process. We argue that the previous works lack the notion of a goal, which a DC is based on.

In our work we use the notion of goal satisfaction as our main correctness criterion for evolution of DC workflows, which can be combined together with the other criteria of the literature [65]. We achieve that through defining sequences of evolutionary steps, that must lead to goal satisfaction. In that sense we also deviate from the previous works in the way that they force their correctness criteria over each step of evolution, while we apply our criteria over sequences of steps such that the criteria must hold for one step in the sequence. This provides more flexibility and is more suitable for our case. In this aspect, we also differ from the literature by combining goal orientation with evolution within one model.

One final contribution to the literature of Adaptive Workflows is that we show how the membership dimension of DCs, represented by multiple members' join and leave, could be a trigger for evolution or change in a running workflow, since each member (agent in this context) might bring its own contribution to the workflow, to solve the problem of or satisfy the goal of the DC.

Replanning in Service-Based Business Processes: *Service-Based Business Processes* are realized through different services that are provided by different members [69]. As modern business processes work in dynamic and changing environements, web services are assumed to provide the dynamicity and flexibility needed for adaptation by such processes [19, 17]. Adaptivity is then in the form of recomposing existing services and consuming new services [23]. Similar to our case of DCs, studies in that field consider processes achieved by multiple members based on collaboration. Besides, they focus on processes with goals such that an adaptation must contribute to goal satisfaction [69]. On the other hand, our work focuses on the influence of dynamic membership on the overall workflow. Besides, studies in *Service-Based Business Processes* focus on the unforeseen part of changes as Chapter 4 of our work, while we cover both kinds of changes, i.e. the foreseen and unforeseen. Furthermore, we investigate the relation between the two kinds of changes by the internalization operation defined in Section 4.4, and allow to move from one to the other. Finally, as they are closer to business domain modeling, other studies in that field, e.g. [23, 9] focus on *Quality of Service (Qus)* as a criterin for adaptation, such that the adaptation avoids deviating from expected QoS values. On the contrary we consider the topic of DC workflows from a concurrent-system perspective in an abstract way such that we consider processes made of actions, namely events, and their relations, even w.r.t. goal modeling (cf. Section 4.6), and we take into consideration equivalences between modeling approaches as in Section 4.3

Evolution in DCs: Khan et al. addressed evolution in DCs [45], by modeling the dynamics of coalition formation in a spectrum-sharing environment, where transmitter-receiver pairs reach stable coalition structures through a time-evolving sequence of steps. They used Markov Chains as a formalism. Although they used evolution as a mechanism to stabilize the coalition, their model was specific for communication, and depending on the environment of the DCs they studied. Similar work has been done by Ye et al. in [86] to provide a formation mechanism for self-adapted dynamic coalitions of sensors. The mechanism enables agents to adjust their degree of involvement in different DCs. Contrary to their approach, we investigate evolution w.r.t. the workflow of the DC, which we study as a special case of concurrent systems.

Dynamic Condition Response Graphs (DCR Graphs): Mukkamala in [54] introduced DCR Graphs as a constraint-based modeling language for workflows. Similar to ESs, DCR Graphs are concerned with defining relations between events. But unlike ESs, DCR Graphs provide the ability to repeat events. Besides, next to causality—which is already available in ESs—DCR Graphs offer other relations, e.g. the *exclude* relation which means skipping a certain event or activity. Furthermore DCR Graphs provide the notion of *restless* events which denote events that must be executed once they are enabled. Compared to ESs, such relations offer a means for modeling that is more suitable for workflows. On the other hand, ESs provide more basic relations which allow for studying dynamicity on a fine level as shown in Section 3 and implementing the relations of DCR Graphs as discussed later. However, the work of Mukkamala and the extension made later to DCR Graphs [55] in order to support ad-hoc changes, make it similar to our work in the sense of modeling Adaptive Workflows and covering both pre-planned as well as ad-hoc changes.

The overlap in covering pre-planned changes is represented by the fact that many concepts and relations of DCR Graphs can be implemented through the concept of Dynamic Causality we present (cf. Chapter 3), where Dynamic Causality can be seen as more fundamental and works on a finer semantical level. For instance, the exclude relation means skipping an activity or event, and can be implemented by Dynamic Causality through disabling that event and dropping it as a causal predecessor from its successors, as illustrated in Section 3.6. The *include* relation of DCR Graphs can be mapped directly to the notion of Growing Causality in our approach. The concept of *restless* events is covered in ESs through the notion of urgent events [44] as illustrated in Sections 2.10 and 5.1. The repetition of events can be implemented through unfolding and event duplication, as illustrated in Section 2.10.3. The only concept, from Dynamic Causality, that is missing in DCR Graphs is Shrinking Causality which provides the ability to drop causality pairs between events without the obligation to skip any of them as in the *exclude* relation.

Regarding ad-hoc changes and adaptation, DCR Graphs were applied in [55] by Mukkamala et al. as a formal modeling language for Adaptive Case Management, where new *Adaptation Operations* were defined in order to allow for ad-hoc changes in a workflow definition. Again the overlap between our evolution concept and this related work is big in the sense that both support ad-hoc changes that need not to be foreseen. One major difference is that, similar to [80], [55] addresses adaptivity from the perspective of small changes that lead to workflow evolution, e.g. *add/remove an event*, *add/remove a constraint between two events*, etc. In our work, we address adaptivity from an evolution perspective by capturing transitions between different ESs or workflow definitions, where such a transition might hold complex changes. Accordingly, our approach highlights the need for a mechanism to extract changes, as provided in Section 4.4. Verification in [55] is done on the new graph after adaptation, by transforming it to *Büchi Automata* and performing model checking for certain properties, e.g. dead-lock freeness and live-lock freeness. For our case, we provide correctness criteria for evolution based on the history of a workflow and its acceptance as a system run in the new workflow definition as illustrated in Section 4.2, in addition to criteria on goal reachability. One common aspect between the two verification approaches is that maintaining correctness criteria in both approaches can be tolerated in some intermediate evolutions. This provides flexibility in modeling and reflects reality in the healthcare domain that inspires both works.

1.7.3. Related Work in Goal Orientation

Goals capture the objectives a system should satisfy, and form the foundations to detect conflicts in requirements; yet they are absent from many formal models [30]. Goal-Oriented Requirements Engineering and Elaboration [78] are concerned with defining goals, and their use in structuring and modifying requirements. This includes refining goals into sub-goals, defining their relations to each other and the way to satisfy them. Goals have been divided into types and categories, e.g. *functional* and *non-functional* [78]. Functional goals were defined to be concerned with services the system is expected to deliver, while non-functional[2] are concerned with how the system delivers services. Others classified goals as *soft* and *hard* goals [28], such that satisfaction of soft goals cannot be established in a clear-cut way [78], while satisfaction of hard goals can be established in a clear-cut sense and proved by verification techniques [29].

As in the research field of Artificial Intelligence, relations between sub-goals were defined (e.g. *AND, OR* links) [28]. Criteria for measuring the achievement of sub-goals—of soft goals—were defined depending on how much they contribute to the main goal [78]. Furthermore, Darimont and van Lamsweerde worked on formally refining goals as in [30] according to patterns. The KAOS methodology [28] was developed by van Lamsweerde for goal modeling in requirements engineering, and defining relations between different goals, agents, objects and actions. Goals are expressed there in terms of formulas to be asserted, using Linear Temporal Logic (LTL), and a real-time variant of it.

We do not contribute in this work to goal-orientation concepts. Rather, we show how to apply goal-orientation in our case of collaboration-based workflows that are goal-driven, i.e. Dynamic Coalitions, and w.r.t. Event Structures. Since we use an event-based model, namely ESs, we model goals through events and show how to satisfy a given goal over the course of evolution of a workflow. We contribute to the literature of goal-orientation

[2]No common definition of non-functional requirements exists. Glinz [38] addresses this problem by showing the different definitions available and the problems accompanied with each, and then contributes concepts for solving these problems.

such that we show how to use goals to guide the evolution of a given workflow, in a way that combines both: Adaptive Workflows and goal orientation. One final note is that refinement of goals is provided to us, in the many examples we use, by the medical doctors themselves, based on their medical knowledge.

1.8. Publications

Dynamic Causality in Event Structures: Except for Sections 3.4.3 and 3.4.4 and the notion of *remainder*, Chapter 3 was submitted (and accepted) in [2] by the author of this thesis as a first author, together with D. Karcher, K. Peters and U. Nestmann respectively. The general idea of dynamic causality, the idea of separating *Growing* from *Shrinking* causality, and the interpretation of each (growing causality corresponds to conditional causality, and shrinking to disjunctive causality), then merging them to study each separately, were the first author's ideas. This included defining the new structures in addition to their traces, configurations and transitions regarding each kind of causality, and suggesting the best notion for the semantics.

D. Karcher and K. Peters helped in comparing the resulting new kinds of ESs with other kinds of ESs from the literature. This implied forming the lemmas and the proofs. K. Peter was responsible mainly for the comparison between Shrinking Causality in Section 3.3 and Dual ESs, including the various kinds of posets in App. B, as well for finding counterexamples for the comparison with other ESs. D. Karcher was responsible for Section 3.4 of Growing Causality including the comparison with ESs for Resolvable Conflict [76], and contributed to the complex definition of DCES's transition relation in Section 3.5. On the contrary, the comparison and embedding of Extended Bundle ESs [47] into DCESs were again the first author's responsibility, including the definitions needed, lemmas and proofs.

U. Nestmann was responsible for motivating the work in the introduction part, as well as for the section of contributions. Based on his experience, he played a role in deciding how broad the comparison should be with other ESs, and in placing the work among the related works. Finally, all authors contributed to reviewing other authors' work, including definitions, texts, and specially proofs, to avoid mistakes.

The sections of Conditional Causality, Concurrency between Target and Modifier in Growing Causality of this thesis were not part of the paper; they were developed mainly by the first author, with the help of D. Karcher. The notion of remainder was later defined for all kinds of dynamic causality ESs by the first author. Motivating the idea of dynamic causality by Dynamic Coalitions and the relation to pre-planned deviations in *Workflows* [80] were also done by the first author, together with adapting the paper into the whole thesis which included applying it to the use cases, and the connection to the evolution chapter.

Priority in Event Structures: Except for its application in DC workflows, Chapter 5 was completely published in [3] by the author of this thesis as a first author, together with K. Peters and U. Nestmann. The whole idea of prioritized ESs was the first author's idea, including defining the new structures, studying priority redundancy and investigating the relation between priority from one side, and causality and conflict from the other side.

In general, the other authors helped in producing a paper out of those ideas, since it was the first-author's first paper to be published. K. Peters helped in forming lemmas and proofs which were limited in this paper. The idea of splitting Prioritized Extended Bundle ESs to Prioritized Bundle ESs and Prioritized Extended Bundle ESs was Peters' idea as well, in order to isolate causality from disabling, and their relation with priority separately. U. Nestmann helped in the orientation part, by writing the introduction, the conclusion, as well as the related work and motivating the whole work in general. The first author was later responsible for adapting the paper into the thesis and applying the work on special use cases obtained from healthcare.

2. Technical Preliminaries: Event Structures as a Formalism for Dynamic-Coalition Workflows

2.1. Introduction

Event Structures (ESs) usually address statically defined relationships, typically represented as *causality* (for precedence) and *conflict* (for choice). These relationships constrain the possible occurrences of events. An event is a single occurrence of an action; it cannot be repeated. ESs were first used to give semantics to Petri nets [81], then to process calculi [15, 47], and recently to model quantum strategies and games [85]. The semantics of an ES itself is usually provided either by the sets of traces compatible with the constraints, or by means of configuration-based sets of events, possibly in their partially-ordered variant (*posets*).

In *interleaving* models, only one event can take place at any given instant of time. There, "concurrency" arises in terms of free choice or non-determinism, i.e. concurrent events may appear in either order. In contrast, the model underlying ESs is *non-interleaving*. Here, concurrency is expressed more explicitly via the dependency between events: events are concurrent if they are neither in conflict nor causally related. This intuition also manifests in system runs, represented by so-called configurations, or in terms of partial orders where concurrent events are simply unordered.

All kinds of ESs defined up to now comprise a set of events, and a causality or enabling relation, which is a basic principle, between events, like in [81, 83, 43]. Only when an event is enabled, it may happen, i.e. it cannot happen without its causal predecessors. Additionally, ESs might be equipped with a labelling function, associating each event with the action it represents. But for simplicity we omit the labelling here since our results are not dependent on it.

In philosophy, causes can be sufficient or necessary [36]. If x is a necessary cause of y, then y cannot happen without x, but if x happens that does not mean y will happen. This optionality is what is modelled by ESs, and thus it is called "enabling" in some structures. While sufficient cause is defined as: if x happens then this implies that y will happen. This might be mapped to urgent events in [44]. Besides, the presence of y does not imply the presence of x as y might be caused alternatively by z, and that can be seen in some structures, as we will see, like disjunctive-causality ESs.

On the other hand, ESs define a conflict relation between events representing choice. It is usually symmetric as in Prime ES [83], or can be asymmetric like in Asymmetric ES [8]. In fact causality and conflict are two sides of the same coin. Causality represents a positive dependency: for event e to happen, it needs event e' to have happened. On the other hand conflict can be seen as a negative dependency: for e to happen, e' should not happen. Accordingly, events not under causality or conflict relation are considered independent events, and hence they can occur concurrently.

In addition, all event structures define the concept of configurations representing system runs. A configuration is a set of events which happened or might happen. Since basic information units in *temporal* systems are events [84] and more events that happened denote more information, a configuration denotes the state of a system. Configurations could be ordered according to the causality relation—and possibly to other relations (like asymmetric conflict [47])—in the structure to give information not only about which events occur, but also in which order. The order could be partial, such that incomparable events are independent and free to occur in any order. A system run could also be shown as a sequence of events, i.e. a total order, which is considered a special case of posets. Additionally, many event structures [47] define the concept of the *remainder*. The remainder of an ES represents what is left from the process to execute after a specific system run, or a configuration, has taken place.

Semantics of ESs can be defined in terms of families of configurations as in [83], or configuration structures as in [75] holding a labelling function additionally. Alternatively, it can be defined in terms of families of posets like in [47] which are more expressive than families of configurations.

Event Structures have a graphical representation, which helps while designing systems. Events are represented as dots. Causality is represented as directed arrows from causes to enabled events. Undirected lines or dashed lines are used to represent conflict, or dashed arrows could be used in case of asymmetric conflict.

2.2. Prime Event Structures

Prime Event Structures (PESs), invented by Winskel [81], are the simplest and first version of ESs. Causality is expressed in terms of an enabling relation, as a partial order between events. For an event to become enabled in PESs, all of its predecessors with respect to the enabling relation must take place. There is also a conflict relation between events to provide choices, given as a binary symmetric relation. Many versions of Prime ESs appear in the literature [83, 81, 82, 58]; here we rely on the one from [83].

Definition 2.2.1. *A* Prime Event Structure (PES) *is a triple* $\pi = (E, \#, \leq)$ *where:*

- *E, a set of* events
- *$\# \subseteq E \times E$, an irreflexive symmetric relation (the* conflict *relation)*
- *$\leq \subseteq E \times E$, a partial order (the* enabling *relation)*

that additionally satisfies the conflict heredity *and* finite causes *constraints, respectively as follows:*

$$\begin{array}{ll} 1. & \forall e, e', e'' \in E \,.\, e \# e' \wedge e' \leq e'' \implies e \# e'' \\ 2. & \forall e \in E \,.\, \{e' \in E \mid e' \leq e\} \textit{ is finite} \end{array}$$

Figure 2.1 shows an example of a PES, where the transitive and reflexive closure of the enabling relation are not shown for simplicity, a dashed line means conflict, and an arrow means causality directed towards the caused event.

In Prime ESs, a configuration is a conflict-free set of events $C \subseteq E$ that is left-closed under the enabling relation, i.e. no two events of C are in conflict and for all predecessors e with respect to $\leq$ of an event $e' \in C$ it holds $e \in C$. Thus, given a Prime ES $\pi = (E, \#, \leq)$, a configuration C represents a *system run* of π (or the *state* of π after this run), where events not related by $\leq$ occur concurrently. We denote the set of configurations of a PES π as $\mathrm{C}(\pi)$.

A *trace* is a sequential version of a system run. It can be defined as a sequence of events that is conflict-free and all the predecessors of an

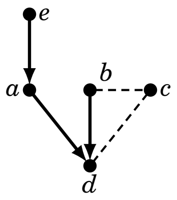


Figure 2.1: *An example of a Prime Event Structure showing conflict heredity.*

event in the trace, w.r.t. $\leq$, precede that event in the trace. As for the formal definition, we choose a different yet equivalent definition of a trace, adapted from [43], on which we will rely when defining priority later on. Let σ be a sequence of events $e_1,\ldots,e_n$ such that $\{e_1,\ldots,e_n\} \subseteq E$ in a PES $\pi = (E,\#,\leq)$. We refer to $\{e_1,\ldots,e_n\}$ by $\bar{\sigma}$, and we call $\mathrm{en}_\pi(\sigma)$ the set of events that are enabled by σ, where:

$$\mathrm{en}_\pi(\sigma) := \big\{e \in (E \setminus \bar{\sigma}) \mid \big(\nexists e' \in \bar{\sigma} \,.\, e\#e'\big) \wedge \big(\forall e' \in E \,.\, e' \leq e \wedge e' \neq e \implies e' \in \bar{\sigma}\big)\big\} \quad (2.1)$$

We use σ_i to denote the prefix $e_1,\ldots,e_i$, for some $i < n$. Then, the sequence $\sigma = e_1,\ldots,e_n$ is called a *trace of* π iff:

$$\forall i \leq n \,.\, e_i \in \mathrm{en}_\pi(\sigma_{i-1}) \quad (2.2)$$

Accordingly, a trace is a linearization of a configuration respecting $\leq$. Usually multiple traces can be derived from one configuration. The differences between such traces of the same configuration result from concurrent events that are independent, i.e. are related neither by enabling nor conflict. For example, in Figure 2.1, the events c and a are independent and thus concurrent in a configuration like $\{e,a,c\}$. From $\{e,a,c\}$ the traces eac, eca, and cea can be derived for the structure in Figure 2.1.

Expressiveness of some kinds of event structures,e.g. PESs, as well as semantics can be shown in terms of families of configurations [62].

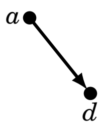
Definition 2.2.2. *A family of configurations is a set $\mathbb{C}$ of configurations satisfying:*

- $\emptyset \in \mathbb{C}$
- $\forall F,G,H \in \mathbb{C} \,.\, F \cup G \subseteq H \implies F \cup G \in \mathbb{C}$
- $\forall F \in \mathbb{C} \,.\, \forall a,b \in F \,.\, a \neq b \implies \exists G \in \mathbb{C} \,.\, G \subseteq F \wedge (a \in G \iff b \notin G)$

The order relation in families of configurations is the subset containment relation $\subseteq$ itself. The figure below shows the largest family of configurations of Figure 2.1, where reflexive and transitive arrows are not shown for simplicity. Families of configurations form themselves a model for concurrency which show how the state of a system—represented by configurations—can evolve.

$$\begin{tikzcd}
 & \{b\} \arrow[r] & \{b,e\} \arrow[r] & \{b,e,a\} \arrow[r] & \{b,e,a,d\} \\
\emptyset \arrow[ur] \arrow[r] \arrow[dr] & \{e\} \arrow[ur] \arrow[r] \arrow[dr] & \{e,a\} \arrow[ur] \arrow[r] & \{e,a,c\} & \\
 & \{c\} \arrow[r] & \{c,e\} \arrow[ur] & &
\end{tikzcd}$$

As mentioned before, the notion of a remainder of an ES denotes the future of a system after a given system run, called a *history* $H \in \mathrm{C}(\pi)$. Events of H are dropped from the remainder. As well, events that conflict with any event of H cannot occur anymore, and thus are dropped from the remainder. Causal predecessors of an event will not be predecessors of that event in the remainder in case they took place in H. Consequently, events whose all predecessors took place in H will be initially enabled in the remainder. For instance, the structure aside is the remainder of the PES in Figure 2.1 after $H = \{e,b\}$.



Definition 2.2.3. *Let $\pi = (E, \#, \leq)$ be a PES, and let $H \in \mathrm{C}(\pi)$ be a configuration of π. The remainder of π after H is $\pi[H] = \big(E', \#', \leq'\big)$, where:*

- $E' = E \setminus \big(H \cup \{e \mid \exists e' \in H \,.\, e\#e'\}\big)$
- $\#' = \# \cap E'^2$
- $\leq' = \leq \cap E'^2$

Since # is irreflexive and symmetric, its projection on E'^2, i.e. $\#'$, is irreflexive and symmetric too. Analogously, it can be seen that $\leq'$ is a partial order, too. The properties of conflict heredity and finite causes hold also on the subsets E' and $\leq'$. Thus $\pi[H]$ is a PES.

Consistency between a structure and its remainder could be seen from the issue that resuming the execution from H in the original structure is the same as starting execution from scratch in the remainder. This is illustrated by the following lemma, where the proof is left to Appendix A.1.

Lemma 2.2.4. *Let π be a PES, let $H \in \mathrm{C}(\pi)$. Then:*

$$\forall C \subseteq E \setminus H \,.\, \big(C \in \mathrm{C}(\pi[H]) \iff H \cup C \in \mathrm{C}(\pi)\big).$$

2.3. Stable Event Structures

To overcome the limitation of unique enabling of PESs, Winskel's Stable ES [83, 82] presents enabling as a relation between sets of events on one side, and a single event to be enabled on the other side. The same event can be in a relation with more than one set, where the occurrence of events in just one set is sufficient to enable that event, but all the events in that set must occur then. Such enabling has a Disjunctive Normal Form.

Stable event structures add an additional and essential constraint on the enabling sets, saying that if an event is enabled by two sets, it must be enabled by their intersection, given that the union of the two sets is conflict-free. This is essential for causality determinism, so that in case one event occurred, it is precisely known which events had caused it, and thus those structures are called "stable". The example in Figure 2.2 respects this constraint, as e_3 conflicts with e_5. The following gives a formal definition of Stable ESs based on [83] with a slight modification.

Definition 2.3.1. *A* Stable Event Structure (Stable ES) *is a triple* $\kappa = (E, \#, \vdash)$ *where:*

- *E, a set of* events
- *$\# \subseteq E \times E$, an irreflexive symmetric relation (the* conflict *relation)*
- *$\vdash \subseteq \mathcal{P}_{fin}(E) \times E$, the* enabling *relation*

that additionally satisfies for all $F, G \subseteq E$ and $e \in E$:

- ***Consistency:*** *$F \vdash e \Rightarrow F$ is conflict free, i.e. $\forall e', e'' \in F \,.\, \neg(e' \# e'')$*
- ***Stability:*** *$(F \vdash e \wedge G \vdash e \wedge F \cup G \cup \{e\}$ is conflict free$) \implies F \cap G \vdash e$*

A configuration C in a Stable ES $\kappa = (E, \#, \vdash)$ is a set of events $C \subseteq E$ that is conflict-free, i.e. $\forall e', e'' \in C \,.\, \neg(e' \# e'')$, and secured, i.e. $\forall e \in C \,.\, \exists e_1, \ldots, e_n \in C \,.\, (e_n = e \wedge \forall i \,.\, \exists X \subseteq \{e_1, \ldots, e_{i-1}\} \,.\, X \vdash e_i)$.

For instance, $\{e_1, e_2, e_3, e_4\}$ is a configuration in Figure 2.2, while $\{e_1, e_2, e_4\}$, $\{e_1, e_2, e_6, e_4\}$ are not. The *stability* property together with the *consistency* prohibit $\{e_1, e_2, e_3, e_4, e_5, e_6\}$ from being a configuration since it is not conflict-free. On the other hand, $\{e_5, e_6, e_4\}$ is a configuration where the

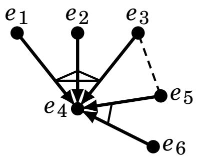


Figure 2.2: *An example of a Stable Event Structure satisfying consistency and stability, where the bars between causality arrows denote one enabling set.*

fact that e_4 can be enabled in different ways provides flexibility in causality; such flexibility was missing in PESs. Thus it is impossible to model the structure in Figure 2.2 by a PES with the same configurations. To do this, events like e_4 need to be duplicated, such that one copy depends on $\{e_1, e_2, e_3\}$ and the other depends on $\{e_5, e_6\}$, given that the two copies are in conflict.

According to Boudol et al. [14], in terms of families of configurations, Stable ESs are strictly more expressive than Prime ESs, i.e. each PES can be modelled through a Stable ES with the same configurations, while the other way around does not hold.

2.4. Bundle Event Structures

In [47], Langerak presented Bundle Event Structures, where the conflict relation is as in Prime ESs an irreflexive and symmetric relation, but the enabling relation offers some optionality, based on bundles. A *bundle* (X, e), also denoted by $X \mapsto e$, consists of a bundle set X and the event e it enables. A *bundle set* is a set of events that are pairwise in conflict. There can be several bundles $(X_1, e), \ldots, (X_n, e)$ for the same event e. So, instead of a set of events as in Prime ESs, an event e in Bundle ESs is enabled by a set $\{X_1, \ldots, X_n\}$ of bundle sets.

When one event of a set X_i takes place, then the bundle $X_i \mapsto e$ is said to be satisfied; and for e to be enabled all its bundles must be satisfied. In Bundle ESs (and also Extended Bundle ESs) no more than one event out of each set X_i can take place; this leads to causal unambiguity [48]. But since a bundle set can be satisfied by any of its members, this yields disjunctive causality and gives flexibility in enabling.

Definition 2.4.1. *A* Bundle Event Structure (BES) *is a triple* $\beta = (E, \#, \mapsto)$ *where:*

- *E, a set of* events

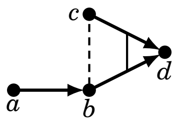


Figure 2.3: *An example of a Bundle Event Structure*

- *$\# \subseteq E \times E$, an irreflexive symmetric relation (the* conflict *relation)*
- *$\mapsto \subseteq \mathcal{P}(E) \times E$, the* enabling *relation*

that additionally satisfies the stability *constraint:*

$$\forall X \subseteq E \, . \, \forall e \in E \, . \, X \mapsto e \implies (\forall e_1, e_2 \in X \, . \, e_1 \neq e_2 \implies e_1 \# e_2) \quad (2.3)$$

Figure 2.3 shows an example of a BES. The solid arrows denote causality, i.e. reflect the enabling relation, where the bar between the arrows shows that they belong to the same bundle and the dashed line denotes again a symmetric conflict. Thus there are two bundles in this example, namely the singleton $\{a\} \mapsto b$ and $\{b,c\} \mapsto d$. As required by (2.3) we have $b\#c$ and $c\#b$.

A configuration of a BES is a conflict-free set of events where all bundles of each event in the configuration are satisfied. Therefore the stability condition avoids causal ambiguity [48]. To exclude sets of events that result from enabling cycles we use traces[1].

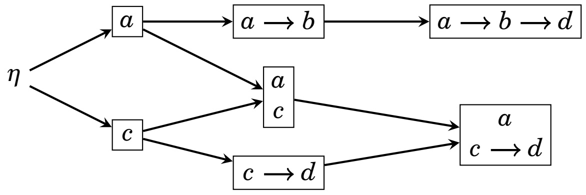
Definition 2.4.2. *Let $\beta = (E, \#, \mapsto)$ be a BES, and let $\sigma = e_1, \ldots, e_n$ be a sequence of events and $\bar{\sigma} = \{e_1, \ldots, e_n\}$ such that $\bar{\sigma} \subseteq E$. We use $\mathrm{en}_\beta(\sigma)$ to refer to the set of events enabled by σ:*

$$\mathrm{en}_\beta(\sigma) := \big\{ e \in (E \setminus \bar{\sigma}) \mid \big(\nexists e' \in \bar{\sigma} \, . \, e \# e'\big) \wedge (\forall X \subseteq E \, . \, X \mapsto e \implies X \cap \bar{\sigma} \neq \emptyset) \big\} \quad (2.4)$$

Then the sequence $\sigma = e_1, \ldots, e_n$ is called a trace *of β iff $\forall i \leq n \, . \, e_i \in \mathrm{en}_\beta(\sigma_{i-1})$*

A set of events $C \subseteq E$ is a *configuration* of β if there is a trace t such that $C = \bar{t}$. This trace based definition of a configuration will be the same for Extended Bundle and Dual ESs. Let $\mathrm{T}(\beta)$ denote the set of traces and $\mathrm{C}(\beta)$ the set of configurations of β. To obtain the posets of a BES, we endow each of its configurations with a partial order.

[1]Here we adapt Katoen's definition [43] which is equivalent to the original definition of Langerak in [47].



η
a
$a \longrightarrow b$
$a \longrightarrow b \longrightarrow d$
a c
c
$c \longrightarrow d$
a $c \longrightarrow d$

Figure 2.4.: *A family of posets of the structure in Figure 2.3. The arrows between sets denote the prefix relation. The transitive and reflexive closures are neither shown in the posets nor in the family.*

Definition 2.4.3. *Let $\beta = (E, \#, \mapsto)$ be a BES and $C \in \mathrm{C}(\beta)$, and $e, e' \in C$. Then $e \prec_C e'$ if $\exists X \subseteq E \,.\, e \in X \land X \mapsto e'$. Let $\leq_C$ be the reflexive and transitive closure of $\prec_C$.*

It is proved in [47] that $\leq_C$ is a partial order over C. We denote the set of posets of β by $\mathrm{P}(\beta)$. It is proved as well in [47] that given two Bundle ESs β, β', it holds:

$$\mathrm{P}(\beta) = \mathrm{P}(\beta') \iff \mathrm{C}(\beta) = \mathrm{C}(\beta') \tag{2.5}$$

Posets can be connected through a prefix relation to form up a family of posets. The prefix relation is defined [62] as:

$$\langle A, \leq \rangle \text{ is a prefix of } \langle A', \leq' \rangle \iff A \subseteq A' \land \leq = \big(\leq' \cap (A' \times A) \big) \tag{2.6}$$

Definition 2.4.4. *A family of posets $\mathscr{P}$ is then a non-empty set of posets downward closed under the prefix relation.*

According to Rensink [62], families of posets form a convenient underlying model for models of concurrency, and are strictly more expressive than families of configurations. Although expressiveness of BESs can still be shown through families of configurations as in [14], families of posets were used in [47] to compare BESs to their extended version, namely *Extended Bundle ESs*.

Figure 2.4 shows the largest family of posets for the example in Figure 2.3. We use η to denote the empty poset $\langle \emptyset, \emptyset \rangle$. A non-empty poset $\langle A, \leq \rangle$ is visualized by a box containing all the events of A and where two events e_1 and e_2 are related by an arrow iff $e_1 \leq e_2$. Reflexive and transitive arrows are usually omitted.

The stability condition in Bundle ESs is stricter than the one in Stable ESs, requiring that a conflict should exist between each two alternatives, which is not the case in Stable ESs. For instance, it is impossible to have a Bundle ES with the same configurations as Figure 2.2, since e_6 is not in conflict with any event of the alternative enabling set of e_4. Boudol et al. in [14] proved that, in terms of families of configurations, Stable ESs are strictly more expressive than Bundle ESs. The translation from Bundle ESs to Stable ESs is done through converting causality from the Disjunctive Normal Form to the Conjunctive Normal Form, with maintaining the same conflict relation. On the other hand, and similar to the disjunction offered in Stable ESs, Prime ESs cannot model the disjunctive causality of Bundle ESs, while Bundle ESs can model Prime ESs using singleton-based bundles. Thus Bundle ESs are strictly more expressive than Prime ESs, as proved in [14]. Figure 2.9 shows the result of these expressiveness comparisons.

2.5. Extended Bundle Event Structure

Langerak in [47] extended Bundle Event Structures with a new asymmetric conflict relation $\rightsquigarrow$ between events, replacing the classical symmetric conflict relation. $e' \rightsquigarrow e$ means that if e occurs, e' can not occur afterwards, thus we can call it an exclude relation. In other words, e' can occur only before e occurs. The classical symmetric conflict between two events can be derived then from this asymmetric conflict relation, when both events exclude each other.

Definition 2.5.1. *An* Extended Bundle Event Structure (EBES) *is a triple* $\varepsilon = (E, \rightsquigarrow, \mapsto)$ *where:*

- *E, a set of* events
- $\rightsquigarrow \subseteq E \times E$*, an irreflexive asymmetric relation (the* disabling *relation)*
- $\mapsto \subseteq \mathcal{P}(E) \times E$*, the* enabling *relation*

that additionally satisfies the stability *constraint:*

$$\forall X \subseteq E. \forall e \in E. X \mapsto e \implies \forall e_1, e_2 \in X \,.\, (e_1 \neq e_2 \implies e_1 \rightsquigarrow e_2) \quad (2.7)$$

Figure 2.5 shows an example of an EBES, where the dashed arrow means an asymmetric conflict, and the rest are like in BESs. It shows two bundles for e_1, both respecting the last constraint; one of them is a singleton $\{e_4\}$.

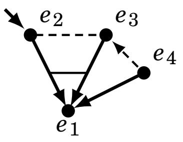


Figure 2.5: *An example of an Extended Bundle Event Structure*

Event e_2 has a empty bundle, which will be according to the definition in next section an impossible event.

Definition 2.5.2. *Let $\varepsilon = (E, \rightsquigarrow, \mapsto)$ be an EBES, and let $\sigma = e_1, \ldots, e_n$ be a sequence of events and $\bar{\sigma} = \{e_1, \ldots, e_n\}$ such that $\bar{\sigma} \subseteq E$. We use $\mathrm{en}_\varepsilon(\sigma)$ to refer to the set of events enabled by σ:*

$$\mathrm{en}_\varepsilon(\sigma) := \Big\{e \in (E \setminus \bar{\sigma}) \mid \big(\nexists e' \in \bar{\sigma}\,.\, e \rightsquigarrow e'\big) \wedge \\ (\forall X \subseteq E\,.\, X \mapsto e \implies X \cap \bar{\sigma} \neq \emptyset)\Big\} \tag{2.8}$$

Then the sequence $\sigma = e_1, \ldots, e_n$ is called a trace *of ε iff $\forall i \leq n\,.\, e_i \in \mathrm{en}_\varepsilon(\sigma_{i-1})$*

For example, in Figure 2.5, the sequence e_3, e_1 is not a trace as the singleton bundle $\{e_4\} \mapsto e_1$ was not satisfied. On the other hand, e_3, e_4, e_1 is not a trace also as e_3 excludes e_4. Besides, the sequence e_2 is not a trace because $\sigma_0 \cap X = \emptyset$ where X is the empty bundle of e_2, so e_2 is an impossible event.

A set of events $C \subseteq E$ is a configuration iff $\exists \sigma$ an event trace: $C = \bar{\sigma}$. Here we can notice that the relation between configurations and traces is one-to-many. We also define $C(\varepsilon)$ as the set of all configurations of ε.

Definition 2.5.3. *Let $\varepsilon = (E, \rightsquigarrow, \mapsto)$ be a BES and $C \in \mathrm{C}(\varepsilon)$, and $e, e' \in C$. Then $e \prec_C e'$ if $\exists X \subseteq E\,.\,\big(e \in X \wedge X \mapsto e'\big) \vee e \rightsquigarrow e'$. Let $\preceq_C$ be the reflexive and transitive closure of $\prec_C$.*

It is proved in [47] that $\preceq_C$ is a partial order over C. It is also proved in [47, 43] that given two EBESs $\varepsilon, \varepsilon'$ it holds that:

$$\mathrm{P}(\varepsilon) = \mathrm{P}\big(\varepsilon'\big) \iff \mathrm{T}(\varepsilon) = \mathrm{T}\big(\varepsilon'\big) \tag{2.9}$$

The asymmetric conflict of EBESs makes it insufficient to use families of configurations for expressiveness. The order relation $\subseteq$ in families of configurations is not able to show that a system cannot proceed from $\{a\}$ to $\{a, c\}$ when $c \rightsquigarrow a$.

On the other hand, the asymmetric conflict relation makes Extended Bundle ESs a generalization of Bundle ESs, and even strictly more expressive than Bundle ESs [47]. Furthermore, Extended Bundle ESs cannot include, i.e. model all, Stable ESs due to the same reason as why Bundle ESs cannot include Stable ESs. Additionally, Stable ESs cannot model the asymmetric conflict of Extended Bundle ES, then Extended Bundle ESs are incomparable to Stable ESs [14], as shown in Figure 2.9.

2.6. Dual Event Structure

The stability constraint in BESs and EBESs prohibits two events from the same bundle to take place in the same system run. Thus e.g. $\{a,b,c,d\}$ is not a configuration in Figure 2.3. It provides some kind of stability to the causality in the structure. More precisely due to stability in every trace or poset, for each event the necessary causes can be determined. Without the conflict between b and c, the trace a,b,c,d would be possible, but then it would be impossible to determine whether d was enabled by b or c. This is the so-called *causal ambiguity* [48].

The definition of Dual ESs varies between [43] and [48], but both exhibit causal ambiguity. In [43] Dual ESs are based on Extended Bundle ESs, while in [48] they are based on Bundle ESs, and thus simpler. In this thesis we are interested in the simplest version exhibiting causal ambiguity, thus we consider the one of [48]. A *Dual Event Structure (DES)* is a triple $\delta = (E,\#,\mapsto)$ similar to Definition 2.4.1 but without the stability condition.

Definition 2.6.1. *A* Dual Event Structure (DES) *is a triple $\delta = (E,\#,\mapsto)$, where E is a set of* events, *$\# \subseteq E^2$ is an irreflexive symmetric relation (the* conflict *relation), and $\mapsto \subseteq \mathcal{P}(E) \times E$ is the* enabling *relation.*

The definitions of $\mathrm{en}_\delta(\sigma)$, traces, and $\mathrm{T}(\delta)$ are similar to Section 2.4 for BESs. Figure 2.6 shows a DES taken from [48], where the sequence a,b,c is a trace, and would not be so in the presence of the stability constraint.

Causal ambiguity affects the way posets are built, since the causal order is not clear. In [48] Langerak et al. tried to solve this problem. They illustrated that there are different causality interpretations possible for causal ambiguity. They defined five different intentional posets: liberal, bundle-satisfaction, minimal, early and late posets. Intentional means posets are

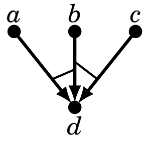


Figure 2.6: *An example of a Dual Event Structure.*

defined depending on the causality relations in the structure, while observational on the other hand means posets are obtained out of event traces when no structure of the system is available, but only behavior. We examine these different kinds and their relations informally here for brevity (cf. Appendix B for formal definitions).

The authors illustrated that in order to detect the cause of an event like d in the trace[2] $abcd$ of Figure 2.6, one should consider the prefix of d (i.e. abc) and then can have the following interpretations:

- Liberal Causality: means that any cause out of the prefix is accepted as long as all bundles are satisfied: abc, ab, b, ac, bc, etc.are all accepted as a cause for d. Then all events in a cause precede d in the built poset, e.g. the posets for the last causes are
$$\begin{tikzcd} a \arrow[dr] \\ b \arrow[r] & d \\ c \arrow[ur] \end{tikzcd}, \begin{tikzcd} a \arrow[dr] \\ b \arrow[r] & d \\ c \end{tikzcd}, \begin{tikzcd} a \\ b \arrow[r] & d \\ c \end{tikzcd}, \begin{tikzcd} a \arrow[dr] \\ b & d \\ c \arrow[ur] \end{tikzcd}, \begin{tikzcd} a \\ b \arrow[r] & d \\ c \arrow[ur] \end{tikzcd}$$
, etc. respectively. We use the same mechanism of building posets for the next types of causalities.
- Bundle-Satisfaction Causality: bundles are satisfied by exactly one event: b,ab,ac are accepted causes but not abc.
- Minimal Causality: bundles are satisfied so that no subset of a cause is accepted. So b,ac are accepted but not ab or bc.
- Early Causality: the earliest bundle-satisfaction cause is accepted: b is accepted, while ac is not since event c took place after event b. This will be discussed formally in Section 3.3.1.
- Late Causality: (cf. [48], it will be skipped here).

The relation between the different kinds of causality is illustrated in Figure 2.7 as taken from [48, 49]. Given two DESs δ_1,δ_2, an arrow from *Lib.* to *Early* means that if δ_1,δ_2 are equivalent w.r.t. liberal-causality posets then they are equivalent w.r.t. early-causality posets, and so on. Furthermore, δ_1,δ_2 are equivalent w.r.t. early-causality posets if and only if they

[2]We omit commas here within traces for simplicity. This coincides with the original definition in [48].

$$\begin{tikzcd}
\text{Lib.} \arrow[dr, Rightarrow] & & \text{Late} \arrow[dl, Rightarrow] & \\
& \text{Early} \arrow[r, Leftrightarrow] & \text{Obs.} \arrow[r, Leftrightarrow] & \text{Traces} \\
\text{B-Sat.} \arrow[uu, Rightarrow] \arrow[ur, Rightarrow] & & \text{Min.} \arrow[ul, Rightarrow] &
\end{tikzcd}$$

Figure 2.7.: *The relation between the different kinds of posets in Dual Event Structure. An arrow means: if two structures are equivalent in terms of Late-causality posets for instance then they are equivalent in terms of Early-causality posets, and so on.*

are equivalent w.r.t. observational posets, and if and only if they are equivalent w.r.t. traces, since traces match early causality by definition.

The absence of the stability condition in Dual ESs makes them able to model any Stable ES. Thus Dual ESs can be proved, similarly to [43], to be strictly more expressive than Stable ESs. Additionally, Dual ESs that are based on Extended Bundle ESs are proved in [43] to be strictly more expressive than both Stable and Extended Bundle ESs.

2.7. Event Structures for Resolvable Conflict

Event Structures for Resolvable Conflicts (RCES) were introduced by van Glabbeek and Plotkin [76] to generalize former types of ESs and to give semantics to general Petri Nets. They allow to model the case where a and b cannot occur together until c takes place, i.e. initially a and b are in conflict until the occurrence of c resolves this conflict. A 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 behavior is defined by a transition relation between sets of events that is derived from the enabling relation $\vdash$. [3]

Definition 2.7.1. *An* Event Structure for Resolvable Conflict (RCES) *is a pair $\rho = (E, \vdash)$, where E is a set of* events *and $\vdash \subseteq \mathcal{P}(E)^2$ is the* enabling relation*.*

[3]Since ESs for resolvable conflicts are a generalization of Winskel's ESs mainly, the enabling relation symbol is the same as in Stable ESs.

$$\begin{tikzcd}
 & \{a\} \arrow[r] & \{a,b\} \arrow[dr] & \\
\emptyset \arrow[ur] \arrow[urr] \arrow[r] \arrow[drr] \arrow[dr] & \{b\} \arrow[ur] \arrow[rr] \arrow[dr] & & \{a,b,c\} \\
 & \{c\} \arrow[r] & \{b,c\} \arrow[ur] &
\end{tikzcd}$$

Figure 2.8.: *The transition graph of a RCES representing resolvable conflict.*

In [76] several versions of configurations are defined. Here we consider only reachable and finite configurations, that will be adequate to compare to variants of Dynamic Causality ESs in Chapter 3.

Definition 2.7.2. *Let $\rho = (E, \vdash)$ be an RCES and $X, Y \subseteq E$. Then $X \rightarrow_{\mathrm{rc}} Y$ iff $(X \subseteq Y \wedge \forall Z \subseteq Y \,.\, \exists W \subseteq X \,.\, W \vdash Z)$. The set of* configurations *of ρ is defined as $\mathrm{C}(\rho) = \{X \subseteq E \mid \emptyset \rightarrow^*_{\mathrm{rc}} X$
$\wedge\, X$ is finite$\}$, where $\rightarrow^*_{\mathrm{rc}}$ is the reflexive and transitive closure of $\rightarrow_{\mathrm{rc}}$.*

As an example consider the RCES $\rho = (E, \vdash)$, where $E = \{a, b, c\}$, $\{b\} \vdash \{a, c\}$, and $\emptyset \vdash X$ iff $X \subseteq E$ and $X \neq \{a, c\}$, adapted from [76]. It models the above described initial conflict between a and c that can be resolved by b.

In Figure 2.8 the respective transition graph is shown, i.e. the nodes are all reachable configurations of ρ and the directed edges represent $\rightarrow_{\mathrm{rc}}$. Note, because of $\{a, c\} \subset \{a, b, c\}$ and $\emptyset \nvdash \{a, c\}$, there is no transition from $\emptyset$ to $\{a, b, c\}$. One drawback of RCESs—compared to other ESs—is the lack of a graphical representation, since RCESs do not address individual events.

Semantics of RCESs is given through transition graphs. We consider two RCESs as equivalent if they have the same transition graphs. Families of configurations cannot be used for semantics of RCESs since they are not able to model resolvable conflict, due to Condition 2.2.2 of Definition 2.2.2 concerning the least upper bound of two configurations. This condition requires, for instance, that $\{a, c\}$ should be an element of the family of configurations of ρ, which cannot be the case. Note that, since we consider only reachable configurations, the transition equivalence defined below is denoted as reachable transition equivalence in [76].

Definition 2.7.3. *Two RCESs $\rho = (E, \vdash)$ and $\rho' = (E', \vdash')$ are* transition equivalent*, denoted by $\rho \simeq_{\mathrm{t}} \rho'$, iff $E = E'$ and $\rightarrow_{\mathrm{rc}} \cap \left(\mathrm{C}(\rho)\right)^2 = \rightarrow'_{\mathrm{rc}} \cap \left(\mathrm{C}(\rho')\right)^2$.*

We lift the definition of $\simeq_{\mathrm{t}}$ to denote transition equivalence between any two event structures of two (possibly different) kinds of ESs that define a transition relation each. To this end, we ignore event-set equivalence between the two structures as unreachable events might be introduced to one of the two ESs to model the other (cf. Chapter 3 for an example).

Definition 2.7.4. *Let δ_1, δ_2 be two ESs of two (possibely different) kinds of ESs, with two transition relations (defined on configurations) $\rightarrow_1, \rightarrow_2$, respectively. δ_1, δ_2 are said to be configuration-transition equivalent, denoted $\delta_1 \simeq_t \delta_2$, iff $\rightarrow_1 \cap (\mathrm{C}(\delta))^2 = \rightarrow_2 \cap (\mathrm{C}(\delta_2))^2$.*

Since RCESs are meant to be a generalization of Winskels ESs, RCESs are more expressive than Stable and Prime ESs, as proved in [76]. Furthermore RCESs are strictly more expressive than Prime and Stable ESs, since none of the latter is able to model a resolvable conflict. It is hinted in [76] how to model EBESs and other structures by RCESs, but not formally proved. However we prove such expressiveness results in Chapter 3 while comparing our new Dynamic Causality ESs to RCESs. To do so, we depend on the following generic definition and lemma, saying that any transition-based ES with certain properties could be naturally embedded into RCESs.

Definition 2.7.5. *Let μ be an ES with a transition relation $\rightarrow$ defined on configurations such that for all configurations X, Y, X', Y' of μ:*

- *$X \rightarrow Y$ implies $X \subseteq Y$*
- *$X \subseteq X' \subseteq Y' \subseteq Y$ implies that $X \rightarrow Y \implies X' \rightarrow Y'$*

Then $\mathrm{rces}(\mu) = \left(E, \{X \vdash Z \mid \exists Y \subseteq E \,.\, X \rightarrow Y \wedge Z \subseteq Y\}\right)$.

The following lemma shows that the resulting structure $\mathrm{rces}(\mu)$ is indeed a RCES that it is transition equivalent to μ.

Lemma 2.7.6. *Let μ be an ES that satisfies the conditions of Definition 2.7.5. Then* $\mathrm{rces}(\mu)$ *is a RCES and* $\mathrm{rces}(\mu) \simeq_t \mu$.

Proof. By Definition 2.7.1, $\mathrm{rces}(\mu)$ is a RCES.

Assume $X \rightarrow Y$. Then, by Definition 2.7.5, $X \subseteq Y$ and $X \vdash Z$ for all $Z \subseteq Y$.Then, by Definition 2.7.2, $X \rightarrow_{\mathrm{rc}} Y$.

Assume $X \rightarrow_{\mathrm{rc}} Y$. Then, by Definition 2.7.2, $X \subseteq Y$ and there is some $X' \subseteq X$ such that $X' \vdash Y$. By Definition 2.7.5 for rces(·), then $X' \vdash Y'$ for all $Y' \subseteq Y$. So there is a set $\widetilde{Y}$ such that $Y \subseteq \widetilde{Y}$ and $X' \vdash \widetilde{Y}'$ for each $\widetilde{Y}' \subseteq \widetilde{Y}$. Then, by Definition 2.7.5 for rces(·), it follows $X' \rightarrow \widetilde{Y}'$ and $X' \subseteq X \subseteq Y \subseteq \widetilde{Y}$. Finally, by the second property of Definition 2.7.5, $X \rightarrow Y$. □

$$\begin{tikzcd}
& & \text{Extended Bundle} \arrow[drr] & & \\
\text{Prime} \arrow[r] & \text{Bundle} \arrow[ur] \arrow[dr] & & & \text{Resolvable Conflict} \\
& & \text{Stable} \arrow[r] & \text{Dual} \arrow[ur] &
\end{tikzcd}$$

Figure 2.9.: *The expressiveness of variouse kinds of event structures from the literature, increasing from left to right.*

To compare RCESs with other ESs, which do not use transition graphs for their semantics, we use our Dynamic Causality ESs of Chapter 3 as a bridge, by defining semantics of the latter structures in different means (e.g. families of posets and transition graphs) and connecting these means through equivalence relations. For example, we use posets to prove that DESs can be embedded in a subset of Dynamic Causality ESs, that is proved to be less expressive than RCESs using transitions on the other hand. Then depending on the connection between the transition graph of our Dynamic Causality ES and their posets, we conclude that DESs are less expressive than RCES. This will be examined more formally in Chapter 3. Figure 2.9 illustrates the landscape of the major ESs mentioned in this chapter w.r.t. expressiveness.

2.8. Other Kinds of Event Structures

Baldan et al. defined a new event structure based on Prime ESs, called Asymmetric ES [8]. They replaced the classical symmetric conflict by an asymmetric one as in Extended Bundle ESs. It denotes a precedence between events, or *weak causality* as called by the authors. We skip this kind of event structures here as we cover the asymetric conflict—through Extended Bundle ESs—while investigating its influence on priority in Section 5.4 and how to model it using Dynamic Causality in Section 3.5.2.

Later in [7] Baldan et al. introduced *Inhibitor ES* based on Prime ES. The key idea was that an event can be disabled by a set of events and re-enabled again by other events. This can model the enabling relation of Prime ESs, the asymmetric conflict of Asymmetric ESs, and even the bundle causality of Bundle ESs. We refer to this kind of ESs In Chapter 3 to compare its different choices of re-enabling to the ones offered by Growing Causality ESs in Section 3.4 and Fully Dynamic Causality in Section 3.5.

In [13] *Flow* event structures were defined where the enabling relation is binary but not a partial order, allowing for cycles. Furthermore the conflict relation is not irreflexive anymore, such that self-conflicting events are impossible. In a configuration, each enabler of an event in that configuration is needed, unless it is in conflict with another enabler that exist in the configuration. This offered more sophisticated causality model than Prime ESs. We skip *Flow* ESs here since the choices they offer in enabling are considered as a special case of the disjunctive causality offered by Stable ESs which are more expressive [13] and covered in this work.

Dual ESs of [43] were extended in the same work [43] by adding a new relation, namely the interleaving relation, under the name of Extended Dual Event Structures. Two events are interleaved when one of them must occur before the other, but not together. We do not consider Extended Dual Event Structures in this work as a more recent publication appeared, namely [48] by Langerak et al, in 1997 to give a more precise and even different definition to posets in Dual ESs which influenced the semantics of Extended Dual ESs accordingly. However we apply interleaving in our work in Chapter 3 between causality modifiers in Dynamic Causality ESs.

2.9. Quantitative and Qualitative Extensions

In [43] as well as in [44] Katoen et al. added the concept of time delay to Extended Bundle ESs. A bundle is associated with a time unit, and so is an initially enabled event, representing the maximum delay of an event to occur once it is enabled. So an event trace in timed ES is a sequence of pairs (e,t): an event and its occurrence time. Intuitively, the earlier an event in a trace takes place, the earlier its occurrence time is. Besides, the occurrence time of an event is greater than or equal to the its enabling time. Casley et al. in [24] added time delay to events. They defined operations for timed events regarding concurrency, e.g. disjoint union and concatenation.

In addition, Katoen et al. introduced [44, 43] the concept of *urgent events* based on their timed Extended Bundle ESs. Urgent events must happen once they are enabled, exactly after a certain amount of time (delay). That was used to model timeouts in the process algebra LOTOS. In an Urgent Event Structure, a trace should be complete, i.e. including all the urgent

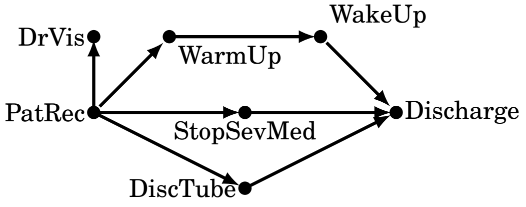


Figure 2.10.: *The default plan for the cardiac patient after she was admitted to the Cardiac Intensive Care Unit.*

events that have been enabled in this trace, and will occur for sure.

Furthermore, a more complex and complete modeling of timed ESs was defined in [43], capturing not only a maximum occurrence time (with delay) and urgency, but also a minimum one, by defining a range of occurrence times or a set of discrete time points.

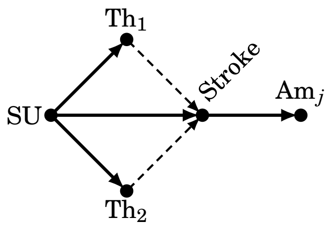
Finally, in [43, 79] probabilistic ESs were defined to give a stochastic variant of ESs. In [43] an event can have a probability of occurrence given that the set of events having the probability sum of 1 (a cluster) must be enabled together and by the same enablers.

2.10. Application in Dynamic-Coalition Workflows

Consider the use case of the cardiac patient of Section 1.1.1. The default plan was to warm the patient up, and then to wake her up. In the meanwhile, severe medication should be stopped, and the tube should be disconnected. Then the patient can be discharged from the CICU. This can be modelled by a PES as illustrated in Figure 2.10, since causality is conjunctive. The fact that StopSevMed, WakeUp and DiscTube are neither related w.r.t. causality nor to conflict means that they are concurrent events, and thus need not to occur in a specific order, rather all of them have to occur before Discharge occurs.

Let us consider another example, where other kinds of ESs can be used, such as Extended Bundle ESs. This example is taken in the research training group *SOAMED* by observing the treatment process of stroke patients: *After a stroke patient is transferred to the stroke unit, the latter works on* *stabilizing the patient's situation. Then rehabilitation starts by therapists to handle the consequences. During the rehabilitation, if the patient suddenly gets a stroke again, the ambulance needs to involve and interrupt the work of therapists.*

This can be modelled by an EBES as illustrated aside. The disabling $\mathrm{Th}_1 \rightsquigarrow \mathrm{Stroke}$ and the disabling $\mathrm{Th}_2 \rightsquigarrow \mathrm{Stroke}$ model the case that once the patient gets a stroke again, therapists cannot proceed anymore, i.e. the sequence Stroke, Th_1 is not a trace. Furthermore, event Am_j could be modelled as an *urgent* event [44] that must take place once it is enabled, during a given time-out.



2.10.1. Modeling DC Membership

Since we consider the workflow of a DC, we model membership through its events, represented by the members' *join* and *leave* events, which are enough to show changes in DC membership, and thus dynamicity. For instance $\mathrm{Th}_1, \mathrm{Th}_2, \ldots$ in the last example represent the join events of various members of the stroke-patient DC. The same applies to all DC examples in this work.

2.10.2. Applying Quantitative and Qualitative Extensions

The timing extension of ESs mentioned earlier could be applied in general in the medical domain, where time is vital in many cases e.g. a stroke. The same applies for probabilities and stochastic processes that could be applied there for statistics and prediction. This will be examined in more details in Chapter 5.

2.10.3. Modeling Repeated Actions in Event Structures

Events in ESs are single instances that cannot be repeated. Many actions in life are often repeated. To model that, several events with the same label—denoting the same action—are used. For instance, an action X that is repeated in a loop, would be modeled by a series of events $e_1, e_2, \ldots$ that are causally dependent (i.e. $e_1 \rightarrow e_2 \rightarrow \ldots$). Each event then would represent a single iteration, where iteration i cannot occur before iteration $i-1$, which is maintained by the causality. For more on this topic, please refer to [41, 82, 77] for unfolding of Petri Nets and their representation in ESs.

2.10.4. Modeling Business Process Schemas and Instances

In Business Processes, a distinction between the process type, known as a *schema*, and *instances* of that type exists. A schema holds the definition of flow and other elements in the process. An instance of a schema represents a running process that follows the definition of that schema. Thus an instance has a state of execution.

A process schema can be represented by an event structure δ, where definitions could be mapped to causality and other relations in the structure, and activities could be mapped to different events as described in the section before. An instance, or any running workflow, is then represented by a pair (δ, H) where δ is an ES representing the schema of the instance, and H represent the state of the instance.

Since the state of a system in ESs is represented through the set of events that have taken place, then $H \in \mathrm{C}(\delta)$. To hold more information, H could be a trace or a poset, holding execution order, i.e. $H \in \mathrm{T}(\delta)$ or $H \in \mathrm{P}(\delta)$. We will use this representation in Chapter 4 to define the evolution of running workflows.

2.10.5. Modeling Nested Coalitions with Action Refinement

DCs could be nested as proved by visits to and observations in hospitals. A coalition might include the main events and involvements of different members, while each of the members can be itself a dynamic coalition. For instance, imagine a DC that is set to rescue a stroke patient, where it requires the join of the ambulance, the emergency room to do a test, and finally the stroke unit. The test made by the emergency room can be seen as a collaboration between nurses and doctors to do the test, i.e. a coalition. The same applies for the stroke unit. If the involvement of the stroke unit must follow the test of the emergency room, at the main-DC level, then the

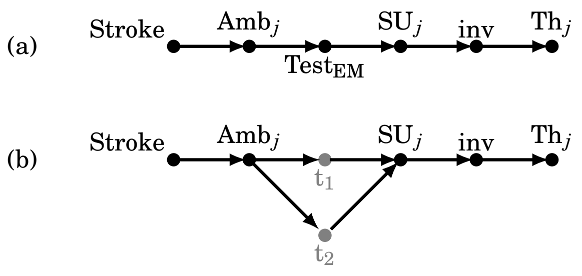


Figure 2.11.: *A Prime Event Structure and its refinement, modeling nested coalitions.*

stroke unit cannot proceed before the collaboration in the emergency room is done, at the sub-DC level.

Assume the involvements of the emergency room and the stroke unit at the main-DC level are represented by two events $\mathrm{Test}_{\mathrm{EM}}, \mathrm{SU}_j$ respectively, such that $\mathrm{Test}_{\mathrm{EM}} \leq \mathrm{SU}_j$ in a PES. Assume also that the collaboration in the emergency room is represented by an ES with events $I_1 = \{\mathrm{t}_1, \mathrm{t}_2\}$. Then SU_j must not take place before events of I_1. To do that, the causality $\mathrm{Test}_{\mathrm{EM}} \leq \mathrm{SU}_j$ must be inherited at the sub-DC level, such that $\mathrm{t}_1 \leq \mathrm{SU}_j$ and $\mathrm{t}_2 \leq \mathrm{SU}_j$. This is provided through *Action Refinement* [63], where each event of a given structure π_1 can be refined to a complete minor ES, and then such minor ESs are aggregated together into one ES π_2 such that causality and conflicts are inherited [74]. In this way, a configuration in π_1 can be refined to a set of configurations in π_2, and each configuration in π_2 would match a configuration in π_1.

For instance, structure (b) in Figure 2.11 is a refinement of structure (a). The set {Stroke, Amb_j, t_2, SU_j} is not a configuration at structure (b). On the other hand, $\{\mathrm{Stroke}, \mathrm{Amb}_j, \mathrm{t}_1, \mathrm{t}_2, \mathrm{SU}_j\}$ is a configuration. Furthermore, it is a refinement of the configuration {Stroke, Amb_j, $\mathrm{Test}_{\mathrm{EM}}$, SU_j}. A configuration like $\{\mathrm{Stroke}, \mathrm{Amb}_j, \mathrm{t}_1\}$ shows that the test is not complete yet, and is considered a refinement of $\{\mathrm{Stroke}, \mathrm{Amb}_j, \mathrm{Test}_{\mathrm{EM}}\}$. Each configuration in (b) should be a refinement of a configuration in (a), as well as the refinement of each configuration in (a) should be a configuration in (b) [74].

Action refinement allows for looking at systems in different levels of abstraction, by allowing actions which are atomic at one level, to be complex processes at another level. This kind of abstraction helps focusing on the overall picture on the higher abstraction level, and dealing with details on a lower abstraction level. Moving up through the abstraction levels is called *Abstraction* [27], and moving down is called *Refinement* [63]. Studies in different models were made in the literature w.r.t. refinement and abstraction, e.g. event-based systems, action trees, process algebra and others [32, 31, 73, 64]. Such a technique can be used while designing DCs to model nested coalitions, and to look at coalitions through different abstraction levels, avoiding unnecessary details.

2.10.6. Limitations and Missing Features

Despite of their great applicability, Event Structures are an example of a formalism where rules are defined to decide about the possible system runs, and possibly try to derive all possible system runs. Thus they fit into the category of static planning, and miss a formal mechanism that shows how to evolve during runtime and transit successfully to a new ES. In that sense, ESs do not reflect the dynamicity of DCs, nor show their unique characteristics.

In Chapter 4 we provide a model that can be applied to ESs to provide the flexibility needed in a robust way that guarantees error freeness and makes it suitable to capture the nature of a DC workflow. Additionally, in Chapter 5 we show some of the missing qualitative extensions that make Event Structures applicable in the domain of our thesis, the healthcare sector, not only for dynamic coalitions, but also for processes and workflows in general.

3. Pre-Planned Changes in Workflows: Dynamic Causality in Event Structures

3.1. Introduction

Modern process-aware systems emphasize the need for flexibility into their design to adapt to changes [80]. One form of flexibility is the ability to change the workflow during the runtime of the system deviating from the default path, due to exceptions or changes in regulations. Such changes could be ad-hoc or captured at build time of the system [61]. For instance—as adapted from [80]—*during the treatment process, and for a particular patient, a planned computer tomography must not be performed in case she has a cardiac pacemaker. Instead, an X-ray activity shall be performed.*

In this example, the activity depending on the tomography will depend instead on the X-ray instead. Such a change can be achieved through more primitive ones: causality between tomography and the next activity should be dropped, and a new causality between X-ray and the next activity shall be created. This can be modelled by disjunctive causality, but then it will not be clear which path is the default one (the dropped causality in our case), and which path is the exceptional one (the created causality) [61]. Besides, other more complex examples cannot be modelled easily with static causality as we will show by comparing expressive powers in Section 3.5. Furthermore, if an event triggers several changes in a structure, it will be hard to notice all these changes in a non-dynamic causality approach.

In this chapter, we allow the main ingredient of ESs, namely causality, to change during a system run. In other words, the flow of events might change based on the occurrence of certain events. Therefore, we call it history-dependent causality, and we call the events changing the causality of other events as modifiers. We will see also how dynamicity in causality will be able to model adding and dropping of conflicts, and other relations like disabling.

ESs that do neither allow for alternatives nor for changes in causality, e.g. PESs [82], solve the problem by duplicating events and assign the same label to the different copies that have different causal predecessor. We argue that duplicating events increases complexity in the model, especially if changes are composed, i.e. depend on each other. On the other hand, allowing event causality to change, like in our model, avoids duplication and thus reduces complexity in the model.

Since causality is changed by the occurrence of some events that are part of the ES and since the changes are declared in the structure, this chapter would contribute to the pre-planned changes that are foreseen. The other kind of changes, namely the ad-hoc one, will be covered in the next chapter.

Overview We will separate the idea of dropping (i.e. shrinking) causality from adding (i.e. growing) causality and study each one separately first, and then combine them. In Section 3.3 we define *Shrinking Causality Event Structures* (SESs), and compare their expressive power with other types of ESs. In Section 3.4 we do the same for *Growing Causality Event Structures* (GESs). In Section 3.5 we combine both concepts within the *Dynamic Causality Event Structures* (DCESs). But first, let us examine the related work, w.r.t. changes in causality and other relations in various models as well as in ESs.

Related Work The concept of *local independence* [40, 72] was defined by Hoogers et al. as a generalization of independence in traces, to give trace semantics to Petri nets. Local independence meant that actions could be independent from each other after a given history. Similarly Local Event Structures were defined [41]. Comparing to our work we provide a mechanism for independence of events, through the growing and shrinking causality, while the former related works abstract from the way actions or events become independent. In [76], van Glabbeek and Plotkin introduced RCESs, where conflicts can be resolved or created by the occurrence of other events. This dynamicity of conflicts is complementary to our own approach. We will see that DCESs and RCESs are incomparable but—similarly to RCESs—DCESs are more expressive than many other types of ESs.

3.2. Prime Event Structures with Causal Cycles

If we allow one to add or drop causal dependencies, it will be hard to maintain the conflict heredity and the transitivity and reflexivity of enabling. Therefore we do not consider the partial order property nor the axiom of conflict heredity in our definition of PESs. The same applies for the finite causes property which will be covered through finite configurations, like Definition 3.3.3 later on. However the following version of PESs has the same expressive power as PESs of Section 2.2, in terms of finite configurations which we limit our concern to.

Definition 3.2.1. *A* Prime Event Structure (PES) *is a triple* $\pi = (E, \#, \rightarrow)$*, where:*

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

As a result, the definition of a configuration should take care of enabling cycles, which makes events impossible, and thus unreachable in configurations.

Definition 3.2.2. *Let $\pi = (E, \#, \rightarrow)$ be a PES. A set of events $C \subseteq E$ is a* configuration *of π iff:*

- $\forall e, e' \in C \,.\, \neg\left(e \# e'\right)$*, i.e. conflict-free*
- $\forall e, e' \in E \,.\, e \rightarrow e' \wedge e' \in C \implies e \in C$*, i.e. left-closed*
- $\forall n \geq 1 \,.\, \nexists e_1, \ldots, e_n \in C \,.\, e_1 \rightarrow \ldots \rightarrow e_n \rightarrow e_1$*, i.e. $\rightarrow \cap\, C^2$ is acyclic*

An event e is denoted as *impossible* in a PES if it does not occur in any of its configurations. Events can be impossible because of enabling cycles, or an overlapping between the enabling and the conflict relation, or because of impossible predecessors as illustrated in Figure 3.1.

To be compatible and comparative to the the other ESs in this chapter as well as to ESs for resolvable conflicts, a transition relation $\rightarrow_{\mathrm{p}}$ can be defined for PESs with cycles.

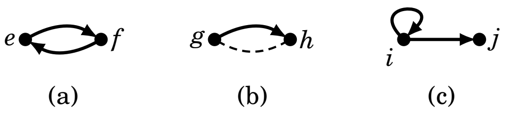


Figure 3.1.: *Impossible events in the variant of PESs with causal cycles.*

Definition 3.2.3. *Let $\pi = (E, \#, \rightarrow)$ be a PES and $X, Y \subseteq E$. Then the transition relation $\rightarrow_{\mathrm{p}}$ is defined as $X \rightarrow_{\mathrm{p}} Y$ iff:*

- $X \subseteq Y$
- $\forall e, e' \in Y \,.\, \neg\big(e \# e'\big)$
- $\forall e \in Y \setminus X \,.\, \{e' \in E \mid e' \rightarrow e\} \subseteq X$

We denote the reflexive and transitive closure of $\rightarrow_{\mathrm{p}}$ as $\rightarrow_{\mathrm{p}}^{*}$.

For instance, the graphs in (a), (b), (c) below are the transition relations for the structures in Figure 3.1 (a), (b) and (c) respectively, where reflexive arrows are not shown for simplicity.

(a)
$$\begin{tikzcd} & \{e\} \arrow[dr] & \\ \emptyset & & \{e,f\} \\ & \{f\} \arrow[ur] & \end{tikzcd}$$

(b)
$$\begin{tikzcd} & \{g\} \\ \emptyset \arrow[ur] & \end{tikzcd}$$

(c)
$$\begin{tikzcd} \emptyset & \{i\} \arrow[r] & \{i,j\} \end{tikzcd}$$

Sets of events like $\{e\}$, $\{f\}$, $\{e,f\}$ are unreachable configurations, containing impossible events. It can be proved that the configurations of Definition 3.2.2 can be obtained by considering finite sets that are transitable from $\emptyset$, successively. The proof will be clear when defining the transition relation for different Dynamic-Causality ESs.

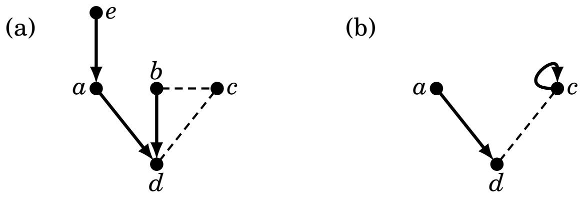
Since this version of PESs allows for modeling impossible events through cycles, the definition of the remainder is slightly different from the one in Definition 2.2.3. The difference is that after a given system run H, events that were in conflict with one of the events of H would become impossible in the remainder instead of being dropped. A formal definition would be as follows:

Definition 3.2.4. *Let $\pi = (E, \#, \rightarrow)$ be a PES, and let $H \in \mathrm{C}(\pi)$ be a configuration of π. The remainder of π after H is $\pi[H] = \big(E', \#', \rightarrow'\big)$, where:*

- $E' = E \setminus H$

- $\#' = \# \cap E'^2$
- $\rightarrow' = \left(\rightarrow \cap E'^2\right) \bigcup \left\{(e,e) \in E'^2 \mid \exists e' \in H \,.\, e' \# e\right\}$

For instance, the structure in (b) below shows the remainder of the PES in (a), after the history $H = \{e, b\}$.



It can be proved that $\pi[H]$ is a PES. Furthermore, the consistency between a structure and its remainder can be seen from the next lemma regarding transitions (cf. Appendix A.1 for proofs of this chapter).

Lemma 3.2.5. *Let π be a PES, $H \in \mathrm{C}(\pi)$. Then:*

$$\forall X \subseteq E \setminus H \,.\, X \in \mathrm{C}(\pi[H]) \iff H \rightarrow_{\mathrm{p}}^{*} H \cup X.$$

3.3. Shrinking Causality

Now we add a new relation that represents the removal of causal dependencies as a ternary relation between events $\rhd \subseteq E^3$. For instance $(a,b,c) \in \rhd$, denoted as $[a \rightarrow c] \rhd b$, models that a is dropped from the set of causes of c by the occurrence of b.

The dropping is visualized in Figure 3.2 (a) by a dashed empty-head arrow, from the initial cause $a \rightarrow b$ towards its dropper c. We add this relation to PESs and denote the result as shrinking causality event structures.

Definition 3.3.1. *A* Shrinking causality Event Structure (SES) *is a pair $\sigma = (\pi, \rhd)$, where*

- *$\pi = (E, \#, \rightarrow)$ is a PES and*
- *$\rhd \subseteq E^3$ is the* shrinking causality *relation*

such that $\left[e \rightarrow e''\right] \rhd e'$ implies $e \rightarrow e''$ for all $e, e', e'' \in E$.

Sometimes we expand $(\pi, \rhd)$ and write $(E, \#, \rightarrow, \rhd)$. For $[a \rightarrow c] \rhd b$ we call b the modifier, c the target, and a the contributed event. We denote the

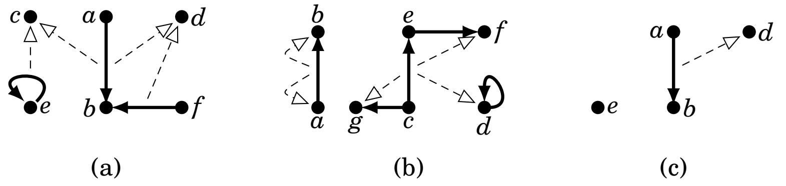


Figure 3.2.: *A Shrinking-Causality ES with shared and multiple droppers in* (a)*, with trivial droppings in* (b)*, and the remainder of* (a) *after* $H = \{c, f\}$ *in* (c).

set of all modifiers that drop a as cause from c by $[a \rightarrow c] \triangleright$. We refer to the set of dropped causes of an event w.r.t. a specific history by the function $\mathrm{dc} \subseteq \mathcal{P}(E) \times E \times \mathcal{P}(E)$ defined as: $\mathrm{dc}(H,e) = \left\{e' \mid \exists d \in H \,.\, \left[e' \rightarrow e\right] \triangleright d\right\}$. Besides, we refer to the initial causes of an event by the function $\mathrm{ic} \subseteq E \times \mathcal{P}(E)$ such that: $\mathrm{ic}(e) = \left\{e' \mid e' \rightarrow e\right\}$.

The semantics of a SES can be defined based on posets similar to BESs, EBESs, or DESs or based on a transition relation similar to RCESs. We consider first the transition relation, then the posets in Section 3.3.1.

Definition 3.3.2. *Let* $\sigma = (E, \#, \rightarrow, \triangleright)$ *be a SES. A* trace *of* σ *is a sequence of distinct events* $t = e_1, \ldots, e_n$ *with* $\overline{t} \subseteq E$ *such that:*

- $\forall 1 \leq i, j \leq n \,.\, \neg\left(e_i \# e_j\right)$
- $\forall 1 \leq i \leq n \,.\, \left(\mathrm{ic}(e_i) \setminus \mathrm{dc}\left(\overline{t_{i-1}}, e_i\right)\right) \subseteq \overline{t_{i-1}}$

Then $C \subseteq E$ *is a* traced-based configuration *of* σ *if there is a trace* t *such that* $C = \overline{t}$*. Let* $\mathrm{C}_{Tr}(\sigma)$ *denote the set of traced-based configurations and* $\mathrm{T}(\sigma)$ *the set of traces of* σ*.*

The combination of initial and dropped causes ensures that a for each $e_i \in \overline{t}$ all its initial causes are either predecessors of e_i or are dropped by other events preceding e_i.

The transition relation of SESs has to be based on the conflict relation as well as the enabling relation which are separate unlike RCESs.

Definition 3.3.3. *Let* $\sigma = (E, \#, \rightarrow, \triangleright)$ *be a SES and* $X, Y \subseteq E$*. Then* $X \rightarrow_{\mathrm{s}} Y$ *iff:*

- $X \subseteq Y$
- $\forall e, e' \in Y \,.\, \neg\left(e \# e'\right)$
- $\forall e \in Y \setminus X \,.\, \left(\mathrm{ic}(e) \setminus \mathrm{dc}(X, e)\right) \subseteq X$

We denote the reflexive and transitive closure of $\to_{\mathrm{s}}$ *as* $\to_{\mathrm{s}}^{*}$.

Again we consider the reachable and finite configurations w.r.t. to $\to_{\mathrm{s}}$.

Definition 3.3.4. *Let* $\sigma = (E, \#, \to, \rhd)$ *be a SES. The set of all configurations of* σ *is* $\mathrm{C}(\sigma) = \{X \subseteq E \mid \emptyset \to_{\mathrm{s}}^{*} X \wedge X \textit{ is finite }\}$.

Both definitions of configurations coincide.

Lemma 3.3.5. *Let* σ *be a SES. Then* $\mathrm{C}_{Tr}(\sigma) = \mathrm{C}(\sigma)$.

Dropping causal predecessors can be seen more concretely in the concept of the remainder. Assume a system run H took place, then pairs dropped by an event of H disappear from the initial causality relation of the remainder. For instance, Figure 3.2 (c) shows the remainder of Figure 3.2 (a) after $H = \{c, f\}$. This can be formally defined as follows.

Definition 3.3.6. *Let* $\sigma = (E, \#, \to, \rhd)$ *be a SES, and let* $H \in \mathrm{C}(\sigma)$ *be a configuration of* σ. *The remainder of* σ *after* H *is* $\sigma[H] = (E', \#', \to', \rhd')$, *where:*

- $E' = E \setminus H$
- $\#' = \# \cap E'^2$
- $\to' = \Big(\big(\to \cap E'^2 \big) \setminus \{ (c,t) \in E'^2 \mid c \in \mathrm{dc}(H,t) \} \Big) \cup \{ (e,e) \in E'^2 \mid \exists e' \in H \,.\, e' \# e \}$
- $\rhd' = \big(\rhd \cap E'^3 \big) \setminus \{ (e,d,e) \in E^3 \mid \exists e' \in H \,.\, e' \# e \}$

It can be proved that $\sigma[H]$ is a SES. The enabling relation is defined as in Definition 3.2.4. On the other hand, the definition of $\rhd'$ can be explained as follows. Assume that $[c \to t] \rhd m$, then this triple would be excluded in the remainder when $t \in H$, or $m \in H$ since it is not dynamic anymore but affects $\to'$, or $c \in H$ since $(c,t) \notin \to'$. On the other hand, the consistency between a structure and its remainder through the following lemma about transitions. Finally, events conflicting with H should have no chance to be dropped, thus we exclude them from $\rhd'$ in case they already exist in $\rhd$.

Lemma 3.3.7. *Let* σ *be a SES,* $H \in \mathrm{C}(\sigma)$. *Then:*

$$\forall X \subseteq E \setminus H \,.\, X \in \mathrm{C}(\sigma[H]) \iff H \to_{\mathrm{s}}^{*} H \cup X.$$

3.3.1. Shrinking Causality versus Disjunctive Causality

Consider the shrinking-causality $[c \to t] \rhd d$. It models the case that initially t causally depends on c but this dependency can be dropped by an occurrence of d. Thus for t to occur either c must occur or d must. This is a disjunctive causality as modeled by DESs. In fact $[c \to t] \triangleright d$ corresponds to the bundle $\{c, d\} \mapsto t$. We prove that we can map each SES into a DES with the same behavior and vice versa.

To translate a SES into a DES we create a bundle for each initial causal dependence and add all its droppers to the the bundle set.

Definition 3.3.8. *Let $\sigma = (E, \#, \to, \triangleright)$ be a SES.*
Then $\mathrm{des}(\sigma) = (E, \#, \mapsto)$*, where $S \mapsto y$ iff: $S \subseteq E$, $y \in E$ and $\exists x \in E \,.\, x \to y \land S = \{x\} \cup [x \to y] \triangleright$.*

We can prove that the above translation from SES into DES shows that for each SES there is a DES with exactly the same traces and configurations. But the most discriminating behavioral semantics of DESs used in literature are families of posets. Thus, in order to prove that SESs and DESs have the same expressive power, we even show that our translation preserves posets.

As shown in Figure 2.7, early-causality equivalence and trace equivalence coincide for Dual ESs. Thus we concentrate on the early causality of DESs [48]. Similarly, we concentrate on early causality of SESs.

Definition 3.3.9. *Let $\sigma = (E, \#, \to, \triangleright)$ be a SES, $t = e_1, \ldots, e_n$ one of its traces, and $1 \leq i \leq n$. A set U is a cause of e_i in t iff:*

- $\forall e \in U \,.\, \exists 1 \leq j < i \,.\, e = e_j$,
- $\left(\mathrm{ic}(e_i) \setminus \mathrm{dc}(U, e_i)\right) \subseteq U$*, and*
- *U is the earliest set satisfying the previous two conditions.*

Let $\mathrm{P_s}(t)$ be the set of posets obtained this way for t.

As adopted from [48], given a trace $t = e_1, \ldots, e_n$ and two sets $C, C' \subseteq \bar{t}$, C is said to be earlier than C' iff the maximal index in t of the events in $C \setminus C'$ is smaller than the maximal index in t of the events in $C' \setminus C$, where the maximal index of $\emptyset$ is considered 0. In [49] a procedure is defined to detect how early a cause is, based on binary numbers.

To show that $\simeq_{\mathrm{p}}$ and $\simeq_{\mathrm{t}}$ coincide on SESs we make use of the result that $\simeq_{\mathrm{p}}$ and trace equivalence coincide on DESs (cf. [48]) and that SESs are as expressive as DESs, as shown later.

Theorem 3.3.10. *For each SES σ there is a DES $\delta = \mathrm{des}(\sigma)$, such that $\sigma \simeq_{\mathrm{p}} \delta$.*

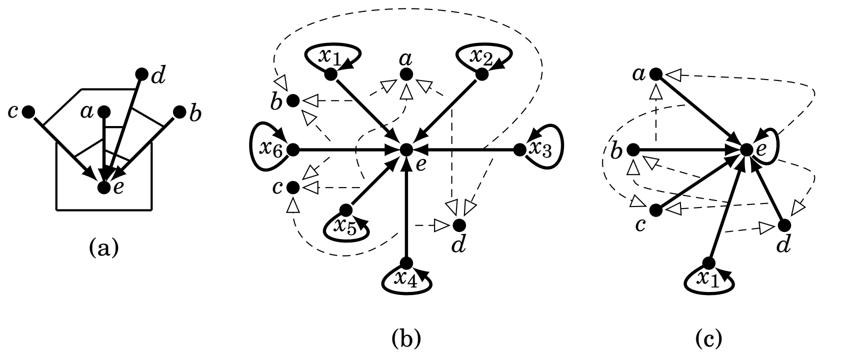


Figure 3.3.: *A Dual ES with bundles more bundles than events in* (a)*, its canonical-translation poset-equivalent SES* (b)*, and another poset-equivalent SES with minimal fresh events in* (c).

An extra modeling feature of SES comparing to Disjunctive Causality: One can note that both $[c \rightarrow t] \triangleright d$ and $[d \rightarrow t] \triangleright c$ correspond to the same bundle $\{c,d\} \mapsto t$. In SES one would be able to distinguish between the initial causality and its alternative, namely the dropper. While a DES would not allow that due to symmetry. This will be used in Section 3.6 to distinguish between Regular execution paths, and exceptional ones [61] in workflows.

In the opposite direction we map each DES into a set of similar SESs such that each SES in this set has the same behavior as the DES. Intuitively we have to choose an initial dependency for each bundle (out of the bundle set) and to translate the rest of the bundle set into droppers for that dependency. Unfortunately the bundles that point to the same event are not necessarily disjoint. Consider for example $\{a,b\} \mapsto e$ and $\{b,c\} \mapsto e$. If we choose $b \rightarrow e$ as initial dependency for both bundles to be dropped as $[b \rightarrow e] \triangleright a$ and $[b \rightarrow e] \triangleright c$, then $\{a,e\}$ is a configuration of the resulting SES but not of the original DES. So we have to ensure that we choose distinct events as initial causes for all bundles pointing to the same event.

The easiest way to ensure, that for all bundles distinct events are chosen, is to use fresh impossible events. More precisely, for each bundle $X_i \mapsto e$ we choose a fresh event x_i as initial cause $x_i \rightarrow e$, make it impossible by a self-loop $x_i \rightarrow x_i$, and add all events d of the bundle X_i as droppers $[x_i \rightarrow e] \triangleright d$.

Definition 3.3.11. *Let $\delta = (E, \#, \mapsto)$ be a DES, $\{X_i\}_{i \in I}$ an enumeration of its bundles, and $\{x_i\}_{i \in I}$ a set of fresh events, i.e. $\{x_i\}_{i \in I} \cap E = \emptyset$. Then* $\text{ses}(\delta) = \left(E', \#, \rightarrow, \triangleright\right)$*, where:*

- $E' = E \cup \{x_i\}_{i \in I}$
- $\rightarrow = \{x_i \rightarrow e \mid X_i \mapsto e\} \cup \{x_i \rightarrow x_i \mid i \in I\}$
- $\triangleright = \{[x_i \rightarrow e] \triangleright d \mid d \in X_i \land X_i \mapsto e\}$

Of course it can be criticized that the translation adds events. But as Figure 3.3 and the next lemma show, it is not always possible to translate a DES into a SES without additional impossible events.

Lemma 3.3.12. *There are DESs $\delta = (E, \#, \mapsto)$, e.g.*

$$\delta = \left(\{a, b, c, d, e\}, \emptyset, \{\{x, y\} \mapsto e \mid x, y \in \{a, b, c, d\} \land x \neq y\}\right)$$

that cannot be translated into a SES $\sigma = \left(E, \#', \rightarrow, \triangleright\right)$ such that $\mathrm{T}(\delta) = \mathrm{T}(\sigma)$.

Of course the above lemma implies that no translation of the above DES, maintaining the same set of events, can result into a SES with the same posets or even configurations.

However, because the x_i are fresh, there are no droppers for the self-loops $x_i \rightarrow x_i$ in $\text{ses}(\delta)$. So the translation ensures that all events in $\{x_i\}_{i \in I}$ remain impossible forever in the resulting SES. In fact we show again that the DES and its translation have the exactly same traces and configurations. Moreover the DES and its translation have exactly the same posets.

Theorem 3.3.13. *For each DES δ there is a SES $\sigma = \text{ses}(\delta)$, such that $\delta \simeq_{\mathrm{p}} \sigma$.*

3.3.2. Expressiveness of Shrinking-Causality Event Structures

From the two way translation from SESs to DESs and vise versa, it can be concluded that both kinds of ESs are equally expressive.

Theorem 3.3.14. *SESs are as expressive as DESs.*

In Appendix B we show that each SES σ and its translation $\text{des}(\sigma)$ as well as each DES δ and its translation $\text{ses}(\delta)$ have the same posets considering liberal, minimal, or late causality instead of early causality. Thus

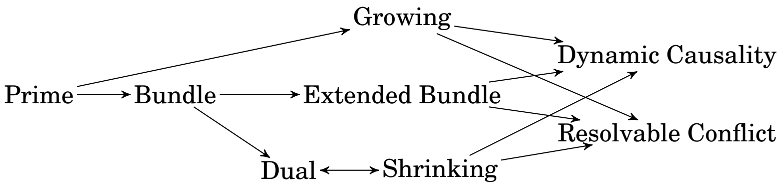


Figure 3.4.: *Expressiveness of Event Structures including the structures of this chapter, where transitive arrows are omitted and unconnected types of ESs are incomparable regarding expressiveness.*

the concepts of SESs and DESs are not only behaviorally equivalent but—except for the additional impossible events—also structurally closely related.

Since SESs are as expressive as DESs w.r.t. families of early-causality posets, then early-causality poset equivalence in SESs and trace equivalence coincides in a similar way to DESs.

Corollary 3.3.15. *Let σ_1, σ_2 be two SES. Then $\sigma_1 \simeq_p \sigma_2 \iff \mathrm{T}(\sigma_1) = \mathrm{T}(\sigma_2)$.*

Then $\simeq_p$, $\simeq_t$, and trace-equivalence coincide on SESs.

Theorem 3.3.16. *Let σ, σ' be two SESs. Then:*

$$\sigma \simeq_p \sigma' \iff \sigma \simeq_t \sigma' \iff \mathrm{T}(\sigma) = \mathrm{T}\left(\sigma'\right).$$

As shown before, SESs allow to model disjunctive causality, without any stability condition, which is not provided by EBESs. On the other hand the asymmetric conflict of an EBES cannot be modeled with a SES. Hence EBESs and SESs are incomparable. Appendix A.2 proofs that by concrete counterexamples.

Theorem 3.3.17. *SESs and EBESs are incomparable.*

By Lemma 2.7.6, the translation of each SES as in Definition 2.7.5 results into a transition-equivalent RCES.

Lemma 3.3.18. *For each SES σ there is a RCES ρ, such that $\sigma \simeq_t \rho$.*

To prove that SESs are strictly less expressive, we show that there are some RCESs that cannot be translated into a transition-equivalent SES.

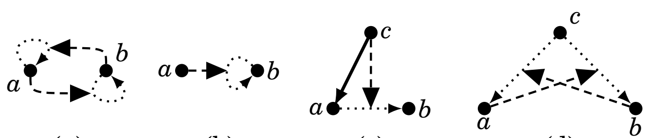


Figure 3.5.: *GESs modeling a conflict in* (a)*, a disabling in* (b)*, a temporary disabling in* (c)*, and a resolvable conflict in* (d)*.*

Lemma 3.3.19. *There is no transition-equivalent SES to the RCES*

$$\rho_\sigma = \Big(\{e,f\}, \big\{\emptyset \vdash \{e\}, \emptyset \vdash \{f\}, \{f\} \vdash \{e,f\}\big\}\Big).$$

Hence SESs are strictly less expressive than RCESs.

Theorem 3.3.20. *SESs are strictly less expressive than RCESs.*

3.4. Growing Causality

As in SESs we base our extension for growing causality on the PESs of Section 3.2. We add the new relation $\blacktriangleright \subseteq E^3$, where $(a,c,b) \in \blacktriangleright$, denoted as $c \blacktriangleright [a \rightarrow b]$, models that c adds a as a cause for b. Thus c is a condition for the causal dependency $a \rightarrow b$.

The adding is visualized in Figure 3.5(d) by a dashed line with a filled head from the modifier c to the added dependency $a \rightarrow b$, which is dotted denoting that this dependency does not exist initially (In this example there is an additional causality $c \rightarrow a$).

Definition 3.4.1. *A* Growing causality Event Structure (GES) *is a pair* $\gamma = (\pi, \blacktriangleright)$*, where:*

- $\pi = (E, \#, \rightarrow)$ *is a PES and*
- $\blacktriangleright \subseteq E^3$ *is the* growing causality *relation*

such that $\forall e, e', e'' \in E \,.\, e' \blacktriangleright [e \rightarrow e''] \implies \neg(e \rightarrow e'')$.

We refer to the causes added to an event w.r.t. a specific history by the function $\mathrm{ac} \subseteq \mathcal{P}(E) \times E \times \mathcal{P}(E)$, defined as $\mathrm{ac}(H, e) = \big\{e' \mid \exists a \in H \,.\, a \blacktriangleright [e' \rightarrow e]\big\}$, and to the initial causality by the function ic as defined in Section 3.3.

Definition 3.4.2. *Let $\gamma = (E, \#, \rightarrow, \blacktriangleright)$ be a GES. A* trace *of γ is a sequence of distinct events $t = e_1, \ldots, e_n$ with $\overline{t} \subseteq E$ such that:*

- $\forall 1 \leq i, j \leq n \,.\, \neg\left(e_i \# e_j\right)$
- $\forall 1 \leq i, j \leq n \,.\, \left(\mathrm{ic}(e_i) \cup \mathrm{ac}\left(\overline{t_{i-1}}, e_i\right)\right) \subseteq \overline{t_{i-1}}$

Then $C \subseteq E$ is a trace-based configuration *of γ if there is a trace t such that $C = \overline{t}$. The set of traces of γ is denoted by $\mathrm{T}(\gamma)$ and the set of its trace-based configurations is denoted by $\mathrm{C}_{Tr}(\gamma)$.*

Semantics of GESs could be captured through posets, but this would complicate the situation. A single configuration then will have multiple posets. For instance, $m \blacktriangleright [c \rightarrow t]$ with the configuration $\{c, m, t\}$ could have two orders: either $\{c \leq t\}$ where m is independent from all other event, or $\{t \leq m\}$ where c is independent from the other events. The situation would be evem more complex when considering Dynamic-Causality ESs in the next section with posets, since not only the order of adders and targets would matter, but also adders and droppers. Therefore, and to be able to compare to the RCESs, configuration transitions were used.

Definition 3.4.3. *Let $\gamma = (E, \#, \rightarrow, \blacktriangleright)$ be a GES, and $X, Y \subseteq E$. Then $X \rightarrow_{\mathrm{g}} Y$ iff:*

- $X \subseteq Y$
- $\forall e, e' \in Y \,.\, \neg\left(e \# e'\right)$
- $\forall e \in Y \setminus X \,.\, (\mathrm{ic}(e) \cup \mathrm{ac}(X, e)) \subseteq X$
- $\forall t, m \in C' \setminus C \,.\, \forall c \in E \,.\, m \blacktriangleright [c \rightarrow t] \implies \left(c \in C \vee m \in \{c, t\}\right)$

We denote the reflexive and transitive closure of $\rightarrow_{\mathrm{g}}$ as $\rightarrow_{\mathrm{g}}^*$.

The last condition prevents the concurrent occurrence of a target and its modifier since they are not independent. One exception is when the contribution has already occurred; in that case, the modifier does not change the target's predecessors. It also captures the trivial case of self adding, i.e. when a target adds a contribution to itself or a contribution adds itself to a target.

Again we consider the reachable and finite configurations, and we consider two GESs as equally expressive if they are transition-equivalent.

Definition 3.4.4. *Let $\gamma = (E, \#, \rightarrow, \blacktriangleright)$ be a GES. The set of all configurations of γ is $\mathrm{C}(\gamma) = \left\{X \subseteq E \mid \emptyset \rightarrow_{\mathrm{g}}^* X \wedge X \text{ is finite}\right\}$*

As in SESs, both notions of configurations of GESs, the traced-based and the transition-based, coincide; and depending on the situation, the most suitable one can be used.

Lemma 3.4.5. *Let γ be a GES. Then $\mathrm{C}_{Tr}(\gamma) = \mathrm{C}(\gamma)$.*

The remainder is generally defined in a similar way to Definition 3.2.4. Furthermore, causality pairs are added to the remainder if there is an adder for them in H, unless one of them took place in H. For instance, the remainder of Figure 3.5 (b) after $H = \{a\}$ is $(\{b\}, \emptyset, \{(b,b)\}, \emptyset)$.

Definition 3.4.6. *Let $\gamma = (E, \#, \rightarrow, \blacktriangleright)$ be a GES, and let $H \in \mathrm{C}(\gamma)$ be a configuration of γ. The remainder of γ after H is $\gamma[H] = (E', \#', \rightarrow', \blacktriangleright')$, where:*

- $E' = E \setminus H$
- $\#' = \# \cap E'^2$
- $\rightarrow' = \left(\rightarrow \cap E'^2\right) \cup \left\{(c,t) \in E'^2 \mid c \in \mathrm{ac}(H,t)\right\} \cup \left\{(e,e) \in E'^2 \mid \exists e' \in H \,.\, e' \# e\right\}$
- $\blacktriangleright' = \left(\blacktriangleright \cap E'^3\right) \setminus \left\{(e,a,e) \mid \exists e' \in H \,.\, e' \# e\right\}$

It can be proved that the remainder of a GES is itself a GES. Furthermore, consistency between the original structure and its remainder can be seen through the following lemma regarding transitions.

Lemma 3.4.7. *Let γ be a GES, and $H \in \mathrm{C}(\gamma)$. Then:*

$$\forall X, Y \subseteq E \setminus H \,.\, X \in \mathrm{C}(\gamma[H]) \iff H \rightarrow_{\mathrm{g}}^{*} H \cup X.$$

3.4.1. Modeling Features of Growing Causality

Disabling as defined in EBESs or the Asymmetric ES of [8] can be modeled by $\blacktriangleright$. For example $b \rightsquigarrow a$ can be modeled by $b \blacktriangleright [a \rightarrow a]$ as depicted in Figure 3.5 (b). Analogously conflicts can be modeled by $\blacktriangleright$ through mutual disabling, as depicted in Figure 3.5 (a), and thus the conflict relation can be omitted in this ES model.

In Inhibitor ESs [7] there is a kind of disabling, where an event e can be disabled by another event d until an event out of a set X occurs. This kind of temporary disabling provides disjunction in the re-enabling that cannot be modeled in GESs but in DCESs (cf. the next section). However temporary disabling without a disjunctive re-enabling can be modeled by a GES as in Figure 3.5 (c).

Also resolvable conflicts can be modeled by a GES. For example the GES in Figure 3.5 (d) with $a \blacktriangleright [c \rightarrow b]$ and $b \blacktriangleright [c \rightarrow a]$ models a conflict between a and b that can be resolved by c. Note that this example depends on the idea that a modifier and its target cannot occur concurrently (cf. Definition 3.4.3). Note also that resolvable conflicts are a reason why families of configurations cannot be used to define the semantics of GESs or RCESs.

$$\begin{tikzcd}
(\rho_\gamma) & \{a\} \arrow[r] \arrow[dr] & \{a,b\} & \\
\emptyset \arrow[ur] \arrow[r] \arrow[dr] & \{b\} \arrow[ur] \arrow[dr] & \{a,c\} \arrow[r] & \{a,b,c\} \\
 & \{c\} \arrow[ur] \arrow[r] & \{b,c\} \arrow[ur] &
\end{tikzcd}$$

Figure 3.6.: *Conjunctive disabling through a RCES*

3.4.2. Expressiveness of Growing-Causality ESs

As shown in Figure 3.5 (b) GESs can model disabling. Nevertheless EBESs and GESs are incomparable, because GESs cannot model the disjunction in the enabling relation that EBESs inherit from BESs. On the other hand EBESs cannot model conditional causal dependencies.

Thus GESs are incomparable to BESs as well as EBESs.

Theorem 3.4.8. *GESs are incomparable to BESs and EBESs.*

GESs are also incomparable to SESs, because the adding of causes cannot be modeled by SESs. Then since BESs are incomparable to GESs, BESs are less expressive than DESs, and DESs are as expressive as SESs, we conclude that GESs and SESs are incomparable.

Theorem 3.4.9. *GESs and SESs are incomparable.*

As illustrated in Figure 3.5 (d) GESs can model resolvable conflicts. Nevertheless they are strictly less expressive than RCESs, because each GES can be translated into a transition equivalent RCES and on the other hand there exists no transition equivalent GES for the RCES $\rho_\gamma = (\{a,b,c\},\vdash)$ illustrated in Figure 3.6. It models the case, where after a and b the event c becomes impossible, i.e. it models disabling by a set instead of a single event.

Theorem 3.4.10. *GESs are strictly less expressive than RCESs.*

3.4.3. Growing Causality as Conditional Causality

From Definition 3.4.3, $a \blacktriangleright [c \rightarrow t]$ can be interpreted as c is needed for t only if a occurs. This is in fact a conditional causality, which is new anyway for event structures. In other words, $\blacktriangleright$ could be seen in a static perspective, without the sense of dynamic changes during the system run.

Based on that, we have two enabling relations: $\rightarrow$ and $\blacktriangleright$, instead of having one enabling relation $\rightarrow$ and the other one $\blacktriangleright$ holding the changes. Thus similar to [7], they can be merged into one relation $\twoheadrightarrow \subseteq \mathcal{P}(E)_1 \times E^2$ where the first parameter can be a set containing one event, or the empty set. For $(\{m\}, t, c) \in \twoheadrightarrow$ we write $c \xrightarrow{\{m\}}\!\!\!\!\twoheadrightarrow t$ which means that if the modifier m occurs, then c is needed for the target t. If the first argument is the empty set, we drop it from the notation and write $c \longrightarrow\!\!\!\!\twoheadrightarrow t$, which means that the cause c is needed anyway for the target t. We use the notation $\mathcal{P}(E)_k$ (with $k \in \mathbb{N}$) for all subsets of E with cardinality less or equal to k.

Definition 3.4.11. *A* Conditional-Causality Event Structure (CES) *is a tripple* $\chi = (E, \#, \twoheadrightarrow)$ *where:*

- *E, a set of* events
- *$\# \subseteq E^2$, an irreflexive symmetric relation (the* conflict *relation)*
- *$\twoheadrightarrow \subseteq \mathcal{P}(E)_1 \times E^2$, the* enabling relation

that satisfies the following constraint:

$$\forall e, e', e'' \in E \,.\, e'' \xrightarrow{\{e\}}\!\!\!\!\twoheadrightarrow e' \implies e'' \not\longrightarrow\!\!\!\!\twoheadrightarrow e' \tag{3.1}$$

We can define the transition relation of a CES according to the interpretation above, then for each CES $\chi = (E, \#, \twoheadrightarrow)$ we can derive a GES $\gamma = (E, \#, \rightarrow, \blacktriangleright)$ where:

- $\rightarrow = \left\{(e', e) \in E^2 \mid e' \longrightarrow\!\!\!\!\twoheadrightarrow e\right\}$
- $\blacktriangleright = \left\{(e', e'', e) \in E^3 \mid e' \xrightarrow{\{e''\}}\!\!\!\!\twoheadrightarrow e\right\}$

The other way around could be done from GESs to CESs. It can be proved that GESs and their translations into CESs are transition-equivalent, and vise versa. This is similar to the situation between SESs and DESs, i.e. GESs can be equivalent to ESs with a totally static perspective, yet still a new contribution to ESs.

3.4.4. A Special Case of Concurrency: Target-Modifier Concurrency

The last condition in Definition 3.4.3 prohibits the concurrent occurrence between a modifier and its target, in case the added cause has not occurred yet. This keeps the theory simple. But let us consider the consequences of dropping such a constraint. Let us denote the GES transition relation

$$\begin{tikzcd}
 & \{a\} & \\
\emptyset \arrow[ur] \arrow[dr] & & \{a,b\} \\
 & \{b\} \arrow[ur] &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
 & \{a\} & \\
\emptyset \arrow[ur] \arrow[dr] \arrow[rr] & & \{a,b\} \\
 & \{b\} \arrow[ur] &
\end{tikzcd}$$

(b)

Figure 3.7.: *The transition graph of Figure 3.5* (b) *w.r.t.* $\rightarrow_{\mathrm{g}}$ *in* (a), *and w.r.t.* $\rightarrow_{\mathrm{g_c}}$ *in* (b).

after dropping the mentioned condition by $\rightarrow_{\mathrm{g_c}}$, and let us consider a new ES that has the same structural definition of Definition 3.4.1, and $\rightarrow_{\mathrm{g_c}}$ as a transition relation.

The new structure would allow the occurrence of two events simultaneously as well as in one order, but not in the other order, e.g. Figure 3.7. This distinction here in the execution order is similar to the *left merge* and *communication* of Baeten et al. [5]. The authors suggested that the first step in the parallel composition of two processes could be taken either by one of the processes (a *left merge*), or by both simultaneously (a *communication*).

Consequently many results in the original GESs, for which proofs were based on the modeling capability of disabling, cannot be proved in the same way considering $\rightarrow_{\mathrm{g_c}}$, if at all. Up to the time of writing this thesis, such a case is new to ESs, since the simultaneous occurrence of two events implies their interleaving, as in RCESs which generalize many other ESs in the literature so far. This will make the resulting structure incomparable to other kinds of ESs including classical GESs, except for PESs.

Semantics of GESs with $\rightarrow_{\mathrm{g_c}}$ cannot be shown by posets, since partial orders never capture the case of concurrent events that are not independent. Consequently, traces are not able to give the semantics of GESc either, since they are a special case of orders. Extensions to posets were introduced in the literature in order to capture such cases, e.g. *stratified order structures* [42].

Stratified order structures (so-structures) capture the relation of *earlier than* which fits to our initial causality, as well as *not later than* which fits to the growing causality. An so-structure is a triple $(X, \prec, \sqsubset)$: a set, the earlier-than relation $\prec$, and the not-later-than relation $\sqsubset$, such that $\prec$ is a partial order, $\sqsubset$ is a strict pre-order, and $a \prec b \implies a \sqsubset b$.

In contrast to posets which can give semantics to the non-concurrent GES, so-structures can be used to give semantics to the concurrent GES. For instance, the configuration $C = \{c, m, t\}$ where $m \blacktriangleright [c \to t]$ could be modeled with two so-structures: $(C, \{c \prec t\}, \{c \sqsubset t\})$ and $(C, \emptyset, \{t \sqsubset m\})$ where the reflexive closure of $\prec$ is omitted for simplicity.

The case where $b \blacktriangleright [a \to c]$ and $a \blacktriangleright [b \to c]$ with the configuration $\{a, b\}$, that can be reached only by the transition $\{a, b\}$, can be modeled through a cycle of $\sqsubset$, i.e. $a \sqsubset b$ and $b \sqsubset a$. A better semantical model for such a case is the *quotient so-structures* defined by Le in [50] which focuses on synchronization steps. Anyway choosing configuration transitions makes GESc comparable to non-concurrent version of GES as well as other dynamic causality structures.

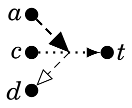
3.5. Fully Dynamic Causality

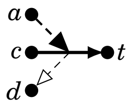
Up to now we have investigated shrinking, and growing causality separately. In this section we combine them and examine the resulting expressiveness.

Definition 3.5.1. *A* Dynamic Causality Event Structure (DCES) *is a triple $\Delta = (\pi, \triangleright, \blacktriangleright)$,expanded as $(E, \#, \to, \triangleright, \blacktriangleright)$, where $\pi = (E, \#, \to)$ is a PES, $\triangleright \subseteq E^3$ is the shrinking causality relation, and $\blacktriangleright \subseteq E^3$ is the growing causality relation such that for all $e, e', e'' \in E$:*

1. $[e \to e''] \triangleright e' \wedge \nexists m \in E \,.\, m \blacktriangleright [e \to e''] \implies e \to e''$
2. $e' \blacktriangleright [e \to e''] \wedge \nexists m \in E \,.\, [e \to e''] \triangleright m \implies \neg(e \to e'')$
3. $e' \blacktriangleright [e \to e''] \implies \neg([e \to e''] \triangleright e')$

Conditions 1 and 2 are just a generalization of the conditions in Definitions 3.3.1 and 3.4.1 respectively. If there are droppers and adders for the same causal dependency as in the examples below, we do not specify whether this dependency is contained in $\to$, because the semantics depends on the order in which the droppers and adders occur. Condition 3 prevents that a modifier adds and drops the same cause for the same target.





The order of occurrence of droppers and adders determines the causes of an event. For example after the trace ad in both examples above, t does not depend on c, whereas after da, t depends on c. Thus configurations like $\{a,d\}$ are not expressive enough to represent the state of such a system.

Therefore in a DCES a state is a pair of a configuration C and a causal state function cs, which computes the causal predecessors of an event, that are still needed. This deviates from the original concept of events suggested by Winskel [83], where causal predecessors of an occurrence of an action, i.e. an event, are fixed, and once the they change, the event is duplicated.

Definition 3.5.2. *Let* $\Delta = (E, \#, \to, \triangleright, \blacktriangleright)$ *be a DCES. The function* mc : $\mathcal{P}(E) \times E \to \mathcal{P}(E)$ *denotes the* maximal causality *that an event can have after some history* $C \subseteq E$*, and is defined as:*

$$\mathrm{mc}(C,e) = \{e' \in E \setminus C \mid e' \to e \vee \exists a \in C\,.\, a \blacktriangleright [e' \to e]\}$$

A state *of* Δ *is a pair* (C, cs) *where* $\mathrm{cs} : E \setminus C \to \mathcal{P}(E \setminus C)$ *such that* $C \subseteq E$ *and* $\mathrm{cs}(e) \subseteq \mathrm{mc}(C,e)$*. We denote* cs *as the* causality state function*, which shows for each event e that did not occur, which events are still missing to enable e. An initial state of* Δ *is* $S_0 = (\emptyset, \mathrm{cs_i})$*, where* $\mathrm{cs_i}(e) = \{e' \in E \mid e' \to e\}$.

Note that S_0 is the only state with an empty set of events; for other sets of events there can be multiple states. Accordingly, the behavior of a DCES is defined by the transition relation on its reachable states with finite configurations.

Definition 3.5.3. *Let* $\Delta = (E, \#, \to, \triangleright, \blacktriangleright)$ *be a DCES and* $C, C' \subseteq E$*. Let* $D = C' \setminus C$*. Then* $(C, \mathrm{cs}) \to_{\mathrm{d}} (C', \mathrm{cs}')$ *iff:*

1. $C \subseteq C'$
2. $\forall e, e' \in C'\,.\, \neg(e \# e')$
3. $\forall e \in D\,.\, \mathrm{cs}(e) = \emptyset$
4. $\forall e, e' \in E \setminus C'\,.\, e' \in \mathrm{cs}(e) \implies \big(e' \notin \mathrm{cs}'(e) \iff [e' \to e]\triangleright \cap D \neq \emptyset\big)$
5. $\forall e, e' \in E \setminus C'\,.\, e' \notin \mathrm{cs}(e) \implies \big(e' \in \mathrm{cs}'(e) \iff \blacktriangleright[e' \to e] \cap D \neq \emptyset\big)$
6. $\forall e, e' \in E \setminus C\,.\, \big([e' \to e]\triangleright \cap D = \emptyset\big) \vee \big(\blacktriangleright[e' \to e] \cap D = \emptyset\big)$
7. $\forall t, m \in D\,.\, \forall c \in E\,.\, m \blacktriangleright [c \to t] \implies \big(c \in C \vee m \in \{c, t\}\big)$

We denote the reflexive and transitive closure of $\to_{\mathrm{d}}$ *as* $\to_{\mathrm{d}}^*$.

Condition 1 insures the accumulation of events. Condition 2 insures conflict freeness. Condition 3 insures that only events which are enabled after C can take place in C'. Condition 4 insures that, a cause disappears iff there is a dropper of it. The same is ensured by Condition 5 for appearing causes and adders. To keep the theory simple, Condition 6 avoids race conditions; it forbids the occurrence of an adder and a dropper of the same causal dependency within one transition. Condition 7 ensures that DCESs coincide with GESs.

Definition 3.5.4. *Let Δ be a DCES. The set of (reachable) states of Δ is defined as* $\mathrm{S}(\Delta) = \{(X, \mathrm{cs}_X) \mid S_0 \rightarrow_{\mathrm{d}}^{*} (X, \mathrm{cs}_X) \wedge X$ *is finite* $\}$. *Two DCESs* $\Delta = (E, \#, \rightarrow, \rhd, \blacktriangleright)$ *and* $\Delta' = \left(E', \#', \rightarrow', \rhd', \blacktriangleright'\right)$ *are* state transition equivalent, *denoted by* $\Delta \simeq_{\mathrm{s}} \Delta'$, *iff* $E = E'$ *and* $\rightarrow_{\mathrm{d}} \cap \left(\mathrm{S}(\Delta)\right)^2 = \rightarrow_{\mathrm{d}}' \cap \left(\mathrm{S}(\Delta')\right)^2$.

The (reachable) configurations are then the configurations of (reachable) states.

Definition 3.5.5. *Let* $\Delta = (E, \#, \rightarrow, \rhd, \blacktriangleright)$ *be a DCES. The set of its (reachable) configurations is* $\mathrm{C}(\Delta) = \pi_1\left(\mathrm{S}(\Delta)\right)$*; the projection on the first component of the states. The configuration-transition relation* $\rightarrow_{\mathrm{c}} \subseteq [\mathcal{P}(E)]^2$ *is defined as* $C \rightarrow_{\mathrm{c}} C'$ *iff there are two states* S, S' *of* Δ *such that* $C = \pi_1(S)$, $C' = \pi_1(S')$ *and* $S \rightarrow_{\mathrm{d}} S'$. *We denote the reflexive and transitive closure of* $\rightarrow_{\mathrm{c}}$ *as* $\rightarrow_{\mathrm{c}}^{*}$.

Let us consider the following DCES:
$\Delta = \left(\{a, b, c, d\}\ ,\ \emptyset\ ,\ \emptyset\ ,\ \{[c \rightarrow d] \rhd b\}, \{a \blacktriangleright [c \rightarrow d]\}\right)$. If we consider only the configurations of Δ saying there is a transition $C \rightarrow_{\mathrm{c}} C'$ whenever $(C, \mathrm{cs}) \rightarrow_{\mathrm{d}} (C', \mathrm{cs}')$, then the transition $\{a, b\} \rightarrow_{\mathrm{c}} \{a, b, d\}$ always exists regardless of the order of a, b. Whereas by considering $\rightarrow_{\mathrm{d}}$, a transition from $\{a, b\}$ to $\{a, b, d\}$ in Δ will exist only if a precedes b. Therefore we consider the more discriminating equivalence $\simeq_{\mathrm{s}}$—instead of $\simeq_{\mathrm{t}}$—to compare DCESs and to compare with DCESs.

Traces of a DCES can be defined then based on transitions between reachable configurations C, C' such that $C' \setminus C$ contains only one event.

Definition 3.5.6. *Let* $\Delta = (E, \#, \rightarrow, \rhd, \blacktriangleright)$ *be a DCES with a configuration-transition relation* $\rightarrow_{\mathrm{c}}$. *A sequence of events* $e_1, e_2, \ldots, e_n \in E$ *is a* trace *of* Δ *iff the transitions* $\emptyset \rightarrow_{\mathrm{c}} \{e_1\} \rightarrow_{\mathrm{c}} \{e_1, e_2\} \rightarrow_{\mathrm{c}} \ldots \rightarrow_{\mathrm{c}} \{e_1, e_2, \ldots, e_n\}$ *exist.*

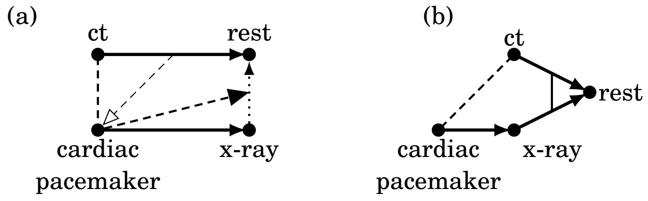


Figure 3.8.: *A DCES modeling in* (a) *of a change in a medical workflow to perform an X-Ray for a cardiac-pacemaker patient instead of a tomography (cf. Section 3.1), and a trace-equivalent DES modeling in* (b) *that does not distinguish between the default and the exceptional paths.*

3.5.1. Embedding Growing and Shrinking Causality Event Structures

To compare DCESs to other ESs we define the Single State Dynamic Causality ESs (SSDCs) as a subclass of DCESs.

Definition 3.5.7. *Let SSDC be a subclass of DCESs such that ϱ is a SSDC iff $\forall e, e' \in E \,.\, \nexists a, d \in E \,.\, a \blacktriangleright [e' \to e] \triangleright d$.*

Since there are no adders and droppers for the same causal dependency, the order of modifiers does not matter and thus there are no two different states sharing the same configuration, i.e. each configuration represents a state. Thus it is enough for SSDC to consider transition equivalence with respect to configurations, i.e. $\simeq_{\mathrm{t}}$.

Lemma 3.5.8. *Let ϱ be a SSDC. Then for the causal-state function cs of any state $(C, cs) \in \mathrm{S}(\varrho)$ it holds $cs(e) = \big(\mathrm{ic}(e) \cup \mathrm{ac}(C,e)\big) \setminus \big(\mathrm{dc}(C,e) \cup C\big)$.*

Additionally, in a SSDC we can see that Conditions 4 and 5 hold automatically whenever $C \subseteq C'$, as shown by the following lemma.

Lemma 3.5.9. *Let ρ be a SSDC and let (C, cs) and (C', cs') be two states of ρ with $C \subseteq C'$, then Conditions 4 and 5 of $\to_{\mathrm{d}}$ hold for those two states.*

Let us look at another property SSDC holds regarding sub-transitions.

Lemma 3.5.10. *Let ρ be a SSDC and $(X, \mathrm{cs}_X) \to_{\mathrm{d}} (Y, \mathrm{cs}_Y)$ a transition in ρ. Then for all X', Y' with $X \subseteq X' \subseteq Y' \subseteq Y$, there is a transition $(X', \mathrm{cs}'_X) \to_{\mathrm{d}} (Y', \mathrm{cs}'_Y)$ in ρ, where:*

- $\mathrm{cs}_{X'}(e) = \Big(\big(\mathrm{ic}(e) \cup \mathrm{ac}(X', e)\big) \setminus \big(\mathrm{dc}(X', e) \cup X'\big)\Big)$
- $\mathrm{cs}_{Y'}(e) = \Big(\big(\mathrm{ic}(e) \cup \mathrm{ac}(Y', e)\big) \setminus \big(\mathrm{dc}(Y', e) \cup Y'\big)\Big)$.

Based on that, the remainder of a SSDC Δ could be taken simply after a reachable configuration H, and is denoted as $\Delta[H]$, where the causality state function of H is derived according to Lemma 3.5.8.

Definition 3.5.11. *Let* $\Delta = (E, \#, \to, \rhd, \blacktriangleright)$ *be a SSDC, and let* $H \in \mathrm{C}(\Delta)$ *be a reachable configuration of* Δ. *Let* $S \in \mathrm{S}(\Delta)$ *such that* $\pi_1(S) = H$, *and let* $\mathrm{cs}_H = \pi_2(S)$. *The remainder of* Δ *after* H *is* $\Delta[H] = \big(E', \#', \to', \rhd', \blacktriangleright'\big)$, *where:*
- $E' = E \setminus H$
- $\#' = \# \cap E'^2$
- $\to' = \big\{(e', e) \in E'^2 \mid e' \in \mathrm{cs}_H(e)\big\} \bigcup \big\{(e, e) \in E'^2 \mid \exists e' \in H \,.\, e' \# e\big\}$
- $\rhd' = \big(\rhd \cap E'^3\big) \setminus \big\{(e, d, e) \mid \exists e' \in H \,.\, e' \# e\big\}$
- $\blacktriangleright' = \big(\blacktriangleright \cap E'^3\big) \setminus \big\{(e, a, e) \mid \exists e' \in H \,.\, e' \# e\big\}$

The definition of $\rhd'$ and $\blacktriangleright'$ is similar to Definition 3.3.6 and 3.4.6 resp. It can be proved that the $\Delta[H]$ is a DCES. Furthermore, It can be seen that it is a SSDC since no new causality adding or dropping was added.

Lemma 3.5.12. *Let* σ *be a SSDC,* $H \in \mathrm{C}(\sigma)$. *Then:*

$$\forall X \subseteq E \setminus H \,.\, X \in \mathrm{C}(\sigma[H]) \iff H \to_{\mathrm{c}}^{*} H \cup X.$$

To embed a SES (or GES) into a DCES it suffices to choose $\blacktriangleright = \emptyset$ (or $\rhd = \emptyset$).

Definition 3.5.13. *Let* $\sigma = (E, \#, \to, \rhd)$ *be a SES. Then its embedding is* $\mathrm{i}(\sigma) = (E, \#, \to, \rhd, \emptyset)$. *Similarly let* $\gamma = (E, \#, \to, \blacktriangleright)$ *be a GES. Then its embedding is* $\mathrm{i}\big(\gamma\big) = (E, \#, \to, \emptyset, \blacktriangleright)$.

Both embeddings are SSDCs. Furthermore for each embedding the causal state coincides with a condition on the initial, added, and dropped causes, that are enforced in the transition relations of SESs and GESs.

Lemma 3.5.14. *Let* σ *be a SES and* $\mathrm{i}(\sigma)$ *its embedding. Then we have for each state* (C, cs) *of* $\mathrm{i}(\sigma)$, $\mathrm{cs}(e) = \mathrm{ic}(e) \setminus \big(\mathrm{dc}(C, e) \cup C\big)$.

Lemma 3.5.15. *Let* γ *be a GES and* $\mathrm{i}\big(\gamma\big)$ *its embedding. Then we have for each state* (C, cs) *of* $\mathrm{i}\big(\gamma\big)$, $\mathrm{cs}(e) = \big(\mathrm{ic}(e) \cup \mathrm{ac}(C, e)\big) \setminus C$.

SESs (resp. GESs) and their embeddings are transition equivalent.

Lemma 3.5.16. *Let* μ *be a GES or SES, then we have* $\mathrm{i}\big(\mu\big) \simeq_{\mathrm{t}} \mu$.

3.5.2. Embedding Extended Bundle Structures

To compare with EBESs, we use the disabling of GESs, and the disjunctive causality of SESs. To achieve that, we define a sub-class of DCESs, where posets could be defined and used for semantics.

Definition 3.5.17. *Let EBDC denotes a subclass of SSDC with the additional requirements:*

1. $\forall c, m, t \in E \, . \, m \blacktriangleright [c \to t] \implies c = t$
2. $\forall c, m_1, \ldots, m_n, t \in E \, . \, [c \to t] \rhd m_1, \ldots, m_n \implies$
$$\forall e_1, e_2 \in \{c, m_1, \ldots, m_n\} \, . \, (e_1 \neq e_2 \implies e_1 \# e_2)$$

The first condition translates disabling into $\blacktriangleright$ and ensures that disabled events cannot be enabled again. The second condition reflects causal unambiguity by $\rhd$ such that either the initial cause or one of its droppers can happen.

Definition 3.5.18. *Let ϑ be a EBDC and $C \in C(\vartheta)$, then we define the precedence relation $<_C \subseteq C \times C$ as $e <_C e'$ iff $e \to e' \vee e' \blacktriangleright [e \to e] \vee \exists c \in E \, . \, [c \to e'] \rhd e$. Let $\leq_C$ be the reflexive and transitive closure of $<_C$.*

The relation $<_C$ indeed represents a precedence relation as the following lemma states.

Lemma 3.5.19. *Let ϑ be a EBDC, $C \in C(\vartheta)$, and let $e, e' \in C \, . \, e <_C e'$. Let also $(C_0, cs_0) \to_{\mathrm{d}} \ldots \to_{\mathrm{d}} (C_n, cs_n)$ with $C_0 = \emptyset$ and $C_n = C$ be the transition sequence of C, then $\exists C_i \in \{C_0, \ldots, C_n\} \, . \, e \in C_i \wedge e' \notin C_i$.*

Then the reflexive and transitive closure of $<_C$ is a partial order.

Lemma 3.5.20. *$\leq_C$ is a partial order over C.*

Let $\mathrm{P}(\vartheta) = \{(C, \leq_C) \mid C \in \mathrm{C}(\vartheta)\}$ denotes the set of posets of the EBDC ϑ. We show that the transitions of a EBDC ϑ can be extracted from its posets.

Theorem 3.5.21. *Let ϑ be an EBDC and $(C, \mathrm{cs}), (C', \mathrm{cs}') \in \mathrm{S}(\vartheta)$ with $C \subseteq C'$. Then $\left(\forall e, e' \in C' \, . \, e \neq e' \wedge e' \leq_{C'} e \implies e' \in C\right) \iff (C, \mathrm{cs}) \to_{\mathrm{d}} (C' \mathrm{cs}')$.*

The following defines a translation from an EBESs into an EBDC. Figure 3.9 provides an example, where conflicts with impossible events are dropped for simplicity.

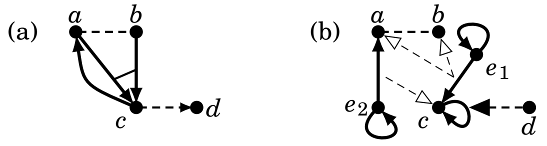


Figure 3.9.: *An EBES and its poset-equivalent DCES.*

Definition 3.5.22. *Let* $\xi = (E, \rightsquigarrow, \mapsto, l)$ *be an EBES. Then:*
$\mathrm{dces}(\xi) = (E', \#', \rightarrow, \triangleright, \blacktriangleright)$ *such that:*

1. $E', \rightarrow, \triangleright$ *are defined as in 3.3.11*
2. $\#' = \{(e,e') \mid e \rightsquigarrow e' \land e' \rightsquigarrow e\} \cup \{(x_i, x) \mid x \in X_i\}$
3. $\blacktriangleright = \{(e,e',e) \in E^3 \mid e \rightsquigarrow e' \land \neg(e' \rightsquigarrow e)\}$.

The translation shows that disabling uses self-loops of target events, while droppers use auxiliary impossible events, not to intervene with the disabling.

Lemma 3.5.23. *Let* ξ *be an EBES. Then* $\mathrm{dces}(\xi)$ *is an EBDC.*

Furthermore, the translation preserves posets.

Lemma 3.5.24. *For each EBES* ξ *there is a DCES* $\mathrm{dces}(\xi)$ *such that:*

$$\mathrm{P}(\xi) = \mathrm{P}(\mathrm{dces}(\xi)).$$

3.5.3. Expressiveness of Dynamic-Causality Event Structures

Since EBESs cannot model the disjunctive causality—without a conflict—of DCESs that is inherited from SESs, the following result holds.

Theorem 3.5.25. *DCESs are strictly more expressive than EBESs.*

To compare with RCESs, we use the counterexample ρ_γ of Figure 3.6 to show a RCES with no transition-equivalent DCES. Moreover RCESs cannot distinguish between different causality states of one configuration. Consequently DCESs and RCESs are incomparable.

Theorem 3.5.26. *DCESs and RCESs are incomparable.*

By construction, DCESs are at least as expressive as GESs and SESs. Furthermore DCESs are incomparable to RCESs which are in turn strictly more expressive than GESs and SESs. Thus DCESs are strictly more expressive then GESs and SESs.

Theorem 3.5.27. *DCESs are strictly more expressive than GESs and SESs.*

3.6. Evaluation

We study the idea that causality may change during system runs in event structures. For this, we enhance a simple type of ESs, namely the PES, by means of additional relations capturing the changes in events' dependencies, triggered by the occurrence of other events.

First, in Section 3.3, we limit our concern to the case where dependencies can only be dropped. We call the resulting event structure Shrinking Causality ES (SES). In that section, we show that the exhibited dynamic causality can be expressed through a completely static perspective, by proving equivalence between SESs and DESs.

Later on, in Section 3.4, we study the complementary style where dependencies can be added to events, resulting in Growing Causality ES (GES). We show that the growing causality can model both permanent and temporary disabling. Besides, it can be used to resolve conflicts and, furthermore, to force conflicts. Unlike the SESs, the GESs are not directly comparable to other types of ESs from the literature, except for PESs; one reason is that they provide a conjunctive style of causality, another is their ability to express conditioning in causality.

Finally, in Section 3.5, we combine both approaches of dynamicity with a new type of event structures, which we call Dynamic Causality ES (DCES). Therein a dependency can be both added and dropped. For this new type of ESs the following two (possibly surprising) facts can be observed: (1) There are types of ESs that are incomparable to both SESs and GESs, but that are comparable to (here: strictly less expressive than) DCESs, i.e. the combination of SESs and GESs; one such type is EBESs. (2) Though SESs and GESs are strictly less expressive than RCESs, their combination—the newly defined DCESs—is incomparable to RCESs, or any other static-causality type of ESs considered in this thesis.

Application in the Use Case: If changes to workflows are to be repeated ofen or learned from, DCESs can be used to formally show the changes caused by some event. For instance, in the cardiac-patient use case, the changes caused by the heart-failure, namely implanting a blood pump, can be detected formally so that this case is generalized for similar future cases [66]. The resulting model would look like in Figure 3.10. In Section 4.6 we will provide a mechanized way of learning from adaptation and inferring changes to be used again. The same applies for cold-leg in Figure 3.11.

Exceptional v.s. Regular Paths: With respect to Exceptional and Default paths of [61], dynamic causality can represent the exceptional path of execution, while initial causality would refer to the default one. For instance, the dependency between BloodPump and WakeUp represents the exceptional path, while the other initial dependencies, like WarmUp → WakeUp, would represent the default one.

Membership Events as Modifiers: The other example, taken from the cardiac-patient use case, shows that *After she joined the coalition, the surgeon changed the plan and decided to perform an angiography, to detect the reason of the cold leg*. In case we generalize this case, the result could be be modelled by a DCES as in Figure 3.12. One main difference is that the modifier that causes the changes is the join of a new member.

Composing Complex Changes: Patterns for changing a workflow were suggested in [80] to avoid errors. For instance, given a flow between two tasks $b \rightarrow c$, the *SERIAL INSERT* pattern denotes the insertion of a new

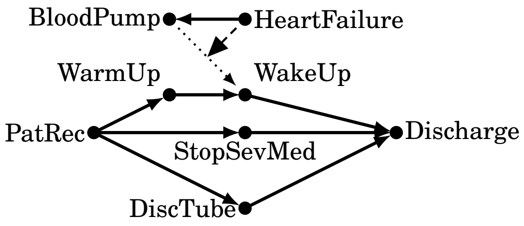


Figure 3.10.: *A GES modeling capturing the case that a heart failure would require implanting a blood pump.*

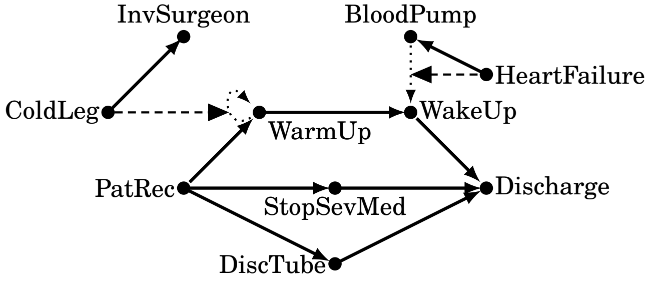


Figure 3.11.: *A GES modeling of the use case, capturing the conclusions that a heart failure demands the implantation of a blood pump, and the cold-leg problem disables the warm-up process.*

task a between b and c such that $a \to b \to c$. Such changes can be composed out by primitive changes of DCESs. The *SERIAL INSERT* pattern can be implemented by first enabling a through dropping its causality cycle, and dropping $b \to c$, and then adding $b \to a$ and $a \to c$. The *SKIP* pattern corresponds to disabling an event, and isolating it by dropping causality links with its successors, and then connecting its successors with its predecessors by growing causality.

Limitations: Since changes are foreseen in DCESs, there is no real adding or dropping of events. Rather, the events to be added to a workflow already exist in the structure, but are initially disabled. Adding these events would mean enabling and pushing them in the right order. The same applies for dropping events. For instance, HeartFailure does not add the event of BloodPump implanting in Figure 3.10. Rather it enables this event through a causal dependency. Another alternative for enabling is through dropping a causality cycle as in Figure 3.12, which has the advantage that it still shows the enabling as an Exceptional path. By such an enabling, no undesired runs can be derived from such a model since new events cannot take place without their adders. Dropping events can be simulated through disabling such events, which cannot appear in reachable configurations anymore. Adding fresh events that are unforeseen, as well as unforeseen dropping of events is covered in the next chapter of evolution.

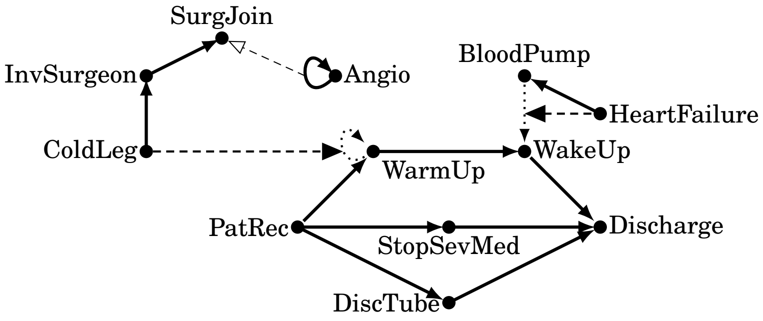


Figure 3.12.: *A DCES modeling for the join of the surgeon as a new member who extends the plan by an angiography to be performed.*

Besides, the connection between two structures: the structures before adding fresh events (or dropping existing events) and the structure after adding (or dropping) these events—i.e. how to move from one structure to the other—and which changes are allowed on the first structure are still missing. Furthermore, triggers of changes, i.e. modifiers in our approach, are always included as part of the process itself. This might not always be the case, since changes could be triggered by external events such as ones from the environment [80]. These limitations are overcome by the notion of evolution in the next chapter, where there is the option to include triggers or exclude them. Additionally, criteria on when a change is accepted and when not are presented in the next chapter.

4. Unforeseen Changes in Workflows: Evolving Event Structures

4.1. Introduction

The dynamicity shown in the different variants of Dynamic-Causality ESs in Chapter 3 corresponds to pre-planned deviations that are captured at build time. On the other hand, other changes can be unforeseen and need to be handled dynamically during runtime [61]. This might be due to the Distributed Problem Solving [34] nature of DCs, where agents contribute to solving the problem, and thus change the plan. Additionally, this can be seen in ad-hoc dynamic coalitions, that might emerge as a response to an acute need [68], where no clear plan exists; rather, the plan evolves.

Observations made in the healthcare sector emphasize that a perfect plan that takes into consideration all execution paths and possibilities never exists. Rather the plan evolves until it reaches a final state where objectives are satisfied. For instance, in the *Cardiac Patient* use case, before the heart failure was found, the plan was as captured by δ_1 in Figure 4.1 and as was shown beforehand in Section 2.10. After the patient was transferred to the Care Unit, i.e. after the system run $\{\mathrm{PatRec}\} \in C(\delta_1)$, the heart failure was found, and thus the plan evolved to the one illustrated by Figure 4.2. Up to now, DCESs do not support such dynamicity, which can be seen in terms of changes in the model.

In process-oriented systems, where running workflows belong to a schema, changes can be seen in two levels [80]. The first is schema (or process type) level. Changes of this kind usually happen due to changes in the business environment such as changes in regulations. Such changes affect current and future instances. The other kind is running-instance level. Changes here are usually ad-hoc ones that occur as a response to excep-

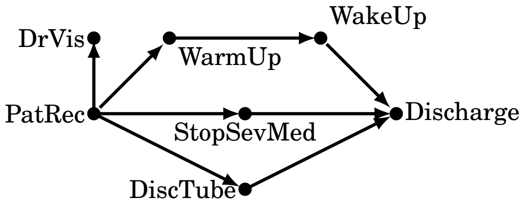


Figure 4.1.: *The default plan for the cardiac patient after she was admitted to the Cardiac Intensive Care Unit.*

tions, e.g. the cold-leg problem, and are instance-specific. Here we abstract from the type of a process or a workflow, and address dynamic changes in general, that occur during the runtime of a workflow. Furthermore, both levels of changes mentioned beforehand finally affect workflows of running instances. However, in Appendix C we address evolution in schema-instance approach, and provide a framework that supports instance migration based on the concepts and formalisms presented in this chapter.

We use the term *trigger* of an evolution, to denote the event that caused the evolution, and to abstract from its nature, i.e. whether it is a change in regulations at the schema level, or an exception [1] that is instance-specific. Besides, we call such an evolution known in the literature, which is a pair or a transition from one process to another, an *evolution step*. We reserve the term *evolution* to be used with goal satisfaction in Section 4.6, to denote that it occurs generally after several evolution steps.

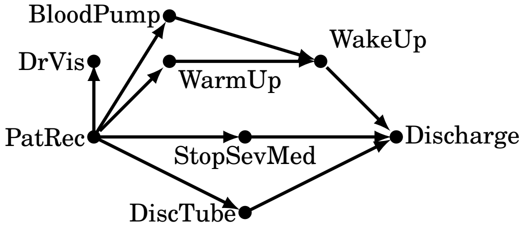


Figure 4.2.: *Evolution of the Cardiac-Patient workflow after the discovery of the heart failure problem.*

Overview In this chapter, we investigate how to model such evolution steps formally, taking into consideration the history of a workflow, as well as the evolution trigger. We present different approaches, in which the trigger of an evolution step can be either external to the evolving structure and thus untraceable, or internalized into the structures and traceable. Besides, we show the relation between the presented approaches. Later on, we define goal-orientation on evolving structures, such that an evolution is guided by the goals of coalitions. By the end of this chapter, we will be able to faithfully model the use case of the cardiac patient. Furthermore, we establish a link between evolution and dynamic causality ESs of the last chapter, to enable extracting changes and learning from an evolution step. Accordingly, we connect all kinds of dynamicity offered in this work regarding foreseen and unforeseen changes. The evolution formalisms presented here in addition to dynamic causality ESs, with the link in between, form a framework for modeling different types of changes in DCs workflows.

4.2. Correctness of Evolution w.r.t. the History of a Workflow

Since the configuration {PatRec} preceded the evolution step, we call it the *history*, represent it by H, and we represent different histories in successive evolution steps by H_1, H_2, . . . etc. History representation could be more expressive, and hold more information, e.g. being a trace or a poset, showing the order in which events took place. For the time being, we focus on configurations, while we discuss the use of other forms in Figure 4.9.

Since events of a given history will have happened when an evolution step occurs, the new workflow or structure has to include such events as part of the model. In other words, Figure 4.2 cannot exclude the event PatRec. Furthermore, the history which is a system run of the old workflow must be a system run of the new structure. Otherwise, the new workflow will not be able to reflect what happened beforehand. To clarify this, consider the example illustrated in Figure 4.3 where the cold-leg problem was found after the blood pump was implanted, and right after the Dr's visit, i.e. after the history $H_2 = H_1 \cup \{\text{BloodPump}, \text{DrVis}\}$ where $H_1 = \{\text{PatRec}\}$. The cold-leg disabled the warming-up process, and led to changes in the plan such that a surgeon was needed in the DC.

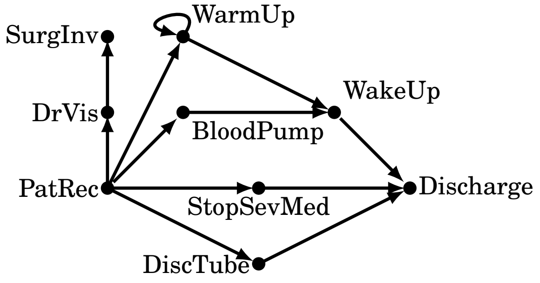


Figure 4.3.: *Evolution of the Cardiac Patient workflow after the discovery of the cold-leg problem.*

Figure 4.3, for example, cannot include the case that PatRec is in conflict with DrVis. This would contradict then with what happened beforehand, since H_2 would not be a configuration of the structure in Figure 4.3. Moreover, in the case that histories are considered to be traces, Figure 4.3 cannot include the case that DrVis $\rightarrow$ PatRec, since the sequence PatRec, BloodPump, DrVis would not be a trace in Figure 4.3.

On the other hand, evolution might lead to dropping events that were part of the old structure, as long as such events do not belong to the history. Changes in regulations, for instance, might force re-ordering the events. For example, Doctors might decide that the blood pump implanting must be done before the first visit by morning, i.e. BloodPump $\rightarrow$ DrVis. This does not contradict with the history captured by the trace PatRec, BloodPump, DrVis since it would still a trace w.r.t. the new order. Such kind of reordering is known as *Down-Sizing* [35].

History Preservation in Dynamic Coalitions and Event Structures: The concept that H should be a system run in the second workflow is called *history preserving*. Certain DCs, such as ad-hoc DCs [68], might have no schema defined in advance for their workflows. Since we are interested in such kind of DCs, history preserving helps building a model in an incremental way, i.e. step-by-step. This in turn yields a model that is able to faithfully represent the whole system run of the DC, from the first event that took place in the DC until the very end. Besides, in models where actions can be repeated, e.g. *DCR Graphs* [55], actions can be changed in an evolution step although they might have occurred [71] since such changes still affect future occurrences. This is not the case in ESs, where events cannot be repeated. Accordingly, changing future occurrences in ESs would mean changing different events, which calls for preserving the history when the model is to be built incrementally.

History preservation is one main criterion that should hold when a workflow changes dynamically [61, 65, 71]. It leads to *Instance Migration* in business processes [80], which denotes the fact a running instance of a given schema fits into the new schema after evolution. Assume the new schema is represented by an ES δ_2, then the result of migration is that the instance fits into the new schema δ_2 with the state H. Appendix C addresses instance migration with schema evolution based on history preserving. The future of the instance or the running workflow is then given by the remainder $\delta_2[H]$ of δ_2 after H. Constraints on the future of the running workflow w.r.t. to the new definition will be in the form of goal satisfaction.

Correctness criteria of dynamic changes in the literature can be classified into two types [65]. The first is based on system runs, while the second is based on graph equivalence. The second depends on comparing the old to the new structure or model of the running instance in order to detect whether the running instance can migrate or not. On the other hand, the first depends on the system run or history of the instance. Here, we apply constraints of the first type since we consider different kinds of ESs where two ESs, e.g. a Bundle ES and a Stable ES, can be different structure-wise but equivalent w.r.t. configurations. The second type would be more appropriate when considering one specific kind of ESs e.g. PESs.

Changes in the literature are meant to be the elementary modifications that could be done against workflows [80]. Examples of such elementary modifications are adding a new task, dropping an existing task, reordering tasks, etc. This implies that the correctness criteria would be applied at the level of such changes. For example, our concept of history preservation will imply the following criteria on the elementary-modification level: no new events can be added as a predecessor of an event $e \in H$ [60, 80], no events can be dropped out of H [60], and so on. Such small changes are error-prone [65] and might not happen alone, but along with other small changes. For instance, reordering two tasks implies dropping the old flow and adding a new one in the opposite direction. This drove others like [80] to define change patterns to avoid errors. Therefore, we do not consider changes, rather the evolution of a workflow caused by the changes and represented by the transition from the old one to the new one, and then our criteria are checked against the new structure. The same is applied by others like [35] using *Flow Nets*, where criteria are defined for the new net resulted from applying a given change to the old one.

We represent an evolution step of δ_1, that takes place just after (at) the history $H \in \mathrm{C}(\delta_1)$ took place and that is triggered by v as $\delta_1 \xrightarrow[H]{v} \delta_2$. It implies history preserving, i.e. $H \in \mathrm{C}(\delta_2)$. For instance, $\delta_1 \xrightarrow[\{\mathrm{PatRec}\}]{\mathrm{HeartFailure}} \delta_2$ denotes the evolution step, triggered by the discovery of the heart-failure, from δ_1 in Figure 4.1 to δ_2 in Figure 4.2. Note that this representation does not capture any information about the modeling approach to evolution, which is illustrated next.

4.3. Various Modeling Approaches to Evolution, and their Relation

Indeed, there are different approaches to model the evolution step that took place after the cold-leg problem. The first approach is illustrated in Figure 4.3 which does not include the trigger as part of the new structure. In other words, it does not track the event that triggered the change, nor clearly capture the changes caused by the trigger. Rather it captures the effects of the trigger, i.e. the warming-up is disabled. The other way is to include the trigger, namely ColdLeg, in the new structure such that the changes it brings to the structure are clearly modeled. For instance, ColdLeg disables WarmUp. This is illustrated in Figure 4.4. The same options apply for heart failure and Figure 4.2.

In the case of trigger inclusion, the workflow will continue running after the history $H'_2 = H_2 \cup \{\mathrm{ColdLeg}\}$. From the remainder point of view, or the future of the workflow after such a change took place, this is equivalent to the old approach in terms of configuration transitions. In other words, let δ_3, δ'_3 denote to the strcutures in Figure 4.3 and Figure 4.4 respectively, then $\delta_3[H_2] \simeq_{\mathrm{t}} \delta'_3[H'_2]$. The main difference between both is that while the first approach focuses on the consequences of the evolution, the second focuses on modeling the changes themselves. Besides, the trigger is traceable in the second approach while it is not in the first. The second ap-

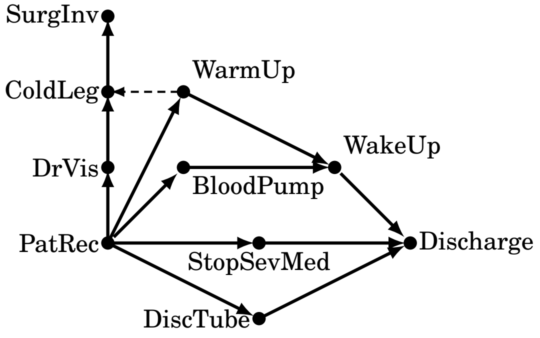


Figure 4.4.: *Evolution of the Cardiac Patient workflow after the discovery of the cold-leg problem as an EBES, capturing the changes caused by the trigger.*

proach could be seen as a transition not only to a new workflow definition, but also to a new system run, i.e. $(\delta_2, H_2), (\delta'_3, H'_2)$. While the first could be seen as a transition to a new workflow definition with the same system run, i.e. $(\delta_2, H_2), (\delta_3, H_2)$. In Section 4.5 we will define a formalization of evolution steps general enough to capture both cases.

Including the trigger into the model is not trivial. Rather it should be modeled in such a way the trigger causes all the changes, assumed to take place, to the original model itself. For instance the ColdLeg event should disable the WarmUp event if to be included; that is what is captured by Figure 4.4. Whereas by assuming the opposite, the remainder of the structure will not be equivalent then, in anyway, to the structure in Figure 4.3.

Indeed it is possible that a model includes the trigger, without completely capturing the changes brought by this trigger. For example, assume $\delta_1 = (\{a,b,c\}, \emptyset, \emptyset)$ to be a PES, that evolved to $\delta_2 = (\{a\}, \emptyset, \emptyset)$ after $H = \{a\}$ by the trigger v. In other words, b, c are dropped after the evolution. The EBES $\delta'_2 = (\{a,b,v\}, \{b \rightsquigarrow v\}, \{a \to v\})$ captures the evolution, including the trigger v, but does not capture the changes brought to δ_1. Rather, it shows the changes brought to b by disabling it, but not to c. Yet, $\delta'_2[\{a,v\}] \simeq_{\mathrm{t}} \delta_2[a]$. This proves that reflecting changes is finer than including the trigger.

Definition 4.3.1. *Let* $\delta_1 \xrightarrow[H]{v} \delta_2$ *be an evolution step. We call* δ_2*:*

1. Consequence-Reflective *iff:* $v \notin E_2$ *where* E_2 *is the event set of* δ_2*.*

2. Trigger-Inclusive *iff:* $\{v\} \in C(\delta_2[H])$.

3. Change-Reflective *iff:*
   - *δ_2 is Trigger-Inclusive, and*
   - $C(\delta_1) = C(\delta_2) \cap \mathcal{P}(E_2 \setminus \{v\})$

Definition 4.3.2. *Let $\delta_1, \delta_2, \delta_2'$ be three ESs, such that $\delta_1 \xrightarrow[H]{v} \delta_2$, $\delta_1 \xrightarrow[H]{v} \delta_2'$, where δ_2 is Consequence-Reflective, and δ_2' is Trigger-Inclusive. Then δ_2 and δ_2' are said to* coincide *iff* $\delta_2[H] \simeq_t \delta_2'[H \cup \{v\}]$.

4.4. Inferring Changes from Evolution – Process Learning

Consider an evolution step $s = \left(\delta_1 \xrightarrow[H]{v} \delta_2\right)$ where δ_2 is effect-based, like the one with the cold-leg problem in Figure 4.3. Changes to the workflow in such an evolution step can be inferred systematically. In the case where there is a process type, i.e. a schema, and instances of that type, change inference might help generalizing instance-specific changes that might be often repeated, to schema level, leading to process learning [66].

In our case, change inference is done by constructing a structure (let us denote it as Δ) that runs like δ_1 originally, until the trigger—which is an internal event—takes place then Δ will run according to δ_2. In other words, Δ would be Change-Reflective.

To show the concept, let us focus on PESs of Definition 3.2.1 as evolving structures. Then the changes will be shown in terms of a Dynamic Causality ES that is presented in the last chapter. Since DCESs cannot model

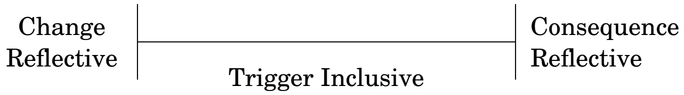


Figure 4.5.: *The two approaches of Change-Reflective and Consequence-Reflective modeling, and the grey range of Trigger-Inclusive modeling approach in-between.*

growing conflict, let us assume that given two PESs $\delta_1 = (E_1, \#_1, \rightarrow_1)$ and $\delta_2 = (E_2, \#_2, \rightarrow_2)$, the following constraint holds:

$$\#_2 \cap E_1^2 \subseteq \#_1 \tag{4.1}$$

Informally speaking, E, the set of events of Δ, will contain events of both E_1 and E_2, in addition to v. If Δ would behave initially like δ_1, $\rightarrow$ should include $\rightarrow_1$ but not $\rightarrow_2$ which will be added through growing causality. Additionally, the events of E_2 should not be enabled before the change v takes place unless they are old events, thus $\rightarrow$ should capture that v is a predecessor of events of $E_2 \setminus E_1$. On the other hand # should include conflicts that remain after the change, while resolved ones are to be modelled through growing causality $\blacktriangleright$.

If Δ is to simulate the transition (δ_1, δ_2), v must not only enable all new events $(E_2 \setminus E_1)$ through $\rightarrow$, but also disable all dropped events $(E_1 \setminus E_2)$ through $\blacktriangleright$. Besides, v should add all new causality $(\rightarrow_2 \setminus \rightarrow_1)$, and drop all dropped causality through $\triangleright$ including causality with dropped events. Furthermore, v should resolve all conflicts in $\#_2 \setminus \#_1$ as well as conflicts with dropped events. Δ is obtained through the internalization function inter as follows.

Definition 4.4.1. *Let* $s = \left(\delta_1 \xrightarrow[H]{v} \delta_2\right)$ *be an evolution step such that* $\delta_1 = (E_1, \#_1, \rightarrow_1)$*,* $\delta_2 = (E_2, \#_2, \rightarrow_2)$ *are two PESs satisfying (4.1), and* $v \notin E_1 \cup E_2$*. The internalization of* v *into* δ_2 *w.r.t.* s *is defined as* $\mathrm{inter}(v, \delta_1, H, \delta_2) = (E, \#, \rightarrow, \triangleright, \blacktriangleright)$*, such that:*

- $E = E_1 \cup E_2 \uplus \{v\}$
- $\# = C_1 \cup C_2 \cup C_3$ *where:*
  - $C_1 := \#_1 \cap (E_1 \setminus E_2)^2$
  - $C_2 := \#_1 \cap \#_2$
  - $C_3 := \#_2 \setminus E_1^2$
- $\rightarrow = \rightarrow_1 \cup I_1 \cup I_2$ *where:*
  - $I_1 := \{(e, v) \mid e \in H\}$
  - $I_2 := \{(v, e) \mid e \in (E_2 \setminus E_1)\}$
- $\triangleright = \{(e, v, e') \mid (e, e') \in (\rightarrow_1 \setminus \rightarrow_2) \wedge e' \in E_2\}$
- $\blacktriangleright = D \cup N \cup R$ *where:*
  - $D := \{(e, v, e) \mid e \in (E_1 \setminus E_2) \wedge \neg(e \rightarrow_1 e)\}$ *to disable dropped events*
  - $N := \{(e, v, e') \mid (e, e') \in (\rightarrow_2 \setminus \rightarrow_1)\}$ *to add new causality*
  - $R := \{(v, e, e') \mid (e, e') \in (\#_1 \setminus \#_2)\}$ *to resolve conflicts*

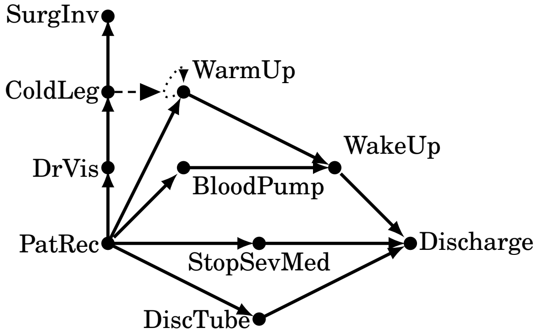


Figure 4.6.: *The internalization of the cold-leg trigger into the Cardiac-Patient workflow, with change reflection.*

Conflicts in the resulting structure can be explained as follows. C_1 represents conflicts $e_1 \# e_2$ where both e_1, e_2 are dropped afterwards. C_2 represents conflicts in both $\#_1$ and $\#_2$ (that never evolve), and C_3 represents conflicts between fresh events $e_1, e_2 \in E_2 \setminus E_1$, as well as conflicts $e_1 \# e_2$ between fresh events on one side $e_1 \in E_2 \setminus E_1$, and old events on the other side $e_2 \in E_1 \cap E_2$. The rest of the conflicts are to be dropped by the growing causality.

For instance, Figure 4.6 shows the internalization of ColdLeg into the structure of Figure 4.3 where changes, i.e. disabling WarmUp and adding SurgInv, are inferred to be made upon the execution of ColdLeg. It can be seen that the structure in Figure 4.6 is configuration-transition equivalent to the structure of Figure 4.4, although the former is a DCES while the latter is an EBES.

The internalization structure is not only a DCES, but also a SSDC (cf. Definition 3.5.7), as the following lemma shows.

Lemma 4.4.2. *The internalization* inter$(v, \delta_1, H, \delta_2)$ *is a SSDC.*

Since we consider here PESs of Definition 3.2.1, and a DCES, where the expressiveness is shown through transition relations which are more discriminating than configurations, we show here that transitions of the internalization structure before v takes place are exactly the ones of δ_1, w.r.t. to reachable configurations.

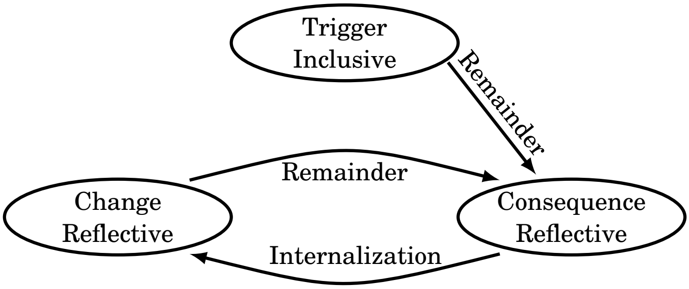


Figure 4.7.: *Transformations between Different Modeling Approaches of Evolution Steps.*

Lemma 4.4.3. *Let* $\Delta = \text{inter}(v, \delta_1, H, \delta_2) = (E, \#, \rightarrow, \triangleright, \blacktriangleright)$ *be an internalization, and let* $\rightarrow_c$ *be its configuration-transition relation. Let* $\rightarrow_{p_1}$ *be the transition relation of* δ_1*. Then:*

$$\rightarrow_{p_1} \cap [C(\delta_1)]^2 \; = \rightarrow_c \cap [\mathcal{P}(E \setminus \{v\})]^2 \cap [C(\Delta)]^2$$

As a result, $C(\delta_1) = C(\Delta)$, and since $H \in C(\delta_1)$ then $H \in C(\Delta)$, i.e. $\delta_1 \xrightarrow[H]{v} \Delta$. To take the remainder of Δ after $H \cup \{v\}$, we need to prove that $H \cup \{v\}$ is a configuration, i.e. v is enabled after H and thus can take place. That is what the following lemma proves.

Lemma 4.4.4. *Let* $\text{inter}(v, \delta_1, H, \delta_2)$ *be an internalization with a configuration-transition relation* $\rightarrow_c$*. Then* $H \rightarrow_c H \cup \{v\}$*.*

Consequently, according to Lemma 3.5.12 we conclude the following.

Corollary 4.4.5. *Let* $\Delta = \text{inter}(v, \delta_1, H, \delta_2)$*, then* $\delta_1 \xrightarrow[H]{v} \Delta$ *holds, such that* Δ *is Change-Reflective.*

Accordingly, the remainder after $H \cup \{v\}$ can be taken. Let $H' = H \cup \{v\}$. Let $\Delta' = \Delta[H']$ represents the remainder of Δ after H', and let $\delta_2' = \delta_2[H]$. Δ' and δ_2' should be transition-equivalent. But since Δ contains the events dropped in δ_2, which are part of the configurations that are unreachable, we should focus on transition equivalence w.r.t. reachable configurations, i.e. $\simeq_t$ of Definition 2.7.4.

Lemma 4.4.6. *Let* $\Delta = \text{inter}(v, \delta_1, H, \delta_2)$ *be an internalization. Then:*

$$\Delta[H \cup \{v\}] \simeq_t \delta_2[H].$$

Corollary 4.4.7. *Let* $\Delta = \text{inter}(v, \delta_1, H, \delta_2)$*, then* Δ *and* δ_2 *coincide.*

The limitations mentioned earlier regarding the use of PESs and non-growing conflicts, highlight the need for a more expressive dynamic causality structure. This could be achieved through defining higher-order dynamic causality, where growing causality can be added, shrinking causality can be dropped, and so on. Based on that, ESs of any kind—except for RCESs—could be used for δ_1, δ_2 instead of PESs, later on they can be converted to DCESs based on the expressiveness results of Section 3.5, where the result of internalization would be then a higher-order dynamic causality structure. Besides set-based dynamic causality can be used to provide growing conflict which will eliminate the need for Constraint (4.1). For instance, a conflict $a\#b$ that is added by c can be modeled by set-based mutual disabling as $\{c, a\}$ disables b and $\{c, b\}$ disables a.

4.5. Formalizing Series of Evolution Steps

Based on the correctness criteria we have, namely history preservation, we can define sequences of evolutionary steps. These sequences will form the model on which constraints regarding goal satisfaction are defined. In contrast to many related works, e.g. [35, 60], by such sequences we capture not only a single step of evolution or a pair of structures, but successive ones. This is necessary to model scenarios like the cardiac-patient use case.

Next to the dynamicity level proided by models like ESs regarding the occurrence of events (represented though system runs), evolutionary steps provide another level of dynamicity in system modeling namely the changes that occur on the level of a structure. This is illustrated in Figure 4.8.

Definition 4.5.1. *A* running workflow *is a pair* (δ, H) *where* δ *is an event structure and* $H \in \mathrm{C}(\delta)$ *is one configuration of* δ.

Since our correctness criteria are based on system runs, we denote a change or an evolution in the workflow definition by having different system runs, i.e. having no configuration-equivalence between the old and the new workflow definition. We do not use inequality between structures to

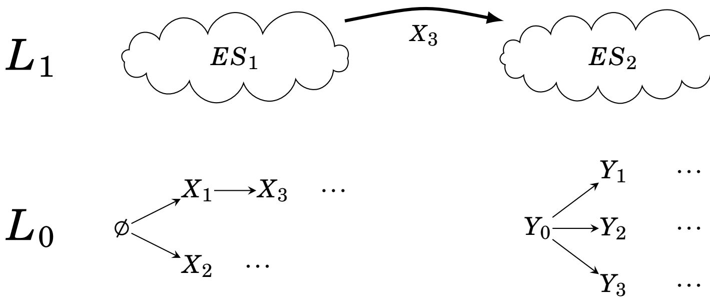


Figure 4.8.: *The two-level dynamicity offered by Evolutionary Sequences*

avoid the case that a PES, for example, is re-written in terms of a Stable ES with the same configurations, which does not denote any evolution in our case.

Definition 4.5.2. *Let* $\Delta = (\delta_1,H_1),(\delta_2,H_2),\ldots$ *be a sequence of running workflows. Then* Δ *is called a* sequence of evolutionary running workflows*, or shortly an* evolutionary sequence*, iff for all* i*:*

- $H_i \in \mathrm{C}(\delta_{i+1})$
- $H_i \subseteq H_{i+1}$
- $\delta_i \not\simeq_{\mathrm{t}} \delta_{i+1} \Longrightarrow H_{i+1} \cap E_i = H_i$ *where* E_i *is the set of events of* δ_i
- $\forall j\,.\,\delta_i \not\simeq_{\mathrm{t}} \delta_j \Longleftrightarrow H_i \neq H_j$.

The first condition emphasizes history preservation. The second implies incremental system runs. The third condition states that whenever an evolution step takes place, the only events that might take place at this step are the triggers, not to mix between system run events and triggers. The last condition ensures distinct running workflows w.r.t. configuration-equivalence. Figure 4.9 illustrates how to increment system runs in case of traces, posets and configuration transitions.

Evolutionary sequences are able to capture trigger-inclusion and change-reflection as well as trigger-exclusion with $H_{i+1} = H_i$, and allow mixing all of them. Theoretically, they can also handle the case that not only one trigger or one exception takes place in a step, but a set of them, which can be concluded by $H_{i+1} \setminus H_i$, where the execution resumes after all of them, i.e. all of them join the history. On the other hand, when the workflow definition never changes in terms of system runs, e.g. $(\delta,H_1),\ldots,(\delta,H_n)$ where

Semantics of ESs	Incrementing System Runs through:
Configurations	Set Containment Rel.: let h, h' be two sets of events, then: $h \subseteq h$
Traces	Trace Prefix Rel.: let $h = e_1, ..., e_m$ then: $h' = e_1, ..., e_m, ..., e_n$
Posets	Poset Prefix Rel. [62]: Let $h = \langle A, \leq \rangle$ and $h' = \langle A', \leq' \rangle$ be two partially ordered sets, then: $A \subseteq A' \wedge \leq = \left(\leq' \cap \left(A' \times A \right) \right)$ (cf. Section 2.4)
Configuration Transitions	Configuration Transition Rel. $\rightarrow_c$: let h,h' be two configurations, then: $h \rightarrow_c h'$ (cf. Chapter 3)

Figure 4.9.: *Incrementing system runs w.r.t. different system run representations, including: configurations, traces, posets and transitions.*

$H_i \subseteq H_{i+1}$ for $1 \leq i < n$, then the sequence simply models step transitions by showing growing configurations.

On the other hand, an evolutionary sequence can still show the case where an ES is re-written in terms of another kind of ESs, but in this case the configuration must be strictly growing. However, for the ease of discussion, we assume in the rest of this section that whenever we use different ESs that are unequal, then they are not configuration-equivalent. Further examples like the evolutionary sequence (δ_1, H_1), (δ_2, H_1), $\ldots$, (δ_n, H_1), (δ_n, H_2) where $H_1 \subset H_2$ shows how the workflow evolves in several steps before the system run proceeds. On the other hand, cyclic sequences like $(\delta_1, H), (\delta_2, H), (\delta_1, H)$ are not evolutionary due to the last condition. Similarly, the sequence $(\delta, H), (\delta, H), \ldots$ is not evolutionary.

So far, by introducing Evolutionary Sequences, we are able to fully model the evolution scenario of the Cardiac-patient use case. Figure 4.10 shows that.

4.5.1. Refinement and Equivalences of Evolution Steps

Let $\Delta = (\delta_1, H_1), (\delta_2, H_2), (\delta_3, H_3)$ be an evolutionary sequence. Then the sequence $\Delta' = (\delta_1, H_1), (\delta_2, H_2'), (\delta_2, H_2), (\delta_3, H_3)$, such that $H_2' \subset H_2$, is evolutionary. Δ' holds the same evolution steps, but shows more details about the progress of the system run than Δ. This leads us to the concept of evolution step refinement.

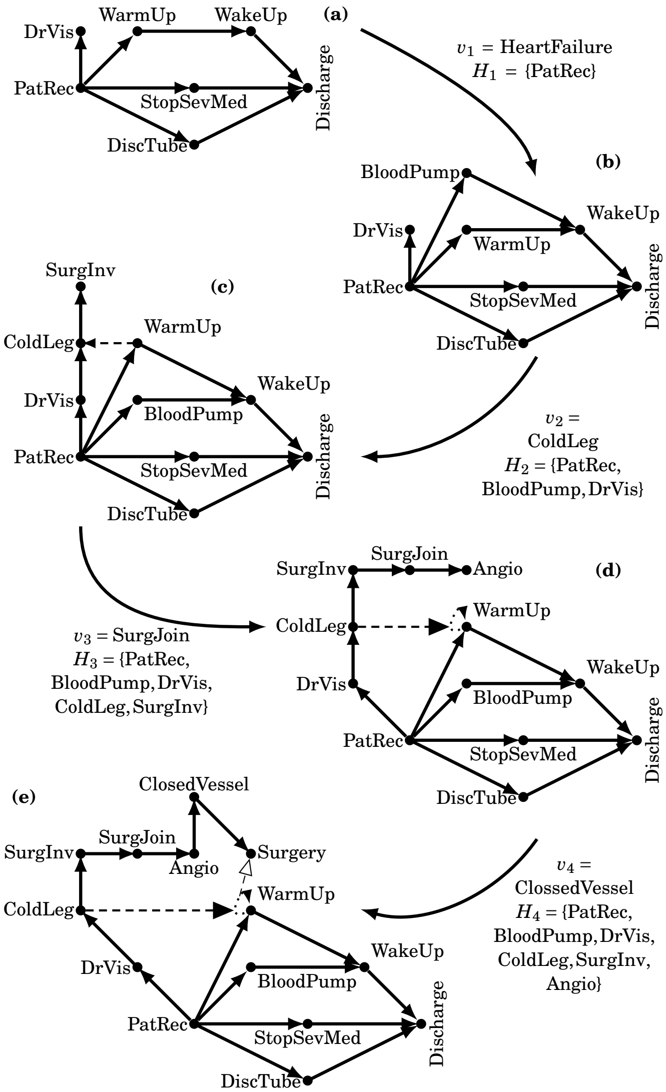


Figure 4.10.: *The evolution sequence of the cardiac-patient use case with a Consequence-Reflective PES in* (b)*, a Change-Reflective EBES in* (c)*, and a Trigger-Inclusive DCES through Internalization in* (e)*.*

Definition 4.5.3. *Let $\Delta = (\delta_1, H_1), (\delta_2, H_2), \ldots$ be an evolutionary sequence. Let $\bar{\Delta}$ denotes the set $\{(\delta_1, H_1), (\delta_2, H_2), \ldots\}$. Let f be a function defined over $\bar{\Delta}$ such that $f(\delta, H) = \{(\delta, H') \mid H' \subseteq H \wedge (\delta, H')$ is a running workflow$\}$. f is called a refinement function of Δ iff for all i:*

- $(\delta_i, H_i) \in f(\delta_i, H_i)$
- $\forall (\delta_i, H_1), (\delta_i, H_2) \in f(\delta_i, H_i) \,.\, H_1 \subset H_2 \vee H_2 \subset H_1$
- $\forall (\delta_i, H') \in f(\delta_i, H_i) \,.\, \delta_{i-1} \simeq_{\mathrm{t}} \delta_i \Longrightarrow H' \supset H_{i-1}$.
- $\delta_i \not\simeq_{\mathrm{t}} \delta_{i-1} \Longrightarrow f(\delta_i, H_i) = \{(\delta_i, H_i)\}$.

Let $R_{\Delta,f} = \bigcup\limits_i f(\delta_i, H_i)$. $R_{\Delta,f}$ is called a refinement of $\bar{\Delta}$ w.r.t. f.

The first condition denotes that a pair should be included in its refinement as an upper bound w.r.t. H. The second ensures a total order within the refinement of a pair, w.r.t. $\subset$. The third ensures a lower bound w.r.t. H. The last condition ensures that evolution steps are not refined. For consistency, (δ_0, H_0) is considered the empty structure with the empty configuration.

Definition 4.5.4. *Let Δ be an evolutionary sequence with an order $\leq$, and let f be a refinement function of Δ. Let $\Delta' = (\delta_1, H_1), (\delta_2, H_2), \ldots$ be a linearization of $R_{\Delta,f}$. Let $f' : R_\Delta \to \bar{\Delta}$ be function such that $f'(\delta, H') = (\delta, H)$ where $f(\delta, H) \ni (\delta, H')$. Δ' is called a refinement of Δ iff for all i:*

$$\big(\delta_i \neq \delta_{i+1} \wedge f'(\delta_i, H_i) \leq f'(\delta_{i+1}, H_{i+1})\big) \vee (\delta_i = \delta_{i+1} \wedge H_i \subset H_{i+1}).$$

For example: $(\delta_1, H_1'), (\delta_1, H_1), (\delta_2, H_2)$ and $(\delta_1, H_1''), (\delta_1, H_1), (\delta_2, H_2)$ are two refinements of the evolutionary sequence $(\delta_1, H_1), (\delta_2, H_2)$.

It can be proved that a refinement of an evolutionary sequence is itself an evolutionary sequence. As a result, the progress of the run in the same process can be abstracted while looking from an evolution perspective. This leads us to define an equivalence relation over evolutionary sequences.

Definition 4.5.5. *Let Δ_1, Δ_2 be two evolutionary sequences. Let t be a function defined over evolutionary sequences, such that $t(\Delta) = \{(\delta, H, \delta') \mid \exists i \,.\, \delta = \delta_i \wedge \delta' = \delta_{i+1} \wedge \delta \neq \delta'\}$. Then we define the relation $\equiv$ as $\Delta_1 \equiv \Delta_2$ iff $t(\Delta_1) = t(\Delta_2)$.*

It is clear that $\equiv$ is an equivalence relation. Let $[\Delta]$ denotes the class of evolutionary sequences equivalent to Δ w.r.t. $\equiv$. $[\Delta]$ is called an *evolutionary trace*. Traces abstract from the internal progress of system runs. For instance, a sequence and its refinement belong to the same evolutionary trace. Besides, the two sequences of the last example belong to the same trace.

4.5.2. Special Cases of Evolution

Let us examine the evolutionary sequence (δ_1,H_0),(δ'_1,H_1),(δ_2,H_2) where $H_1 \in \mathrm{C}(\delta_1)$. It shows that the detailed changes never affected the system run which could have progressed through the old process. That is what we call *ineffective* changes w.r.t. the execution path. It can be summarized to the sequence $(\delta_1,H_1)(\delta_2,H_2)$ which shows that all changes occurred at once leading to an evolution step that took place at H_1.

Indeed the last case is a generalization of the case (δ_1,H_0), (δ'_1,H_0), (δ''_1,H_0), (δ_2,H_1) which can be summarized to (δ_1,H_0), (δ_2,H_1). This leads us to the following definitions.

Definition 4.5.6. *Let* $\Delta = (\delta_1,H_1),\ldots$ *be an evolutionary sequence.* Δ *is* Static-Evolution-Free*, abbreviated as* SEF*, if and only if* $\nexists i\,.\,H_i = H_{i+1}$.

To obtain a SEF sequence out of an evolutionary sequence Δ, assume we have a sub-sequence of Δ, made up of all the instances sharing the same configuration, then we take only the first instance out of it and drop the rest, and so on.

Definition 4.5.7. *Let* $\Delta = (\delta_1,H_1),\ldots$ *be an evolutionary sequence.* Δ *is* Effective-Changes*, abbreviated as* EC*, if and only if* $\nexists i\,.\,H_{i+1} \in \mathrm{C}(\delta_i)$.

Clearly, EC sequences are SEF.

4.6. Evolution towards Goal Satisfaction

The correctness criteria of evolution are discussed only w.r.t. the history of a workflow so far. For the future of a workflow after an evolution step takes place, we consider goals. Goals capture the objectives a system should satisfy, and form the root to detect conflicts in requirements; yet they are absent from many formal models [30]. Goal-Oriented Requirements Engineering and Elaboration [78] are concerned with defining goals, and their use in structuring and modifying requirements. This includes refining goals into sub-goals, defining their relations in-between, and the way to satisfy them. Since DCs are directed by their goals [59], the constraints over the future in evolutionary sequences should focus on satisfying a given goal.

In Goal-Oriented Requirements Engineering [29, 78], goals can be classified into different types. Some classify them as *functional* and *non-functional* goals [78]. Functional are concerned with services the system is expected to deliver at the end, while non-functional[1] are concerned with how the system delivers services rather than what it delivers. Others classify goals as *soft* and *hard* goals [28]. Satisfaction of soft goals cannot be established in a clear-cut way [78], while the satisfaction of hard goals can be established in a clear-cut sense and verified by verification techniques [29].

Furthermore, goals are refined to subgoals. Subgoals can be connected through links (e.g. *AND, OR* links) [28]. Subgoals of *soft* goals can be measured depending on how much they contribute to the main goal [78]. AND links denote a conjunctive nature of satisfaction, while OR links denote a disjunctive nature. For instance, in the Cardiac-patient use case, the goal of Discharge is of a *hard* type, and refined into three subgoals: WakeUp, DiscTube, and StopSevMed, connected by AND links. Goals can be formally refined as in [30] according to patterns. However in our case goals are of a medical nature and are refined usually by medical personnel, according to a pre-defined medical guideline.

Goals are usually expressed in terms of properties to be maintained in the system [29, 78]. For example, KAOS uses Temporal Logic with real time capabilities to assert properties [28, 29]. Since we are using ESs as a formalism, goals would be expressed in our case in terms of events to be reached. We then represent a goal by a set of events, which all must take place for a goal to be reached, i.e. a conjunctive nature. For sub-goals with OR links, i.e. of disjunctive nature, disjunctive causality of ESs like Dual and Stable ESs could be used to model that.

Additionally goals, of which satisfaction is done through temporal behavior verification, can be *Achieve* and *Maintain* goals [28]. Achievement goals should eventually take place, while Maintain goals should be permanently available, unless some other property holds. For instance, the Discharge

[1]No common definition of non-functional requirements exists. Glinz [38] addresses this problem by showing the different definitions available and the problems accompanied with each, and then contributes concepts for solving these problems.

goal of the Cardiac-Patient use case should be eventually reached, i.e. an *Achieve* goal. This matches the concept of a DC that starts with an unreached goal, and progress until the goal is reached finally.

To model DC workflows with goals, we follow the same approach as DCR Graphs [54] and equip our evolutionary sequences with a set of events—representing a goal—to be reached. In order to achieve a given goal, and because we are addressing evolving structures, the goal must be reachable. In other words a structure should evolve and adapt such that the set of goal events becomes reachable. To this end we introduce the concept of *evolution* w.r.t. goals. For instance, the set G = {StopSevMed, DiscTube, WarmUp, WakeUp, Discharge} becomes reachable in the structure of Figure 4.10 (b) after adding the blood pump.

Definition 4.6.1. *Let* MC(δ) *denote the set of maximal configurations of an ES δ w.r.t. $\subseteq$, and let G denotes the set of goal events. An evolution step* $(\delta_1,H_1),(\delta_2,H_2)$ *is said to be a* Goal-Evolution *iff* $\exists C \in \mathrm{MC}(\delta_2[H_2])\,.\,G \subseteq C$.

We focus on the remainder of δ_2 since goal events might be reachable before the history H_2 takes place, but not afterwards. For example, the event Discharge is reachable in Figure 4.10 (c), but not after the event ColdLeg takes place. Besides, we consider a set of events reachable in a structure if there exists a maximal system run that contains that set. This allows for extending the workflow with new system runs such that the goal events are reachable again. For instance, the event Discharge was not reachable anymore in Figure 4.10 (c) after H = {PatRec, DrVis, ColdLeg}, but by extending the structure in (d) and (e), it became reachable again. This is different from other studies, e.g. [27], that address non-evolving structures and consider a set of events reachable iff all maximal system runs contain this set.

Since a goal would never be reached before it becomes reachable, the concept of Evolution cen be used to warn the modeller or the designer of a workflow whenever an exception takes place. For example, the cold-leg disabled the goal such that a further adaptation was required to make the goal reachable again. Such a check can be done for example with our tool via linearization and reachability checking as illustrated in Section 1.6.

In Indulgent Algorithms [39] which tolerate failure detectors in Distributed Algorithms, Liveness [70] is eventually ensured after regaining stability in the system. Similarly, we assume that adapting the workflow such that goals become reachable again after an exception, which might be seen as stabilizing the system in Indulgent Algorithms, would lead eventually to goal satisfaction. This is due to the reason that progress in system runs is usually made towards a maximal system rum which contains goal events. Measures about how far the progress is from reaching the goal can be defined; this is discussed in Section 4.7. The goal is reached then in a DC workflow, represented by an evolutionary sequence, iff $\exists i\,.\,G \subseteq H_i$.

In some other DCs, goals are of the type *Maintain*. For instance, as taken from [68], the goal in a given DC is to perform read-write operations persistently. Note that in event structures such goals would be represented as different events since events cannot be repeated. Accordingly the definition of evolution could be adapted accordingly. However since the goal in our use case is of the type *Achieve*, we base our definition on type *Achieve*.

4.7. Evaluation

In this chapter, we provided a general understanding of how evolution in workflows can be modeled formally. Since we focus on case management, a main criteria to consider was preserving the history of the case workflow, that preceded the evolution. We compared our approach to related works and showed the big overlap concerning history preserving as a major criteria that leads to migrations of instances when considering schemas.

To model an evolution step, we provided different possible approaches. One major approache was concerned with modeling the effects and consequences of the evolution, without an explicit modeling of changes nor how they are caused by the trigger. We called this approach Consequence-Reflective. Another proposed approach was concerned, on the other hand, with modeling changes brought by the trigger; we called this approach Change-Reflective. Change modeling would require and imply capturing the evolution trigger as part of the model, while consequence capturing would not allow for tracing the trigger as part of the running-workflow history.

Given an evolution step modeled in a Consequence-Reflective approach, the need might emerge in some cases to extract the changes that were applied to the first (old) workflow definition, and that led to the second (new) definition. Reasons behind could be the ability to trace the evolution trigger in the history or run of the workflow; or the wish to learn from the workflow evolution, in order to apply the same changes to other similar workflows or to the schema type as a generalization. To that end, we provided a mechanism for inferring such changes, that would give the result in the form of a DCES of Section 3.5 since changes would become a past by the moment of learning. This mechanism would then connect the two main models offered in this thesis, namely evolution for unforeseen changes and Dynamic Causality structures for pre-planned changes, and integrate them both in one framework.

Finally, to limit arbitrary changes, and to guide evolution, certain and constraints were defined in a way suitable and unique for our use case of Dynamic Coalitions. These criteria were defined in form of goals to be satisfied. Goals were defined in form of events to be reached since the formalism we use is event-based. When such events become unreachable due to exceptions or changes, the structure then should be adapted such that events of the goal become reachable again. This was formally modelled through the concept of Evolution. Beforehand, evolutionary steps were formally defined and their modeling features were well analyzed through the notion of evolutionary sequences. Evolutionary sequences allowed for capturing both evolution steps as well as progress in the system run. As well, relations and equivalences between different sequences were defined, next to methods allowing to focus on evolution steps and abstract them from other details like a system run progress.

Application and Practical Considerations: One main application of the formalisms of this chapter would be based on the constraints defined over evolution. This includes history preserving constraints as well as goal-orientation ones. Whenever history preserving is violated by a new evolution step, a warning can be given to the modeller informing her that she is deviating from the case being modelled. On the other hand, since a predefined goal is to be reached finally, a warning can be given to the modeller whenever the defined goal becomes unreachable in any evolution step.

Measuring Progress towards Goal Satisfaction: Measures of how far a running workflow is from reaching a goal can be defined depending on the case. *Soft Goals* [28] for instance, which measures how sub-goals contribute positively or negatively to reach a goal, can be used. As well, a maximum number of evolution steps, that can take place before the goal is satisfied, can be defined as a measure. On the other hand, when considering a real time situation as in hospitals and surgery departments, a maximum timeout can be defined for a goal to be satisfied, where the workflow can evolve to make the goal reachable during the timeout. That is indeed the situation in the *Cardiac Intensive Care Unit (CICU)* were we obtained the case study from. In this work, we give the basic definition and notion of goal satisfaction, such that changes are not left arbitrary, while other variants of constraints and applications can be based on the basic definition and understanding offered here.

5. Domain-Oriented Extensions: Priority in Event Structures

5.1. Introduction

From the several visits to, and observations made in, *Charité Hospital in Berlin*, the need for quantitative and qualitative extensions, e.g. timing, of the underlying model in medical scenarios in general were found. For example *a doctor should join in an hour*. Such extensions are not part of the core modeling of dynamic coalitions nor adaptive processes, rather orthogonal and can be used in general processes. The most important ones are:

- *Timing*; defining maximum timing for events occurrences, minimum timing and delays. Example: not less than one hour after the second doctor' s join, the first Dr. on duty can leave.
- *Urgency*; where some events must occur at some point of time. Example: when a patient gets a stroke, the ambulance must involve to transfer her to the emergency.
- *Priority*; where some events should have higher priorities than others to happen. Example: when a stroke patient has another stroke during the rehabilitation, the ambulance has a higher priority to involve and interrupt other running members.

As shown in 2.9, Timing and Urgency have been added to event structures through several publications [43, 44, 79, 24]. What is still missing and have not been added to ESs is the notion of priority.

Overview In this chapter, we add priority to certain kinds of ESs as a state of art, including disjunctive causality, disabling and causal ambiguity. The rest of event structure types e.g. Stable ESs can be extended with priority analogously. Since this chapter was developed before the concept of dynamic causality, priority was not considered for variants of Dynamic Causality ESs. In Section 5.2, we start with the simplest form of ESs, the Prime ESs, add priority to it, discuss the overlapping between priority and the other relations of Prime ESs, and show how to reduce this overlapping. In Section 5.3, we add priority to Bundle ESs and investigate the relation between priority and other event relations of BESs like enabling and precedence. In Section 5.4 and Section 5.5 we then study the two extensions Extended Bundle ESs and Dual ESs of Bundle ESs and how their different causality models modify the relationship of priority and the other event relations of the ESs. In Section 5.6, we summarize the work and conclude by comparing the results, and examine the applicability by an example from healthcare. As stated in Section 1.8, this work was published before in [3].

Related Work. Priorities are used as a mechanism to provide interrupts in systems concerned with processes. For example, in Process Algebra, Baeten et al. were the first to add priority in [6]. They defined it as a partial order relation <. Moreover, Camilleri, and Winskel integrated priority within the Summation operator in CCS [22]. Also, Lüttgen in [51] and Cleaveland et al. in [25] considered the semantics of Process Algebra with priority.

In Petri Nets, which are a non-interleaving model like Event Structures, Bause in [10, 11] added priority to Petri Nets in two different ways: static and dynamic. Dynamic means the priority relation evolves during the system run, while the static one means it is fixed since the beginning and will never change till the end. In that sense, static priority is what we define here. Priority could be seen in Programming Languages. In Ada 95 for example, priority can be assigned to tasks (processes) which were added natively to the language [56] supporting real-time systems.

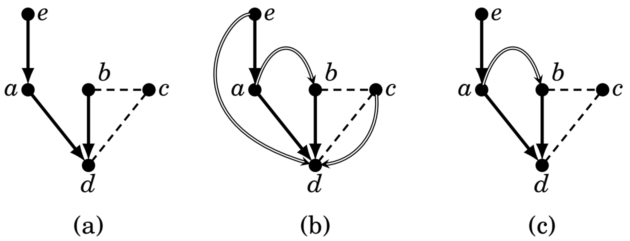


Figure 5.1.: *A Prime ES without priority in* (a)*, with priority in* (b)*, and after dropping redundant priority pairs in* (c)*.*

5.2. Priority in Prime Event Structures

If we add priority to PESs, it should be a binary relation between events such that, whenever two concurrent events ordered in priority are enabled together, the one with the higher priority must pre-empt the other.[1] Thus we add a new acyclic relation $\lessdot \subseteq E \times E$, the *priority* relation, to Prime ESs and denote the pair $(\delta, \lessdot)$ as *prioritized Prime ES (PPES)*. Later on, we add priority in a similar way to other kinds of Event Structures. Sometimes, we expand a prioritized ES $(\delta, \lessdot)$, where $\delta = (E, r_1, r_2)$, to $(E, r_1, r_2, \lessdot)$.

Figure 5.1 (b) illustrates a prioritized variant of Figure 5.1 (a), where the priority relation is represented by a double-lined arrow from the higher-priority event to the lower-priority one, showing the direction of precedence (pre-emption). Sometimes representing both an Event Structure and its associated priority relation in the same diagram becomes too confusing. In that case we visualize the priority relation in a separate diagram next to the structure.

Let us define the interpretation of $\lessdot$ in a formal way: let $\sigma = e_1, \ldots, e_n$ be a sequence of events in a PPES $\delta' = (\delta, \lessdot)$. We call σ a trace of δ' iff it is 1.) a trace of δ, and 2.) satisfies the following constraint:

$$\forall i < n \,.\, \forall e_j, e_h \in \bar{\sigma} \,.\, \left(e_j \neq e_h \;\wedge\; e_j, e_h \in \mathrm{en}_\delta(\sigma_i) \;\wedge\; e_h \lessdot e_j\right) \implies j < h \tag{5.1}$$

For example, the sequence $ebad$ is a trace of Figure 5.1 (a) but not of Figure 5.1 (b) due to priority. Let us denote the set of traces of a structure

[1]In fact we could define priority as a partial order. However after dropping redundant priority pairs as explained later the priority relation is usually no longer transitive, i.e. no longer a partial order.

as $\mathrm{T}(\delta)$. By definition, the traces of a PPES $\delta' = (\delta, \lessdot)$ are a subset of the traces of the Prime ES δ.

Proposition 5.2.1. $\mathrm{T}(\delta, \lessdot) \subseteq \mathrm{T}(\delta)$.

If we analyze Figure 5.1 (a) and (b) we observe that, because of the conflict relation, no trace can contain both c and d. And even without the priority relation the enabling relation ensures that e always has to precede d. Since neither c and d nor e and d can be enabled together, i.e. do never compete, applying the priority relation between them is useless or trivial. Indeed we can always reduce the priority relation by dropping all pairs between events that are under $\leq$ or $\#$ without affecting the set of traces.

Theorem 5.2.2. *Let $(E, \#, \leq, \lessdot)$ be an PPES, and let*

$$\lessdot' := \lessdot \setminus \left\{(e,e') \mid e'\#e \vee e' \leq e \vee e \leq e'\right\}.$$

Then $\mathrm{T}(E, \#, \leq, \lessdot) = \mathrm{T}\big(E, \#, \leq, \lessdot'\big)$.

Proof. Straightforward from the definitions of traces, (5.1), and (2.1). □

Figure 5.1 (c) shows the result of dropping the priority pairs that are redundant in Figure 5.1 (b). Note that after dropping all redundant pairs, there is no overlapping, neither between the priority and the enabling relation, nor between the priority and the conflict relation. The following theorem insures minimality of reduction.

Theorem 5.2.3. *Let $(E, \#, \leq, \lessdot)$ be an PPES, and let*

$$\lessdot' := \lessdot \setminus \left\{(e,e') \mid e'\#e \vee e' \leq e \vee e \leq e'\right\} \text{ and let } \lessdot'' \subset \lessdot'.$$

Then $\mathrm{T}(E, \#, \leq, \lessdot) \neq \mathrm{T}\big(E, \#, \leq, \lessdot''\big)$.

Proof. Straightforward from the definitions of traces, (5.1), and (2.1). □

Such a result is good for a modeler, since it implies unambiguity about whether a priority relation affects the behavior or not. In other words, after dropping all the redundant priority pairs, the remaining priority pairs always lead to pre-emption, limit concurrency and narrow down the possible traces. This is not the case for the following ESs, since they offer other causality models.

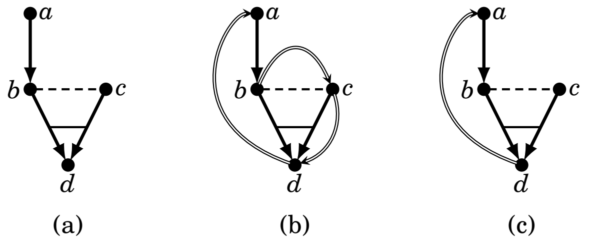


Figure 5.2.: *A Bundle ES without priority in* (a), *with priority in* (b), *and after dropping redundant priority pairs in* (c).

5.3. Priority in Bundle Event Structures

Again we add priority $\lessdot \subseteq E \times E$ to EBESs as a binary acyclic relation between events such that, whenever two events are enabled together, the one with the higher priority pre-empts the other. We denote $\beta' = (\beta, \lessdot) = (E, \#, \mapsto, \lessdot)$ as *prioritized Bundle ES (PBES)*. Figure 5.2 (b) illustrates a prioritized version of the BES in Figure 5.2 (a).

Also the semantics of $\lessdot$ is defined similarly to Prime ESs. A sequence of events $\sigma = e_1, \ldots, e_n$ is a trace of $(\beta, \lessdot)$ iff 1.) $\sigma \in \mathrm{T}(\beta)$ and 2.) σ satisfies the following constraint:

$$\forall i < n \,.\, \forall e_j, e_h \in \bar{\sigma} \,.\, \left(e_j \neq e_h \wedge e_j, e_h \in \mathrm{en}_\beta(\sigma_i) \wedge e_h \lessdot e_j \right) \implies j < h \tag{5.2}$$

Again the traces of a PBES $(\beta, \lessdot)$ are a subset of the traces of the corresponding BES β.

Proposition 5.3.1. $\mathrm{T}(\beta, \lessdot) \subseteq \mathrm{T}(\beta)$.

For example the sequence *cad* is a trace of the BES in Figure 5.2 (a), but it is not a trace of the PBES in Figure 5.2 (b). Of course a larger priority relation filters more traces out than a smaller one.

Lemma 5.3.2. *Let* $(\beta, \lessdot)$ *and* $(\beta, \lessdot')$ *be two PBES with* $\lessdot' \subseteq \lessdot$. *Then:*

$$\mathrm{T}(\beta, \lessdot) \subseteq \mathrm{T}(\beta, \lessdot') \,.$$

Proof. Straightforward from the definition of traces, (5.2), and $\lessdot' \subseteq \lessdot$. $\square$

We adapt the notion of a configuration to prioritized BESs such that $\sigma \in \mathrm{C}(\beta, \lessdot)$ for a PBES $(\beta, \lessdot)$ iff $\exists \sigma \in \mathrm{T}(\beta, \lessdot) \,.\, C = \bar{\sigma}$. In Section 2.4 the semantics of BESs is defined by families of posets. Unfortunately doing the same for PBESs is not that simple. Consider the poset $\boxed{a \quad c \longrightarrow d}$ of the BES β in Figure 5.2 (a). According to Figure 5.2 (b), d has a higher priority than a, i.e. $a \lessdot d$. Hence $\boxed{a \quad c \longrightarrow d}$ does not describe the semantics of the PBES $(\beta, \lessdot)$ with respect to the configuration $\{a, c, d\}$, because $cad \in \mathrm{T}(\beta)$ but $cad \notin \mathrm{T}(\beta, \lessdot)$. In fact we cannot describe the semantics of PBESs by families of posets in a simple way. Instead, to describe the semantics of $(\beta, \lessdot)$ with respect to $\{a, c, d\}$ we need the two different posets $\boxed{a \longrightarrow c \longrightarrow d}$ and $\boxed{c \longrightarrow d \longrightarrow a}$.

The enabling relation defines precedence between events as used for $\prec_C$ in (2.4.3), whereas priority rather defines some kind of conditional precedence. Priority affects the semantics only if the related events are enabled together. Thus the same problem with the definition of posets appears for all kinds of Event Structures that are extended by priority. We leave the problem on how to fix the definition of posets as future work.

5.3.1. Priority versus Enabling and Conflict

Again, as in Section 5.2, we can reduce the priority relation by removing redundant pairs, i.e. pairs that due to the enabling or conflict relation do not affect the semantics of the PBES. First we can | as already done in PPES | remove a priority pair $e \lessdot e'$ or $e' \lessdot e$ between an event e and its cause e', because an event and its cause are never enabled together. Therefore e.g. the pair $d \lessdot c$ in Figure 5.2 (b) is redundant because of $\{b, c\} \mapsto d$. Also a priority pair $e \lessdot e'$ between two events that are in conflict is redundant, because these conflicting events never occur in the same trace. Consider for example the events b and c in Figure 5.2 (b). Because of $b \# c$ the pair $c \lessdot b$ is redundant.

Lemma 5.3.3. *Let $\beta = (E, \#, \mapsto)$ be an BES, and let $e, e' \in E$ such that:*

$$\exists X \subseteq E \,.\, e \in X \wedge X \mapsto e'.$$

Then: $\forall \sigma = e_1, \ldots, e_n \in \mathrm{T}(\beta) \,.\, \nexists i < n \,.\, e, e' \in \mathrm{en}_\beta(\sigma_i)$.

Proof. Let $\sigma = e_1, \ldots, e_n \in \mathrm{T}(\beta)$ and $X \subseteq E$ such that $e \in X \wedge X \mapsto e'$. Assume $e \in \mathrm{en}_\beta(\sigma_i)$ for some $i < n$. Then:

$$e \in \mathrm{en}_\beta(\sigma_i) \wedge e \in X \wedge X \mapsto e' \overset{(2.3)}{\Longrightarrow}$$

$$e \in \mathrm{en}_\beta(\sigma_i) \wedge e \in X \wedge X \mapsto e' \wedge \big(\forall e'' \in (X \setminus \{e\}) \,.\, e\#e''\big) \overset{(2.4)}{\Longrightarrow}$$

$$X \mapsto e' \wedge \big(\forall e'' \in X \,.\, e'' \notin \bar{\sigma}_i\big) \overset{(2.4)}{\Longrightarrow} e' \notin \mathrm{en}_\beta(\sigma_i)$$

Hence $\nexists i < n \,.\, e, e' \in \mathrm{en}_\beta(\sigma_i)$. □

The last lemma proves that an event cannot be enabled together with its cause in BESs, i.e. whenever an event e is in a bundle set X pointing to e', i.e. such that $X \mapsto e'$, then e and e' cannot be enabled together. This lemma helps proving the next theorem regarding priority reduction.

Theorem 5.3.4. *Let* $(\beta, \lessdot) = (E, \rightsquigarrow, \mapsto, \lessdot)$ *be a PBES, and let* $\lessdot' = \lessdot \setminus \big\{(e,e'), (e',e) \mid e\#e' \vee \big(\exists X \subseteq E \,.\, e \in X \wedge X \mapsto e'\big)\big\}$. *Then* $\mathrm{T}(\beta, \lessdot) = \mathrm{T}(\beta, \lessdot')$.

Proof. $\mathrm{T}(\beta, \lessdot) \subseteq \mathrm{T}(\beta, \lessdot')$ follows from Lemma 5.3.2.

To show $\mathrm{T}(\beta, \lessdot') \subseteq \mathrm{T}(\beta, \lessdot)$, assume a trace $\sigma = e_1, \ldots, e_n \in \mathrm{T}(\beta, \lessdot')$. We have to show that $\sigma \in \mathrm{T}(\beta, \lessdot)$, i.e. that $\sigma \in \mathrm{T}(\beta)$ and that σ satisfies Condition (5.2). $\sigma \in \mathrm{T}(\beta)$ follows from $\sigma \in \mathrm{T}(\beta, \lessdot')$ by the definition of traces. σ satisfies Condition (5.2) when $\forall i < n \,.\, \forall e_j, e_h \in \bar{\sigma} \,.\, e_j \neq e_h \wedge e_j, e_h \in \mathrm{en}_\beta(\sigma_i) \wedge e_h \lessdot e_j \implies j < h$. Let us fix $i < n$ and $e_j, e_h \in \bar{\sigma}$. Assume $e_j \neq e_h$, $e_j, e_h \in \mathrm{en}_\beta(\sigma_i)$, and $e_h \lessdot e_j$. It remains to prove that $j < h$. Because of the definition of $\lessdot'$, there are three cases for $e_h \lessdot e_j$:

Case $e_h \lessdot' e_j$**:** Then $e_h \lessdot' e_j \wedge e_j, e_h \in \bar{\sigma} \wedge \sigma \in \mathrm{T}(\beta, \lessdot') \overset{(5.2)}{\Longrightarrow} j < h$.

Case $e_j\#e_h \vee e_h\#e_j$**:** This case is not possible, because it is in contradiction to (2.4.2), (2.4), and $e_j, e_h \in \bar{\sigma}$.

Case $\exists X \subseteq E \,.\, \big(e_h \in X \wedge X \mapsto e_j\big) \vee \big(e_j \in X \wedge X \mapsto e_h\big)$**:** This case is not possible, because it is in contradiction to $e_j, e_h \in \mathrm{en}_\beta(\sigma_i)$ and Lemma 5.3.3. □

Note that priority is redundant for all pairs of events that are directly related by the bundle enabling relation or the conflict relation regardless of the direction of the priority pair. We say that this reduction is done at the structure level, since it is done w.r.t. the relations which are part of the Event Structure.

In PPESs enabling is a transitive relation and we can drop all priority pairs between events that are related by enabling. In the case of PBESs neither conflict nor enabling are transitive relations. For example in the event structure $\overset{e_1}{\bullet}\text{-----}\overset{e_2}{\bullet}\text{-----}\overset{e_3}{\bullet}$ (which can be both; a PES as well as a BES) we have $e_1 \# e_2$ and $e_2 \# e_3$ but not $e_1 \# e_3$. Accordingly we cannot drop a priority pair $e_1 \lessdot e_3$ because else the sequence $e_1 e_3$ becomes a trace.

However in PPESs enabling is transitive, so whenever $e_1 \leq e_2$ and $e_2 \leq e_3$ there is $e_1 \leq e_3$ and we can also drop priority pairs relating e_1 and e_3 (compare e.g. with e, a, and d in Figure 5.1). In PBES the situation is different. For the PBES in Figure 5.2 we have $\{a\} \mapsto b$ and $\{b, c\} \mapsto d$ but d does not necessarily depend on a and thus we cannot drop the pair $a \lessdot d$ since $cad \notin \mathrm{T}(\beta, \lessdot)$. Unfortunately this means that we do not necessarily drop the whole redundancy in priority if we reduce the priority relation as described in Theorem 5.3.4. For example $e_1 \lessdot e_3$ is redundant in $(\{e_1, e_2, e_3\}, \emptyset, \{\{e_1\} \mapsto e_2, \{e_2\} \mapsto e_3\}, \{e_1 \lessdot e_3\})$, because in this special case e_1 is indeed a necessary cause for e_3. Thus for PBESs $\lessdot'$ is not necessarily minimal, i.e. we cannot prove $\forall \lessdot'' \subset \lessdot' .\ \mathrm{T}((E, \rightsquigarrow, \mapsto, \lessdot)) \neq \mathrm{T}\big((E, \rightsquigarrow, \mapsto, \lessdot'')\big)$ as we have done in Theorem 5.2.3 for PPESs.

For the PBES in Figure 5.2 the reduction described in Theorem 5.3.4 indeed suffices to remove all redundant priority pairs. The result is presented in Figure 5.2 (c).

5.3.2. Priority versus Precedence

In order to identify some more redundant priority pairs we consider configurations and posets. If we analyze for example the configurations $\{a, b, c\}$ and $\{a, c, d\}$ of the PBES in Figure 5.2, we observe that, because of $\{a\} \mapsto b$ and $\{b, c\} \mapsto d$, the priority pair $a \lessdot d$ is redundant in the first configuration while it is not in the second one. Thus, in some cases, i.e. with respect to some configurations (or posets), we can also ignore priority pairs of events that are indirectly related by enabling. Since such a redundancy is relative to specific configurations and their traces, and since dropping priority pairs affects the whole set of traces obtained from a ES, we use the term "ignorance" rather than "dropping" for distinction, and we say that this ignorance is done at the configuration level. Priority ignorance is necessary while linearizing configurations and trying to obtain traces.

The cases in which priority pairs are redundant with respect to some configuration C are already well described by the precedence relation $\preceq_C$, i.e. we can identify redundant priority pairs easily from the posets for C. Note that in BESs (and also EBESs) each configuration leads to exactly one poset. The priority pair $a \lessdot d$ is obviously redundant in the case of $\boxed{a \longrightarrow b \longrightarrow d}$ but not in the case of $\boxed{a \quad c \longrightarrow d}$.

To formalize this let $\mathrm{T}(\beta, \lessdot) \upharpoonright_C := \{\sigma \mid \sigma \in \mathrm{T}(\beta, \lessdot) \wedge \bar{\sigma} = C\}$ be the set of traces over the configuration $C \subseteq E$ for some BES $\beta = (E, \rightsquigarrow, \mapsto)$. Therefore $\mathrm{T}(\beta, \lessdot) \upharpoonright_C$ consists of all the traces of $\mathrm{T}(\beta, \lessdot)$ that are permutations of the events in C. Then for all configurations C all priority pairs $e \lessdot e'$ such that $e' \preceq_C e$ or $e \preceq_C e'$ can be ignored.

Theorem 5.3.5. *Let $(\beta, \lessdot)$ be a PBES, $\langle C, \preceq_C \rangle \in \mathrm{P}(\beta)$, and let*

$$\lessdot' := \lessdot \setminus \{(e, e') \mid e' \preceq_C e \vee e \preceq_C e'\}.$$

Then $\mathrm{T}(\beta, \lessdot) \upharpoonright_C = \mathrm{T}(\beta, \lessdot') \upharpoonright_C$.

Proof. Note that by induction on $\preceq$ and Lemma 5.3.3, $e_j \preceq_C e_h$ as well as $e_h \preceq_C e_j$ imply that e_j and e_h cannot be enabled together in a trace of $\mathrm{T}(\beta) \upharpoonright_C$. With this argument the proof is straightforward from the definitions of traces, $\lessdot'$, traces over a configuration, Lemma 5.3.2, and (5.2). □

Consider once more the PBES $(\beta, \lessdot)$ of Figure 5.2 with respect to the configuration $\{a, b, d\}$. We have $\{a\} \mapsto b$, $\{b, c\} \mapsto d$, and $a \lessdot d$. As explained before we cannot drop the priority pair $a \lessdot d$, because of the sequence $cad \notin \mathrm{T}(\varepsilon, \lessdot)$. However with Theorem 5.3.5 we can ignore $a \lessdot d$ for the semantics of $(\beta, \lessdot)$ if we limit our attention to $\{a, b, d\}$, because $\mathrm{T}(\beta, \lessdot) \upharpoonright_{\{a,b,d\}} = \{abd\} = \mathrm{T}(\beta) \upharpoonright_{\{a,b,d\}}$.

For PBESs ignorance ensures that $\lessdot'$ is minimal with respect a configuration C.

Theorem 5.3.6. *Let $(\beta, \lessdot)$ be a PBES and $\langle C, \preceq_C \rangle \in \mathrm{P}(\beta)$ for some configuration $C \in \mathrm{C}(\beta, \lessdot)$. Let also:*

$$\lessdot' := \lessdot \setminus \{(e, e') \mid e' \preceq_C e \vee e \preceq_C e'\}, \lessdot'' \subset \lessdot'.$$

Then $\mathrm{T}(\beta, \lessdot) \upharpoonright_C \neq \mathrm{T}(\beta, \lessdot'') \upharpoonright_C$.

Proof. Because of $\lessdot'' \subset \lessdot'$, there are some $e, e' \in E$ such that $e \lessdot e'$ but $e \not\lessdot' e'$, $e' \not\preceq_C e$, and $e \not\preceq_C e'$. Note that each linearization of a given poset that respects the precedence relation is a trace [47]. Thus $e' \not\preceq_C e$ and $e \not\preceq_C e'$ imply that $\mathrm{T}(\beta, \lessdot'') \upharpoonright_C$ contains a trace such that e and e' are enabled together and e precedes e'. Because of $e \lessdot e'$ such a trace cannot be contained in $\mathrm{T}(\beta, \lessdot) \upharpoonright_C$. So $\mathrm{T}(\beta, \lessdot) \upharpoonright_C \neq \mathrm{T}(\beta, \lessdot'') \upharpoonright_C$. □

In the following two sections we consider two extensions of Bundle ESs.

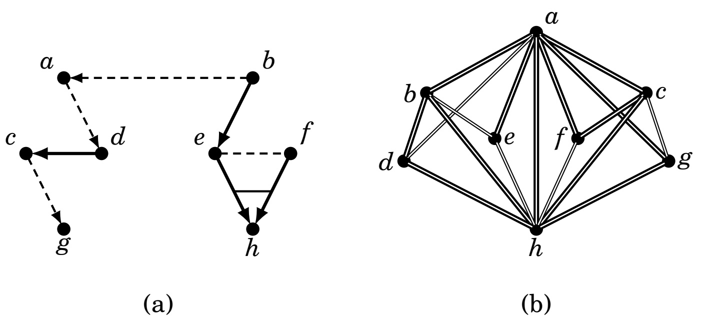


Figure 5.3.: *A PEBES* $(\varepsilon, \lessdot)$ *with* ε *in* (a) *and* $\lessdot$ *in* (b) *as a Hasse diagram with transitivity exposed, where thin links represent redundant priority pairs.*

5.4. Priority in Extended Bundle Event Structures

As in PBESs, we equip EBESs with the same priority relation, and we call $(\varepsilon, \lessdot) = (E, \rightsquigarrow, \mapsto, \lessdot)$ a *prioritized Extended Bundle ES (PEBES)*, where $\varepsilon = (E, \rightsquigarrow, \mapsto)$ is an EBES and $\lessdot \subseteq (E \times E)$ is the acyclic priority relation. Figure 5.3 illustrates an example of a PEBES with the EBES in Figure 5.3 (a) and the priority relation in Figure 5.3 (b). Furthermore, a sequence of events $\sigma = e_1, \ldots, e_n$ is a trace of $(\varepsilon, \lessdot)$ iff 1.) $\sigma \in \mathrm{T}(\varepsilon)$ and 2.) σ satisfies the following constraint:

$$\forall i < n \, . \, \forall e_j, e_h \in \bar{\sigma} \, . \, e_j \neq e_h \wedge e_j, e_h \in \mathrm{en}_\varepsilon(\sigma_i) \wedge e_h \lessdot e_j \implies j < h \qquad (5.3)$$

$C \in \mathrm{C}(\varepsilon, \lessdot)$ iff $\exists \sigma \in \mathrm{T}(\varepsilon, \lessdot) \, . \, \bar{\sigma} = C$. Again $\mathrm{T}(\varepsilon, \lessdot) \subseteq \mathrm{T}(\varepsilon)$ and $\lessdot' \subseteq \lessdot$ implies $\mathrm{T}(\varepsilon, \lessdot) \subseteq \mathrm{T}(\varepsilon, \lessdot')$.

Lemma 5.4.1. *Let* $(\varepsilon, \lessdot)$ *and* $(\varepsilon, \lessdot')$ *be two PEBES with* $\lessdot' \subseteq \lessdot$. *Then:*

$$\mathrm{T}(\varepsilon, \lessdot) \subseteq \mathrm{T}(\varepsilon, \lessdot').$$

Proof. Straightforward from the definition of traces, (5.3), and $\lessdot' \subseteq \lessdot$. □

Similar to PBESs, we can remove a priority pair $e \lessdot e'$ or $e' \lessdot e$ between an event e and its cause e', because an event and its cause are never enabled together. Therefore e.g. the pair $e \lessdot b$ in Figure 5.3 is redundant because of $\{b\} \mapsto e$. Also a priority pair $e \lessdot e'$ between an event e' and its disabler e, i.e. for $e' \rightsquigarrow e$, does not affect the semantics, since e must follow e' anyway. Consider for example the events a and d in Figure 5.3. Because of $a \rightsquigarrow d$, a always pre-empts d and thus $d \lessdot a$ is redundant.

Theorem 5.4.2. *Let* $(\varepsilon, \lessdot)$ *be a PEBES and* $\lessdot' := \lessdot \setminus \{(e,e') \mid e' \rightsquigarrow e \vee (\exists X \subseteq E\,.\,(e \in X \wedge X \mapsto e') \vee (e' \in X \wedge X \mapsto e))\}$. *Then* $\mathrm{T}(\varepsilon, \lessdot) = \mathrm{T}(\varepsilon, \lessdot')$.

Proof. Similar to the proof of Theorem 5.3.4, where the second case is replaced by:

Case $e_j \rightsquigarrow e_h$**:** Because of the definition of traces, $\sigma = e_1, \ldots, e_n \in \mathrm{T}(\varepsilon)$ and $e_j, e_h \in \bar{\sigma}$ imply that $e_j \in \mathrm{en}_\varepsilon(\sigma_{j-1})$. Then $e_j \rightsquigarrow e_h \wedge e_j \in \mathrm{en}_\varepsilon(\sigma_{j-1}) \wedge e_h \in \bar{\sigma} \implies j < h$. □

Note that priority for events that are directly related by the bundle enabling relation is always redundant, regardless whether the cause has the higher priority or the effect does. On the other hand we can reduce pairs of events that are related by disabling only if an event has a higher priority than its disabler. Consider for example the PEBES $(\{e,e'\}, \{e \rightsquigarrow e'\}, \emptyset, \{e \lessdot e'\})$. The only traces of this PEBES are e and e', but if we remove the priority pair $e \lessdot e'$ we have the additional trace ee'. Similarly we cannot remove the $e \rightsquigarrow e'$ here, because this yields to the additional trace $e'e$.

Redundant priority pairs, according to Theorem 5.4.2, in Figure 5.3 are distinguished by thin double links. Note that after dropping such redundant pairs, the priority relation is not a partial order anymore.

Again limiting our attention to a specific configuration allows us to ignore some more priority pairs. In contrast to PBESs we can sometimes also ignore priority pairs that overlap with disabling. Consider for example $b \rightsquigarrow a$ and $a \rightsquigarrow d$ of the PEBES in Figure 5.3. The priority pair $d \lessdot b$ is redundant with respect to the configuration $\{a,b,d\}$ but not with respect to the configuration $\{b,d\}$. Note that again the direction of the priority pair is important in the case of indirect disabling but not in the case of indirect enabling. If we for instance replace $d \lessdot b$ in Figure 5.3 by $b \lessdot d$, then $\{a,b,d\}$ is not a configuration anymore and $b \lessdot d$ is not redundant in all remaining configurations containing b and d.

The cases in which priority pairs are redundant with respect to some configuration C are again well described by the precedence relation $\preceq_C$, i.e. we can identify redundant priority pairs easily from the poset of C.

The priority pair $h \lessdot b$ is obviously redundant in the case of $\boxed{b \longrightarrow e \longrightarrow h}$ but not in the case of $\boxed{b \quad f \longrightarrow h}$ and $d \lessdot b$ is obviously redundant in the case of $\boxed{b \longrightarrow a \rightarrow \ d}$ but not in the case of $\boxed{b \quad d}$. Let $\mathrm{T}(\varepsilon, \lessdot) \upharpoonright_C$ denote the set of traces over C, i.e. $\mathrm{T}(\varepsilon, \lessdot) \upharpoonright_C := \{\sigma \mid \sigma \in \mathrm{T}(\varepsilon, \lessdot) \wedge \bar{\sigma} = C\}$. Then for all configurations C we can ignore all priority pairs $e \lessdot e'$ such that $e' \preceq_C e$.

Theorem 5.4.3. *Let* $(\varepsilon, \lessdot)$ *be a PEBES,* $\langle C, \preceq_C\rangle \in \mathrm{P}(\varepsilon)$*, and let:*

$$\lessdot' := \lessdot \setminus \left\{(e, e') \in C \times C \mid e' \preceq_C e\right\}.$$

Then: $\mathrm{T}(\varepsilon, \lessdot) \upharpoonright_C = \mathrm{T}(\varepsilon, \lessdot') \upharpoonright_C$.

Proof. Note that $\mathrm{T}(\varepsilon, \lessdot) \upharpoonright_C \subseteq \mathrm{T}(\varepsilon, \lessdot)$ and
$\mathrm{T}(\varepsilon, \lessdot') \upharpoonright_C \subseteq \mathrm{T}(\varepsilon, \lessdot')$.

By Lemma 5.4.1, $\mathrm{T}(\varepsilon, \lessdot) \subseteq \mathrm{T}(\varepsilon, \lessdot')$ and thus also $\mathrm{T}(\varepsilon, \lessdot) \upharpoonright_C \subseteq \mathrm{T}(\varepsilon, \lessdot') \upharpoonright_C$.

To show $\mathrm{T}(\varepsilon, \lessdot') \upharpoonright_C \subseteq \mathrm{T}(\varepsilon, \lessdot) \upharpoonright_C$, assume a trace $\sigma = e_1, \ldots, e_n \in \mathrm{T}(\varepsilon, \lessdot') \upharpoonright_C$. We have to show that $\sigma \in \mathrm{T}(\varepsilon, \lessdot) \upharpoonright_C$, i.e. that $\forall e \in \bar{\sigma}\,.\, e \in C$, $\sigma \in \mathrm{T}(\varepsilon)$, and that σ satisfies Condition (5.3). $\forall e \in \bar{\sigma}\,.\, e \in C$ and $\sigma \in \mathrm{T}(\varepsilon)$ follows from $\sigma \in \mathrm{T}(\varepsilon, \lessdot') \upharpoonright_C$ by the definition of traces of PEBESs. σ satisfies Condition (5.3) if $\forall i < n\,.\, \forall e_j, e_h \in \bar{\sigma}\,.\, e_j \neq e_h \wedge e_j, e_h \in \mathrm{en}_\varepsilon(\sigma_i) \wedge e_h \lessdot e_j \implies j < h$. Let us fix $i < n$ and $e_j, e_h \in \bar{\sigma}$. Assume $e_j \neq e_h$, $e_j, e_h \in \mathrm{en}_\varepsilon(\sigma_i)$, and $e_h \lessdot e_j$. It remains to prove that $j < h$.

Because of the definition of $\lessdot'$, assumption $e_h \lessdot e_j$ implies that $e_h \lessdot' e_j$ or $e_j \preceq_C e_h$. In the first case $j < h$ follows, because of the definition of traces and (5.3), from $e_h \lessdot' e_j$, $e_j, e_h \in \bar{\sigma}$, and $\sigma \in \mathrm{T}(\varepsilon, \lessdot') \upharpoonright_C$. The other case, i.e. that $e_j \preceq_C e_h$ and $e_j \neq e_h$ implies $j < h$, was already proved in [47]. □

Consider once more the PEBES $(\varepsilon, \lessdot)$ of Figure 5.3 with respect to the configuration $\{b, e, h\}$. We have $\{b\} \mapsto e$, $\{e, f\} \mapsto h$, and $h \lessdot b$. As explained before we cannot drop the priority pair $h \lessdot b$, because of the trace $fhb \notin \mathrm{T}(\varepsilon, \lessdot)$. However with Theorem 5.4.3 we can ignore $h \lessdot b$ and also $h \lessdot e$ and $e \lessdot b$ for the semantics of $(\varepsilon, \lessdot)$ if we limit our attention to $\{b, e, h\}$, because $\mathrm{T}(\varepsilon, \lessdot) \upharpoonright_{\{b,e,h\}} = \mathrm{T}(\varepsilon, (\lessdot \setminus \{h \lessdot b, h \lessdot e, e \lessdot b\})) \upharpoonright_{\{b,e,h\}}$.

Similarly we can ignore $c \lessdot a$ if we limit our attention to the configuration $C = \{a, c, d\}$,
since $\mathrm{T}(\varepsilon, \lessdot) \upharpoonright_C = \mathrm{T}(\varepsilon, (\lessdot \setminus \{c \lessdot a\})) \upharpoonright_C$. Note that here the precedence pair $a \preceq_C c$ that allows us to ignore $c \lessdot a$ results from the correlation between

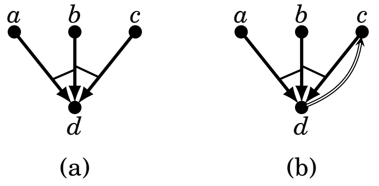


Figure 5.4.: *A Dual ES without priority in* (a) *and with priority in* (b).

a disabling pair $a \rightsquigarrow d$ and an enabling pair $\{d\} \mapsto c$. Thus with Theorem 5.4.3 we can ignore even priority pairs that are redundant in specific situations because of combining enabling and disabling.

This combination prohibits us on the other hand from ignoring priority of the opposite direction, the direction which is compatible with the precedence direction. That is possible only with precedence resulted from enabling purely as it is the case in Theorem 5.3.5 for PBESs. For instance, suppose that $b \lessdot h$ for the structure in Figure 5.3 then we can ignore this priority pair in a configuration $\{b, e, h\}$. That is not formulated in the Theorem 5.4.3 above, since $\preceq_C$ abstracts from the relation between events. While in contrast to EBESs, the conflict relation is symmetric in Bundle ESs, and precedence results only from enabling. Thus, in contrast to PBESs, we do not have minimality of priority ignorance in PEBESs.

5.5. Priority in Dual Event Structures

We add priority to DESs in the same way as before. A *prioritized Dual ES (PDES)* is the tuple $(\Delta, \lessdot)$, where Δ is a DES and $\lessdot$ is the acyclic priority relation. Also the definitions of traces, $\mathrm{T}(\Delta, \lessdot)$, configurations, and $\mathrm{C}(\Delta, \lessdot)$ are adapted similar as in the section before.

Since the conflict relation provided here is the same as in Bundle ESs, we can remove redundant priority pairs that overlap with the conflict relation as described in Theorem 5.3.4, i.e. whenever there is $e\#e'$ or $e'\#e$ then $e \lessdot e'$ is redundant and can be removed. The situation for enabling is different because of the missing stability condition. The priority pair $c \lessdot d$ in the PDES in Figure 5.4 is not redundant, because it removes some traces. The reason is that c is not anymore a necessary cause for d, since d can be enabled by b even if c occurs in the same trace. So at the structure level of

PDESs we cannot in general remove priority pairs because of overlapping with the enabling relation.

In the case of PBESs and PEBESs partial orders help to identify redundant priority pairs at the configuration level. Unfortunately, we cannot do the same here. Let us consider the configuration $\{a,b,c,d\}$, and consider liberal causality. Indeed applying (5.3.5) on the poset $\boxed{\begin{smallmatrix}a\\b\end{smallmatrix} \rightrightarrows d \leftarrow c}$ and the priority $c \lessdot d$ yields the sequence $abcd$ which is not a trace. On the other hand, considering bundle-satisfaction causality, the poset $\boxed{a \ \begin{smallmatrix}b\\c\end{smallmatrix} \rightrightarrows d}$ with the same priority yields the same sequence $abcd$ again. The same will be for minimal causality and a poset like $\boxed{\begin{smallmatrix}a\\c\end{smallmatrix} \rightrightarrows d \ \ b}$.

In fact none of the mentioned kinds of posets can be used alone without the priority, and thus ignorance is not possible with causal ambiguity w.r.t. a single poset for a configuration. Even when $c \lessdot d$ seems to yield preemption in the poset $\boxed{a \ \ b \longrightarrow d \ \ c}$, one can have the linearization $acbd$ which is a trace, and priority turns out to be redundant in this very trace (but not in the whole poset). The reason is that partial orders in DESs do not necessarily represent necessary causes.

5.6. Evaluation

We have added priority to different Event Structures: Prime ESs as a simple model with conjunctive causality, Bundle ESs with disjunctive causality, Extended Bundle ESs with asymmetric conflict, and Dual ESs with causal ambiguity. In all cases, priority led to trace filtering and limited concurrency and non-determinism. We then analyzed the relationship between the new priority relation and the other relations of the ESs. Since priority has an effect only if the related events are enabled together, overlappings between the priority relation and the other relations of the ESs sometimes lead to redundant priority pairs.

In PPESs, PBESs, and PEBESs priority is redundant between events that are related directly by causality. Moreover in all considered ESs priority is redundant between events that are related directly by the conflict relation. But in the case of PEBESs the conflict relation implements asymmetric conflicts. Hence in contrast to the other ESs we have to take the direction of the disabling relation into account.

The main difference between redundancy of priority in PPESs and the other three models is due to events that are indirectly related by causality. In PPESs causality is a transitive relation, i.e. all pairs which are indirectly related by causality are directly related by causality as well. The enabling relation of the other models is not transitive. Thus priority pairs between events that are only indirectly related by enabling are not necessarily redundant. Unfortunately and unlike PPESs, this means that we cannot ensure after removing the redundant priority pairs that the remaining priority pairs necessarily lead to pre-emption. So the other models hold more ambiguity to a modeler.

Instead we show that if we limit our attention to a specific configuration C, a priority pair $e \lessdot e'$ is redundant if $e' \preceq_C e \vee e \preceq_C e'$ for PBESs, or if $e' \preceq_C e$ for PEBESs. This allows us to ignore—for the semantics with respect to specific configurations— additional priority pairs between events indirectly related by enabling for PBESs; and by enabling, disabling, or even by combinations of enabling and disabling for PEBESs. In the case of PBESs we obtain a minimality result this way.

Unfortunately in PDESs even priority pairs between events that are directly related by causality are not necessarily redundant. So from a modeler's perspective, priority in DESs hold the biggest ambiguity among all the studied ESs. In other words, one cannot figure out the role priority plays at design time or structure level, and whether this priority yields pre-emption or not. Even at the configuration level, that is not possible in general due to causal ambiguity.

Thus the main contributions of this chapter are: 1) We add priority as a binary acyclic relation on events to ESs. 2) We show that the relation between priority and other event relations of an ES can lead to redundant priority pairs, i.e. to priority pairs that do never (or at least for some configurations not) affect the behavior of the ES. 3) Then we show how to completely remove such pairs in PPESs and that this is in general not possible in ESs with a more complex causality model like PBESs, PEBESs, or PDESs. 4) Instead we show how to identify all priority pairs that are redundant with respect to configurations in PBESs and that the situation in PEBESs and DESs is different. 5) We show how to identify (some of the) redundant priority pairs at the level of configurations in PEBESs and 6) that again this is in general not possible in the same way for PDESs.

After dropping or ignoring redundant priority pairs as described above, the minimum potential for overlapping between priority and causality can be found in PPESs, while the maximum is in PDESs. In PPESs all remaining priority pairs indeed affect the semantics, i.e. exclude traces. In PBESs the same holds with respect to specific configurations. In PEBESs after dropping the redundant priority pairs the disabling relation has no overlapping with only priority directed in the opposite direction.

In Section 5.3 we showed that adding priority complicates the definition of families of posets to capture the semantics of prioritized ESs. We observe that because of priority, a single configuration may require several posets to describe its semantics. The same already applies for DESs because of the causal ambiguity. However note that priority does not lead to causal ambiguity. Thus, we can define the semantics of prioritized ESs by families of posets if we do not insist on the requirement that there is exactly one poset for each configuration. We leave the definition of such families of posets for future work. Such families of posets for prioritized ESs may also help to identify and ignore redundant priority pairs in the case of PEBESs and PDESs. Another interesting topic for further research is to analyze how priority influences the expressive power of ESs.

Application in Healthcare: Consider the following special example of priority taken from Charité Berlin and DHZB. *A 65 sear old patient is admitted to the oncology department of Charité with symptoms of a cancer. Within the staging process there it is diagnosed that the patient has a malignant cancer of her kidneys. Furthermore the patient has a cardiac valve insufficiency. The treatment of both problems could not happen concurrently. Thus the oncologist had to invite a cardiac specialist to decide which problem has a higher priority. The team decided that the cardiac problem was more severe, so that the oncological process was interrupted and the patient was transferred to the DHZB. There the patient had a heart valve surgery and admitted to the intensive care unit, where the warming up and wake up process were begun. At the end the discharge was done to the oncology again to proceed with the oncological pathway of treating the cancer. The cancer was treated with an adjuvant chemotherapy and afterwards with a surgical resection regarding the cancer.*

This is an example not only of priority, but also of Dynamic Coalitions and their prioritized activities with evolution and adaptation. Priority in this example can be modeled straightforward using our Prioritized ESs as

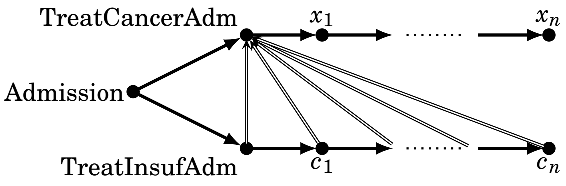
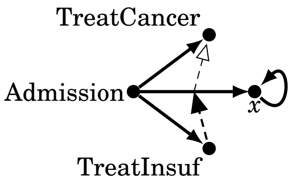


Figure 5.5.: *A PPES modeling priority between cancer treatment and cardiac-insufficiency treatment for a 65-year patient admitted to Charité.*

shown in Figure 5.5. Note that the whole treatment path that is causally dependant on the cardiac problem has a higher priority than any event of the cancer treatment path or than the admission event TreatCancerAdm of cancer treatment. In other words it not enough to have a single pair of priority in the structure. This is some kind of heredity in priority similar to conflict heredity of Prime ESs (cf. Section 2.2).



Evolution can be applied here to model the transition from the first WF that preceded the cooperation with the cardiac specialist. This WF can be modeled such that no priority yet exists; rather with an interleaving between the two treatments that prohibits them from running concurrently. For that purpose, a DCES (cf. Section 3.5), with a dropper and an adder for the same causal dependency of an impossible target, can be used as illustrated aside. This proves the idea of Evolving Structures to combine different kinds of ESs with an evolution relation, even prioritized ESs and non-prioritized ESs, as long as they define the notion of configurations, or more specifically traces in this case.

Further Enhancements: The notion of heredity of priority mentioned before highlights the need for defining action refinement with priority. In this case each treatment can be modeled as a single event with a priority pair from the cardiac treatment to the cancer one. The two events can be refined in a different level of abstraction to sub-workflows, where priority needs to be inherited between the events of the first treatment and the events of the second one such that consistency between the two levels w.r.t.

system runs holds, as illustrated in Section 2.10.5. It is worthy to note that in this case semantics of PESs should be defined in terms of posets instead of traces. The reason [73] is that interleaving semantics does not reflect real independence between events, which is needed when refining the two events to determine independence between their refinement events. For instance, assume a and b are two independent events, and are refined to a_1, a_2 and b_1, b_2 respectively. Then if the system runs at the refinement level must match ones at the abstract level, a_1,a_2,b_1,b_2 and b_1,b_2,a_1,a_2 would match a,b and b,a respectively, while a_1,b_1,a_2,b_2 which is a system run at the refinement level due to independence between $\{a_1,a_2,b_1,b_2\}$ inherited from the abstract level would match no system run at the abstract level.

6. Summary and Future Work

6.1. Summary and Conclusions

Dynamic Coalitions (DCs) denote a temporary collaboration between different entities to achieve a common goal. What gives DCs their dynamic nature is dynamic membership, where members can join and leave after the coalition is set [46, 21]. This is considered a key feature of DCs, which distinguishes DCs from classical coalitions. This thesis studies workflows in *Dynamic Coalitions*, analyzes their features, highlights their unique characteristics and similarities to other workflows, and investigates their relation with *Dynamic Membership*. To this end, we use the formal model of *Event Structures (ESs)* and extend it to faithfully model scenarios taken as use cases from healthcare. ESs allow for workflow modeling in general, and for modeling *Dynamic Membership* in DCs through capturing the *join* and *leave* events of members. Besides ESs are suitable for our case of human-based collaboration as we do not address problems such as recursion where other formalisms might be more suitable [43]. Furthermore, when we apply *Dynamic Causality* and *Evolution*, the basic relations of ESs, namely causality and conflict, help in understanding such extensions.

We first extend ESs with *Dynamic Causality* to address the dynamic nature of DCs. *Dynamic Causality* allows some events to change the causal dependencies of other events in a structure. This helps us to understand changes in a formal way, and isolate them from other concerns e.g. goal orientation. We study the expressive power of the resulting ESs and show that they contribute only to a specific kind of changes in workflows, namely pre-planned changes. Second, we present *Evolving Structures* in order to support ad-hoc and unforeseen changes in workflows, as required by the use cases taken from healthcare. *Evolving Structures* connect different ESs with an *evolution* relation which allows for changing an ES during a system run, including adding and dropping events as well as changing flow and conflict of the events. We consider different approaches to model evolution and study their relation. Furthermore, we show why the history of a workflow should be preserved in our case of evolution in DCs, and allow for extracting changes from an evolution to support *Process Learning*. Third, to capture the goals of DCs, we equip *Evolving Structures* with constraints concerning the reachability of a set of events that represents a goal. This helps us in guiding an evolution by limiting arbitrary changes. The former extensions of *Dynamic Causality* and *Evolving Structures* together with *Goal-Orientation* allow the examination of changes and evolutions caused by members, and the examination of members' contributions to goal satisfaction, through their *join* and *leave* events. Finally, we highlight many modeling features posed as requirements by our DC use case domain, namely the healthcare, which are independent from the nature of DCs, e.g. timing. We examine the literature of ESs for supporting such features, and show that the notion of *Priority* is missing in ESs. To this end, we add *Priority* to various kinds of ESs from the literature. Furthermore, we study the relation between priority on one side, and conjunctive causality, disjunctive causality, causal ambiguity and various kinds of conflict on the other side.

Comparing to *Adaptive Workflows*, which are concerned with workflows that evolve as a response to changes in their business environment or to exceptions, the thesis shows that DC workflows are not only goal-oriented but also adaptive. Besides, the thesis adds one extra reason for evolution and changes in DC workflows to the ones in *Adaptive Workflows*, namely the join of new members, which is missing in static coalitions. Finally the thesis contributes to bridging the gap in modelling between theory and domain experts by supporting step-by-step modelling applied regularly in healthcare and other domains.

To summarize, we cover the topic of DC workflows from different perspectives needed to faithfully model such workflows w.r.t. our use cases, and provide a mature study about the nature and characteristics of DC workflows. We use ESs as a formalism, which proves to provide a fine understanding of Dynamic Causality and changes in a workflow. ESs has allowed us to give precise definitions of how to dynamize a structure and of what to preserve vs. what to evolve in a structure.

Conclusions: The workflow of a DC (based on human collaboration) is a special case of both Adaptive Workflows and Goal-Oriented Workflows.

Adaptations in DC workflow might happen due to changes in the DC environment, to exceptions, or due to contributions new members bring in upon their join. A study concerning DCs should address dynamic membership, and examine its influence on the problem under study. Besides, formalisms like Event Structures—and similar ones—need to support flexibility so that they are able to model DC workflows. This might include the ability to change particular ingredients of the formalism during runtime, or even the whole structure.

6.2. Contributions in Detail

Here we summarize the contributions of this thesis from different perspectives. The first section covers contributions in Dynamic Coalitions, while the rest cover contributions in Event Structures.

6.2.1. The Adaptive and Goal-Oriented Nature of Dynamic-Coalition Workflows

We prove that workflows of Dynamic Coalitions are a special kind of Adaptive Workflows, where the join of a new member in a DC might be a trigger for evolution of the DC workflow. By that we show the unique features of DC workflows, and how they are affected by membership; this was an open question in the literature of DCs [16, 20, 68, 45]. Accordingly, we show that a formalism used to model DC workflows needs to support evolution and changes during system runs, and to show the influence of members on the workflow, which was not the case with the formalisms used to model DC workflows in the literature, e.g. VDM [16] and RAISE [57]. To that end, we apply Adaptive-Workflows state of the art to model DC workflows, and apply goal orientation to model goals of DCs. We cover both pre-planned and unforeseen changes in workflows of DCs. Additionally, since DCs are goal orienegtd, we show that DC evolution needs to respect goal satisfaction at the end.

6.2.2. Dynamic Causality in Event Structures

We add Dynamic Causality to Event Structures and provide the ability to change causality upon occurrence of some events. Furthermore, we study the expressive power of the new model. We show that in its simple forms, i.e. Growing and Shrinking causality, dynamic causality can be modelled by Conditional and Disjunctive Causality respectively, while in its most complex form, we show that it is not possible always to model dynamic causality with static relations. Additionally, we discuss concurrency and independence of different modifiers. We show how dynamic causality can partially model dynamicity of other relations e.g. conflict through the ability to model resolvable (i.e. shrinking) conflict, and show how dynamic causality is able to model disabling as well as mutual conflict. Finally, we equipped our extension with a graphical notation.

6.2.3. Evolution in Event Structures

We introduce Evolving Event Structures and connect different ESs through an evolution relation which captures the conditions of replacing a structure by another after a system run. To this end, we address transitions between structures rather than changes that might occur to one structure, and provide a way to infer the changes. Furthermore, we defin a mechanism based on DCES to infer changes between the structures; this connects the two contributions of Evolving ESs and Dynamic Causality ESs. Additionally, we constrain the course of evolution by the reachability of a fixed set of events representing a high-level goal. We define chains of evolution steps through the introduction of *evolutionary sequences*. Besides, we discuss different approaches to model an evolution step from one structure to another, and the relation in-between the different approaches.

6.2.4. Priority in Event Structures

We add priority to different kinds of ESs in the literature. We study the relation between priority on one hand and the disjunctive causality of Bundle ESs, asymmetric conflict of Extended Bundle ESs, and causal-ambiguity of Dual ESs on the other hand. To this end, we show that priority would be redundant against causality of PESs, while it is ambiguous to decide whether it is redundant in other kinds of ESs. Additionally, we study the relation between priority and the precedence relation at the configuration level through posets. Finally, we equipped our extension with a graphical notation.

6.3. Future Work

This thesis highlights a number of topics as future work. Some of these topics, e.g. the first two, have not been achieved due to time limitations, while others are enhancements that would enrich the framework and allow for more modeling features and capabilities.

Time Constraints on Evolution: One further research is to define time constraints not only on event occurrences, but also on evolution steps. For instance, a heart failure occurred that called for adapting the plan within a maximum time or timeout. Formalizing such constraints on evolution relation would allow for more properties to be checked against the evolution of a WF, e.g. *time-illness* [43].

Evolution Guiding towards Goal Satisfaction: Another possible research is about guiding evolution of a given WF in a specific domain. Our work implements goal satisfaction in a declarative way by forcing some constraints through the notion of Evolution (cf. Section 4.6). A possible alternative would be to develop methods, as in [23], suggesting which evolution steps to make such that the goal is reachable again in case case of exceptions. Such a study would be dependent on the domain itself and would require domain knowledge.

Evolution and Property-Based Modeling of Goals: In Event Structures, the state of a system is decided based on the history of events that took place. For instance, the property that the patient is no more given any severe medication can be determined from the occurrence of the event $StopSevMed$. An alternative is to use *properties*, especially for goal modeling. For example the final goal of discharging the patient can be modeled as a property stating that *the patient is discharged*. Besides, as illustrated in Section 4.6, such a goal would be refined to sub-goals modeled as properties: the ventilation tube is disconnected, the patient is awake and no more severe medication is given to her (with AND links in-between). Then an exception blocking one of these properties to hold, e.g. heart failure regarding patient-is-awake property, calls for adapting the workflow. Property-based modeling of goals, along with goal refinement to sub-goals, are already defined in [29, 28, 78] as illustrated in Section 4.6. Nevertheless since our work combines workflow adaptation with goal orientation, the concepts of goal orientation presented in Section 4.6 should be adapted w.r.t. property-based modeling as a future work.

Internalization with Chains of Triggers: In Section 4.4, an internalization shows the history of a model with the transition from an old ES to a new one represented after the occurrence of the trigger event v. Up to know the internalization extracts changes from a single evolution step, i.e. from a pair of ESs, and shows the transition between them. This can be extended such that changes can be extracted from an evolutionary sequence. Accordingly, a series of trigger events v_1, v_2, . . . can be generated and the relation between these events can be defined. This helps in understanding the relation between evolution steps that take place on a running workflow, by knowing their dependencies and conflicts, which in turn allow for analyzing properties of the evolution and learning from possible evolution patterns [18].

Higher-Order and Set-Based Dynamics in Causality: On the other hand, dynamicity highlights new topics to work on, and motivates to investigate further complexity of it, by considering higher-order dynamics. This helps in supporting process learning of Section 4.4 to infer changes, as well as to find a common upper bound, w.r.t. expressiveness, for the ESs that are incomparable to DCESs as shown in Figure 2.9. Besides, set-based dynamic causality can be used to simulate dynamicity of other relations, e.g. growing conflict. As an example, consider a conflict $a\#b$ added by c that it can be modeled by set-based mutual disabling as $\{c,a\}$ disables b and $\{c,b\}$ disables a.

Action Refinement with Dynamic Causality: It would be interesting to investigate action refinement in Dynamic Causality ESs. A target event in a modification might be refined to a complete structure where the causality added or dropped from that target is to be added or dropped respectively from the events of its refinement in the lower abstraction level. In other words, the modification is to be inherited in similarly to the way causal predecessors are inherited in [74]. The challenge would be then to refine the modifier itself. For instance, if a modifier, that adds a causal predecessor to a target, is refined into a set of events at the lower level, the question is then which of these events would be the adder at the lower level. Indeed this would lead the work back to Set-Based Dynamic Causality, and the resulting structure of refinement would be a Set-Based DCES. It is interesting to investigate the relation between the structures of the two levels and to check the concurrency consistence between both w.r.t. the transition relation.

Integrating Researches in SOAMED regarding DCs: As mentioned in the related work, N. Sarrouh, a former member of the *SOAMED Research Training Group*, worked in his PhD thesis [68] on building a modeling framework for privacy-aware DCs. His work was based on the *Abstract State Machine (ASM)* formalism which allowed for a structural description of dynamic coalitions by means of ASM programs, thereby enabling simulation, and verification possibilities. To this end he used the *Core ASM* implementation of ASM which allowed for defining states and their updates. Besides, Core ASM allowed for the execution of ASM Programs, being a number of sequential or concurrent updates, thereby allowing for tests and simulation. A possible future work is to integrate our work with Sarrouh's work, to build a larger framework that benefits from both formalism's capabilities. Such an integration can serve as a bridge between the two approaches. For example, starting from a structural ASM definition of a dynamic coalition a corresponding ES maybe created, which defines how and in what order events may occur. In a next step, these ES relations may be enhanced, for instance, adding priorities to some events (cf. Chapter 5). With the resulting ES, all possible scenarios of the dynamic coalition at hand may be derived as system runs by means of ES techniques and by tool support. The resulting system runs could be translated into concrete ASM programs, in order to execute, simulate or test properties like termination or privacy properties. Besides, the sequentiality and parallelism in the ASM program might be obtained from the generated system runs in their partially ordered form.

List of Abbreviations

BES	Bundle Event Structure
CES	Conditional-Causality Event Structure
CICU	Cardiac Intensive Care Unit
DC	Dynamic Coalition
DCES	Dynamic-Causality Event Structure
DES	Dual Event Structure
DHZB	Deutsches Herzzentrum Berlin
EBDC	Extended Bundle subclass of DCES
EBES	Extended Bundle Event Structure
ES	Event Structure
GES	Growing-Causality Event Structure
PBES	Prioritized Bundle Event Structure
PDES	Prioritized Dual Event Structure
PEBES	Prioritized Extended Bundle Event Structure
PES	Prime Event Structure
PPES	Prioritized Prime Event Structure
RCES	Event Structure for Resolvable Conflict
SES	Shrinking-Causality Event Structure
SSDC	Single State subclass of DCES
WF	Workflow

List of Figures

Bibliography

[1] Michael Adams, Arthur HM ter Hofstede, Wil MP van der Aalst, and David Edmond. Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803 of *Lecture Notes in Computer Science*, pages 95–112. Springer Berlin Heidelberg, 2007.

[2] Youssef Arbach, David Karcher, Kirstin Peters, and Uwe Nestmann. Dynamic Causality in Event Structures. In *Proceedings of 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*, pages 83–97. Springer International Publishing, 2015.

[3] Youssef Arbach, Kirstin Peters, and Uwe Nestmann. Adding Priority to Event Structures. In *Proceedings of Combined 20th International Workshop on Expressiveness in Concurrency and 10th Workshop on Structural Operational Semantics, EXPRESS/SOS 2013, Buenos Aires, Argentina, 26th August, 2013*, volume 120 of *Electronic Proceedings in Theoretical Computer Science*, pages 17–31, 2013.

[4] Sebastian Bab and Nadim Sarrouh. Towards a Formal Model of Privacy-Sensitive Dynamic Coalitions. In Proceedings of Third Workshop on *Formal Aspects of Virtual Organisations,* Sao Paolo, Brazil, 18th October 2011, volume 83 of *Electronic Proceedings in Theoretical Computer Science*, pages 10–21. Open Publishing Association, 2012.

[5] Jos CM Baeten, Twan Basten, and MA Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2010.

[6] Jos CM Baeten, Jan A Bergstra, and Jan Willem Klop. Syntax and Defining Equations for an Interrupt Mechanism in Process Algebra. *Fundamenta Informaticae*, 9:127–167, 1986.

[7] Paolo Baldan, Nadia Busi, Andrea Corradini, and G Michele Pinna. Domain and Event Structure Semantics for Petri Nets with Read and Inhibitor Arcs. *Theoretical Computer Science*, 323(1):129–189, 2004.

[8] Paolo Baldan, Andrea Corradini, and Ugo Montanari. Contextual Petri Nets, Asymmetric Event Structures, and Processes. *Information and Computation*, 171(1):1–49, 2001.

[9] Lina Barakat, Simon Miles, and Michael Luck. Efficient adaptive QoS-based service selection. *Service Oriented Computing and Applications*, 8(4):261–276, 2014.

[10] Falko Bause. On the analysis of Petri nets with static priorities. *Acta Informatica*, 33:669–685, 1996.

[11] Falko Bause. Analysis of Petri nets with a dynamic priority method. In *Proceedings of Application and Theory of Petri Nets*, Lecture Notes in Computer Science, pages 215–234. Springer, 1997.

[12] Rakesh Bobba, Serban Gavrila, Virgil Gligor, Himanshu Khurana, and Radostina Koleva. Administering Access Control in Dynamic Coalitions. In *Proceedings of the 19th conference on Large Installation System Administration Conference - Volume 19*, LISA '05, pages 23–23, Berkeley, CA, USA, 2005. USENIX Association.

[13] Gérard Boudol and Ilaria Castellani. Permutation of transitions: An event structure semantics for CCS and SCCS. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 411–427. Springer Berlin Heidelberg, 1989.

[14] Gérard Boudol and Ilaria Castellani. Flow Models of Distributed Computations: Event Structures and Nets. Technical report, INRIA, 1991.

[15] Gérard Boudol and Ilaria Castellani. Flow Models of Distributed Computations: Three Equivalent Semantics for CCS. *Information and Computation*, 114(2):247–314, 1994.

[16] JW Bryans, JS Fitzgerald, CB Jones, and I Mozolevsky. Dimensions of Dynamic Coalitions. Technical report, School of Computing Science, University of Newcastle upon Tyne, 2006.

[17] Antonio Bucchiarone, Raman Kazhamiakin, Annapaola Marconi, and Marco Pistore. Adaptivity in Dynamic Service-based Systems. In *Proceedings of the First International Workshop on European Software Services and Systems Research: Results and Challenges*, S-Cube '12, pages 36–37, Piscataway, NJ, USA, 2012. IEEE Press.

[18] Antonio Bucchiarone, Annapaola Marconi, Marco Pistore, and Adina Sirbu. A Context-Aware Framework for Business Processes Evolution. In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International*, pages 146–154, August 2011.

[19] Antonio Bucchiarone, Marco Pistore, Heorhi Raik, and Raman Kazhamiakin. Adaptation of Service-based Business Processes by Context-Aware Replanning. In *2011 IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2011, Irvine, CA, USA, December 12-14, 2011*, pages 1–8, 2011.

[20] Paul A Buhler, José M Vidal, and Harko Verhagen. Adaptive Workflow = Web Services + Agents. In *Proceedings of the International Conference on Web Services*, volume 3, pages 131–137. CSREA Press, 2003.

[21] Luis M Camarinha-Matos, Ivan Silveri, Hamideh Afsarmanesh, and Ana Ines Oliveira. Towards a Framework for Creation of Dynamic Virtual Organizations. In *Collaborative Networks and Their Breeding Environments*, volume 186 of *IFIP – The International Federation for Information Processing*, pages 69–80. Springer US, 2005.

[22] Juanito Camilleri and Glynn Winskel. CCS with Priority Choice. *Information and Computation*, 116(1):26–37, 1995.

[23] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. QoS-Aware Replanning of Composite Web Services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '05, pages 121–129, Washington, DC, USA, 2005. IEEE Computer Society.

[24] Ross Casley, Roger F Crew, José Meseguer, and Vaughan Pratt. Temporal Structures. *Mathematical Structures in Computer Science*, 1(02):179–213, 1991.

[25] Rance Cleaveland, Gerald Lüttgen, and V Natarajan. Priority in Process Algebras. ICASE report, NASA, 1999.

[26] Massimo Coppola, Yvon Jégou, Brian Matthews, Christine Morin, Luis Pablo Prieto, Oscar David Sánchez, Erica Y Yang, and Haiyan Yu. Virtual Organization Support within a Grid-Wide Operating System. *Internet Computing, IEEE*, 12(2):20–28, March 2008.

[27] Ruggero Costantini and Arend Rensink. Abstraction and Refinement in Configuration Structures. Hildesheimer Informatik-Bericht 18/92, Institut für Informatik, University of Hildesheim, Germany, 1992.

[28] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.

[29] Robert Darimont and Axel van Lamsweerde. Formal Refinement Patterns for Goal-driven Requirements Elaboration. *SIGSOFT Software Engineering Notes*, 21(6):179–190, October 1996.

[30] Robert Darimont and Axel van Lamsweerde. Formal Refinement Patterns for Goal-driven Requirements Elaboration. In *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '96, pages 179–190, New York, NY, USA, 1996. ACM.

[31] Philippe Darondeau and Pierpaolo Degano. Event Structures, Causal Trees, and Refinements. In *Mathematical Foundations of Computer Science 1990*, volume 452 of *Lecture Notes in Computer Science*, pages 239–245. Springer Berlin Heidelberg, 1990.

[32] Philippe Darondeau and Pierpaolo Degano. Refinement of actions in event structures and causal trees. *Theoretical Computer Science*, 118(1):21–48, 1993.

[33] Pallab Dasgupta, Jatindra Kumar Deka, and Partha Pratim Chakrabarti. Model checking on timed-event structures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(5):601–611, May 2000.

[34] Edmund H Durfee, Victor R Lesser, and Daniel D Corkill. Trends in Cooperative Distributed Problem Solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, March 1989.

[35] Clarence Ellis, Karim Keddara, and Grzegorz Rozenberg. Dynamic Change Within Workflow Systems. In *Proceedings of Conference on Organizational Computing Systems*, COCS '95, pages 10–21, New York, NY, USA, 1995. ACM.

[36] Susanna S Epp. *Discrete Mathematics With Applications, Third Edition*. Mathematics Series. Brooks/Cole-Thomson Learning, 2004.

[37] Andreas Glausch and Wolfgang Reisig. Distributed Abstract State Machines and Their Expressive Power. Technical Report 196, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, Institut für Informatik, 2006.

[38] Martin Glinz. On non-functional requirements. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 21–26, October 2007.

[39] Rachid Guerraoui. Indulgent algorithms (preliminary version). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, pages 289–297, New York, NY, USA, 2000. ACM.

[40] PW Hoogers, HCM Kleijn, and PS Thiagarajan. A Trace Semantics for Petri Nets. *Information and Computation*, 117(1):98–114, 1995.

[41] PW Hoogers, HCM Kleijn, and PS Thiagarajan. An event structure semantics for general Petri nets. *Theoretical Computer Science*, 153(1–2):129–170, 1996.

[42] Ryszard Janicki and Maciej Koutny. Semantics of Inhibitor Nets. *Information and Computation*, 123(1):1–16, 1995.

[43] Joost-Pieter Katoen. *Quantitative and Qualitative Extensions of Event Structures*. PhD thesis, University of Twente, 1996.

[44] Joost-Pieter Katoen, Rom Langerak, Ed Brinksma, Diego Latella, and Tommaso Bolognesi. A Consistent Causality-Based View on a Timed Process Algebra Including Urgent Interactions. *Formal Methods in System Design*, 12(2):189–216, March 1998.

[45] Zaheer Khan, Savo Glisic, Luiz A DaSilva, and Janne J Lehtomäki. Modeling the Dynamics of Coalition Formation Games for Cooperative Spectrum Sharing in an Interference Channel. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(1):17–30, 2011.

[46] Hristo Koshutanski and Antonio Mana. Interoperable Semantic Access Control for Highly Dynamic Coalitions. *Security and Communication Networks*, 3(6):565–594, 2010.

[47] Rom Langerak. *Transformations and Semantics for LOTOS*. PhD thesis, Universiteit Twente, 1992.

[48] Rom Langerak, Ed Brinksma, and Joost-Pieter Katoen. Causal ambiguity and partial orders in event structures. In *Proceedings of CONCUR*, Lecture Notes in Computer Science, pages 317–331. Springer, 1997.

[49] Rom Langerak, Ed Brinksma, and Joost-Pieter Katoen. Causal ambiguity and partial orders in event structures. Technical report, University of Twente, Centre for Telematics and Information Technology, 1997.

[50] Dai Tri Man Le. On Three Alternative Characterizations of Combined Traces. *Fundamenta Informaticae*, 113(3):265–293, 2011.

[51] Gerald Lüttgen. *Pre-emptive Modeling of Concurrent and Distributed Systems*. PhD thesis, University of Passau, 1998.

[52] Tomasz Michalak, Jacek Sroka, Talal Rahwan, Michael Wooldridge, Peter McBurney, and Nicholas R. Jennings. A Distributed Algorithm for Anytime Coalition Structure Generation. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*, AAMAS '10, pages 1007–1014, Richland, SC,

2010. International Foundation for Autonomous Agents and Multiagent Systems.

[53] Igor Mozolevsky and John S Fitzgerald. Common Representation of Information Flows for Dynamic Coalitions. In *Proceedings of Second Workshop on Formal Aspects of Virtual Organisations, FAVO 2009, Eindhoven, The Netherlands, 3rd November 2009*, volume 16 of *Electronic Proceedings in Theoretical Computer Science*, pages 15–25. Open Publishing Association, 2010.

[54] Raghava Rao Mukkamala. *A formal model for declarative workflows: dynamic condition response graphs*. PhD thesis, IT University of Copenhagen, June 2012.

[55] Raghava Rao Mukkamala, Thomas Hildebrandt, and Tijs Slaats. Towards Trustworthy Adaptive Case Management with Dynamic Condition Response Graphs. In *Enterprise Distributed Object Computing Conference (EDOC), 2013 17th IEEE International*, pages 127–136, September 2013.

[56] Manfred Nagl. *Softwaretechnik Mit Ada 95: Entwicklung Großer Systeme*. Vieweg+Teubner Verlag, May 2003.

[57] Mohammad Reza Nami, Mohsen Sharifi, and Abbas Malekpour. A Preliminary Formal Specification of Virtual Organization Creation with RAISE Specification Language. In *Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management & Applications*, SERA '07, pages 227–232, Washington, DC, USA, 2007. IEEE Computer Society.

[58] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri Nets, Event Structures and Domains, Part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

[59] Talal Rahwan, Sarvapali D Ramchurn, Nicholas R Jennings, and Andrea Giovannucci. An Anytime Algorithm for Optimal Coalition Structure Generation. *Journal of Artificial Intelligence Research*, 34(2):521, 2009.

[60] Manfred Reichert and Peter Dadam. ADEPTflex–Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[61] Manfred Reichert, Peter Dadam, and Thomas Bauer. Dealing with forward and backward jumps in workflow management systems. *Software and Systems Modeling*, 2(1):37–58, 2003.

[62] Arend Rensink. Posets for Configurations! In *Proceedings of CONCUR*, volume 630 of *Lecture Notes in Computer Science*, pages 269–285. Springer Berlin Heidelberg, 1992.

[63] Arend Rensink. *Models and Methods for Action Refinement*. PhD thesis, University of Twente, 1993.

[64] Arend Rensink. An Event-Based SOS for a Language with Refinement. In *Structures in Concurrency Theory*, Workshops in Computing, pages 294–309, Berlin Germany, 1995. Springer Verlag.

[65] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. Correctness criteria for dynamic changes in workflow systems – a survey. *Data & Knowledge Engineering*, 50(1):9–34, 2004. Advances in business process management.

[66] Stefanie Rinderle, Barbara Weber, Manfred Reichert, and Werner Wild. Integrating Process Learning and Process Evolution – A Semantics Based Approach. In *Business Process Management*, volume 3649 of *Lecture Notes in Computer Science*, pages 252–267. Springer Berlin Heidelberg, 2005.

[67] Walid Saad, Zhu Han, Mérouane Debbah, Are Hjorungnes, and Tamer Basar. Coalitional game theory for communication networks. *Signal Processing Magazine, IEEE*, 26(5):77–97, September 2009.

[68] Nadim Sarrouh. *Privacy-Aware Dynamic Coalitions : A Formal Framework*. PhD thesis, Technische Universität Berlin, 2014.

[69] Zongmin Shang. Research on Adaptation of Service-based Business Processes. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 12(1):442–449, January 2014.

[70] A Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994.

[71] Ping Sun and Changjun Jiang. Analysis of workflow dynamic changes based on Petri net. *Information and Software Technology*, 51(2):284–292, 2009.

[72] Hoang Chi Thanh. Semi-traces and Their Application in Concurrency Control Problem. In *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems*, volume 5796 of *Lecture Notes in Computer Science*, pages 174–182. Springer Berlin Heidelberg, 2009.

[73] Rob van Glabbeek and Ursula Goltz. Refinement of Actions in Causality Based Models. In *Stepwise Refinement of Distributed Systems Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 267–300. Springer Berlin Heidelberg, 1990.

[74] Rob van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37:229–327, 2001.

[75] Rob van Glabbeek and Gordon Plotkin. Configuration Structures. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, LICS '95, pages 199–209, Washington, DC, USA, 1995. IEEE Computer Society.

[76] Rob van Glabbeek and Gordon Plotkin. Event Structures for Resolvable Conflict. In *Mathematical Foundations of Computer Science 2004*, volume 3153 of *Lecture Notes in Computer Science*, pages 550–561. Springer Berlin Heidelberg, 2004.

[77] Rob van Glabbeek and Gordon Plotkin. Configuration structures, event structures and Petri nets. *Theoretical Computer Science*, 410(41):4111–4159, 2009.

[78] Axel van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 249–262. IEEE Computer Society, 2001.

[79] Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic Event Structures and Domains. In *CONCUR 2004 - Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 481–496. Springer Berlin Heidelberg, 2004.

[80] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change patterns and change support features – Enhancing flexibility in process-aware information systems. *Data & Knowledge Engineering*, 66(3):438–466, 2008.

[81] Glynn Winskel. *Events in Computation*. PhD thesis, University of Edinburgh, 1980.

[82] Glynn Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer Berlin Heidelberg, 1987.

[83] Glynn Winskel. An Introduction to Event Structures. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop*, pages 364–397, London, UK, UK, 1989. Springer-Verlag.

[84] Glynn Winskel. Events, Causality and Symmetry. *The Computer Journal*, 54(1):42–57, 2011.

[85] Glynn Winskel. Distributed Probabilistic and Quantum Strategies. *Electronic Notes in Theoretical Computer Science*, 298(0):403–425, 2013. Proceedings of the Twenty-ninth Conference on the Mathematical Foundations of Programming Semantics, MFPS XXIX.

[86] Dayong Ye, Minjie Zhang, and Danny Sutanto. Self-Adaptation-Based Dynamic Coalition Formation in a Distributed Agent Network: A Mechanism and a Brief Survey. *IEEE Transactions on Parallel and Distributed Systems*, 24(5):1042–1051, 2013.

[87] Mikołaj Zuzek, Marek Talik, Tomasz Świerczyński, Cezary Wiśniewski, Bartosz Kryza, Łukasz Dutka, and Jacek Kitowski. Formal Model for Contract Negotiation in Knowledge-Based Virtual Organizations. In *Computational Science–ICCS 2008*, volume 5103 of *Lecture Notes in Computer Science*, pages 409–418. Springer Berlin Heidelberg, 2008.

A. Appendix: Proofs

A.1. Proofs of Sections 2.2, 3.2: Prime Event Structures

Lemma 2.2.4 states:

Let π be a PES, let $H \in \mathrm{C}(\pi)$. Then:

$$\forall C \subseteq E \setminus H \,.\, \big(C \in \mathrm{C}(\pi[H]) \iff H \cup C \in \mathrm{C}(\pi)\big).$$

Proof. Similar to the next proof of Lemma 3.2.5. □

Lemma 3.2.5 states:

Let π be a PES, $H \in \mathrm{C}(\pi)$, and $\rightarrow_{\mathrm{p}}^{*}$ be the reflexive and transitive closure of $\rightarrow_{\mathrm{p}}$. Then: $\forall X \subseteq E \setminus H \,.\, X \in \mathrm{C}(\pi[H]) \iff H \rightarrow_{\mathrm{p}}^{*} H \cup X$.

Proof. Let $\pi = (E, \#, \rightarrow)$ and let $\pi[H] = \big(E', \#', \rightarrow'\big)$ with the transition relation $\rightarrow_{\mathrm{p}'}$. **First:** let us prove $\forall X, Y \subseteq E' \,.\, \big(X \rightarrow_{\mathrm{p}'} Y \wedge X \in \mathrm{C}(\pi[H])\big) \implies H \cup X \rightarrow_{\mathrm{p}} H \cup Y$. Regarding set containment, we have:

$$\begin{aligned} X \rightarrow_{\mathrm{p}'} Y &\implies X \subseteq Y \\ &\implies H \cup X \subseteq H \cup Y \end{aligned}$$

Next:

$$\begin{aligned} X \rightarrow_{\mathrm{p}'} Y &\implies \forall e, e' \in Y \,.\, \neg(e \#' e') \\ &\implies \forall e, e' \in Y \,.\, \neg(e \# e') \end{aligned}$$

but we know that $\forall e, e' \in H \,.\, \neg(e \# e')$ since $H \in \mathrm{C}(\pi)$. Then to prove that $H \cup Y$ is conflict-free we need to prove that $\forall e \in H, e' \in Y \,.\, \neg(e \# e')$. Assume

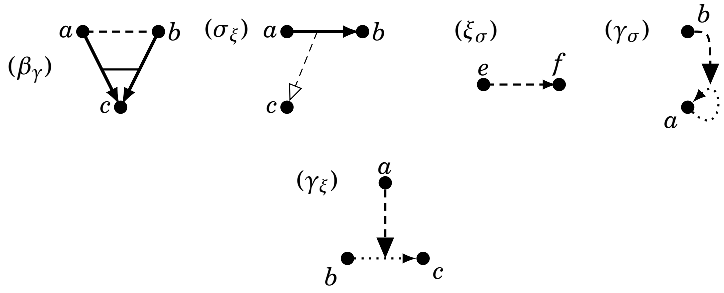


Figure A.1.: *Counterexamples for Equivalences between variations of Dynamic Causality Event Structures and Structures from the literature.*

the opposite, i.e. $\exists e \in H, e' \in Y \,.\, e \# e'$, then $e' \rightarrow' e'$, which makes e' unreachable (impossible). But $X \in \mathrm{C}(\pi[H])$, i.e. is reachable, then Y is reachable, and $e' \in Y$, which is a contradiction. Then $\forall e, e' \in H \cup Y \,.\, \neg(e \# e')$.

Let us prove $\forall e \in (H \cup Y) \setminus (H \cup X) \,.\, \{e' \in E \mid e' \rightarrow e\} \subseteq (H \cup Y)$. We know that $(H \cup Y) \setminus (H \cup X) = Y \setminus X$. Let $e \in Y \setminus X$, and $e' \in E$ such that $e' \rightarrow e'$, then we have two cases:

Case: $e' \in H$ Then $e' \in H \cup X$.

Case: $e' \notin H$ Then $e' \rightarrow' e$, then $e' \in X$ since $X \rightarrow_{\mathrm{p}'} Y$, then $e' \in H \cup X$.

Then by taking $X = \emptyset$ and applying transitivity we conclude:

$$\forall X \subseteq E' \,.\, X \in \mathrm{C}(\pi[H]) \implies H \rightarrow_{\mathrm{p}}^{*} X \cup H \tag{A.1}$$

Second: $\forall X, Y \subseteq E' \,.\, H \cup X \rightarrow_{\mathrm{p}} H \cup Y \implies X \rightarrow_{\mathrm{p}'} Y$ can be similarly proved. Then by taking $X = \emptyset$ and applying transitivity, we conclude that

$$\forall X \subseteq E' \,.\, H \rightarrow_{\mathrm{p}}^{*} X \cup H \implies X \in \mathrm{C}(\pi[H]) \tag{A.2}$$

From A.1 and A.2 we conclude: $\forall X \subseteq E' \,.\, X \in \mathrm{C}(\pi[H]) \iff H \rightarrow_{\mathrm{p}}^{*} X \cup H$. $\square$

A.2. Proofs of Section 3.3: Shrinking Causality

Lemma 3.3.5 states:

Let σ be a SES. Then $\mathrm{C}_{\mathrm{Tr}}(\sigma) = \mathrm{C}(\sigma)$.

Proof. Let $\sigma = (E, \#, \rightarrow, \triangleright)$. By Definition 3.3.2, $C \in \mathrm{C}_{\mathrm{Tr}}(\sigma)$ implies that there is some $t = e_1, \ldots, e_n$ such that $\overline{t} \subseteq E$, $\forall 1 \leq i, j \leq n \,.\, \neg\left(e_i \# e_j\right)$, $\forall 1 \leq i \leq n \,.\, \left(\mathrm{ic}(e_i) \setminus \mathrm{dc}\left(\overline{t_{i-1}}, e_i\right)\right) \subseteq \overline{t_{i-1}}$, and $C = \overline{t}$. Hence, by Definition 3.3.3, $\overline{t_i} \rightarrow_{\mathrm{s}} \overline{t_{i+1}}$ for all $1 \leq i \leq n$ and $\emptyset \rightarrow_{\mathrm{s}} \{e_1\}$. Thus, by Definition 3.3.4, $C \in \mathrm{C}(\sigma)$.

By Definition 3.3.4, $C \in \mathrm{C}(\sigma)$ implies that there are $X_1, \ldots, X_n \subseteq E$ such that $\emptyset \rightarrow_{\mathrm{s}} X_1 \rightarrow_{\mathrm{s}} \ldots \rightarrow_{\mathrm{s}} X_n$ and $X_n = C$. Then, by Definition 3.3.3, we have:

$$\emptyset \subseteq X_1 \subseteq X_2 \subseteq \ldots \subseteq X_n \subseteq E \tag{C1}$$

$$\forall e, e' \in X_n \,.\, \neg\left(e \# e'\right) \tag{C2}$$

$$\forall e \in X_1 \,.\, (\mathrm{ic}(e) \setminus \mathrm{dc}(\emptyset, e)) \subseteq \emptyset \tag{C3}$$

$$\forall 1 \leq i < n \,.\, \forall e \in X_{i+1} \setminus X_i \,.\, (\mathrm{ic}(e) \setminus \mathrm{dc}(X_i, e)) \subseteq X_i \tag{C4}$$

Let $X_1 = \left\{e_{1,1}, \ldots, e_{1,m_1}\right\}$ and $X_i \setminus X_{i-1} = \left\{e_{i,1}, \ldots, e_{i,m_i}\right\}$ for all $1 < i \leq n$. Then, by Definition 3.3.2:
$t = e_{1,1}, \ldots, e_{1,m_1}, \ldots, e_{n,1}, \ldots, e_{n,m_n} = e'_1, \ldots, e'_k$ is a trace such that $\overline{t} \subseteq E$ (because of (C1)), $\neg\left(e'_i \# e'_j\right)$ for all $1 \leq i, j \leq k$ (because of (C1) and (C2)), for all $1 \leq i \leq k$ and all $1 \leq j \leq m_i$ we have $\left(\mathrm{ic}\left(e_{i,j}\right) \setminus \mathrm{dc}\left(\overline{t_{i-1}}, e_{i,j}\right)\right) \subseteq \overline{t_{i-1}}$ (because of (C3) and (C4)), and $\overline{t} = C$ (because $X_n = C$). Thus $C \in \mathrm{C}_{\mathrm{Tr}}(\sigma)$. □

Moreover the following technical Lemma relates transitions and the extension of traces by causally independent events.

Lemma A.2.1. *Let $\sigma = (E, \#, \rightarrow, \triangleright)$ be a SES and $X, Y \in \mathrm{C}(\sigma)$. Then $X \rightarrow_{\mathrm{s}} Y$ iff there are:*
$t_1 = e_1, \ldots, e_n, t_2 = e_1, \ldots, e_n, e_{n+1}, \ldots, e_{n+m} \in \mathrm{T}(\sigma)$ *such that* $X = \overline{t_1}$, $Y = \overline{t_2}$, *and* $\forall e, e' \in Y \setminus X \,.\, (\mathrm{ic}(e) \setminus \mathrm{dc}(X, e)) \subseteq X$.

Proof. By Definition 3.3.2 and Lemma 3.3.5, $X \in \mathrm{C}(\sigma)$ implies that there is a trace $t_1 = e_1, \ldots, e_n \in \mathrm{T}(\sigma)$ such that $X = \overline{t_1}$.

If $X \rightarrow_{\mathrm{s}} Y$ then, by Definition 3.3.3: $X \subseteq Y$, $\forall e, e' \in Y \,.\, \neg\left(e \# e'\right)$, and $\forall e \in Y \setminus X \,.\, (\mathrm{ic}(e) \setminus \mathrm{dc}(X, e)) \subseteq X$. Then, by Definition 3.3.2, $t_2 = e_1, \ldots, e_n, e_{n+1}$

, ... , $e_{n+m} \in \mathrm{T}(\sigma)$ and $Y = \overline{t_2}$ for an arbitrary linearization $e_{n+1},\dots,e_{n+m}$ of the events in $Y \setminus X$, i.e. with $\{e_{n+1},\dots,e_{n+m}\} = Y \setminus X$ such that $e_{n+i} \neq e_{n+j}$ whenever $1 \leq i,j, \leq m$ and $i \neq j$.

If there is a trace $t_2 = e_1,\dots,e_n,e_{n+1},\dots,e_{n+m} \in \mathrm{T}(\sigma)$ such that $Y = \overline{t_2}$ and $\forall e,e' \in Y \setminus X\,.\,(\mathrm{ic}(e) \setminus \mathrm{dc}(X,e)) \subseteq X$ then $X \subseteq Y$. Moreover, by Definition 3.3.2, $t_2 \in \mathrm{T}(\sigma)$ implies $\forall e,e' \in Y\,.\,\neg\left(e\#e'\right)$. Thus, by Definition 3.3.3, $X \rightarrow_{\mathrm{s}} Y$. □

Note that the condition $\forall e,e' \in Y \setminus X\,.\,\left(\mathrm{ic}(e) \setminus \mathrm{dc}(X,e)\right) \subseteq X$ states that the events in $Y \setminus X$ are causally independent from each other.

Lemma 3.3.7 states:

Let σ be a SES, $H \in \mathrm{C}(\sigma)$. Then:

$$\forall X \subseteq E \setminus H\,.\,X \in \mathrm{C}(\sigma[H]) \iff H \rightarrow_{\mathrm{s}}^{*} H \cup X.$$

Proof. Let $\sigma[H] = \left(E',\#',\rightarrow',\rhd'\right)$. Let us prove:

$$\forall X,Y \subseteq E'\,.\,\left(X \rightarrow_{\mathrm{s}}' Y \wedge X \in \mathrm{C}(\sigma[H])\right) \implies H \cup X \rightarrow_{\mathrm{s}} H \cup Y$$

Regrading set containment, $X \rightarrow_{\mathrm{s}}' Y$ means $X \subseteq Y$, then $H \cup X \subseteq H \cup Y$. Conflict-freeness is proved similarly to Lemma 3.2.5, with taking into consideration that loops can be dropped in SES, but not for events conflicting with H according to the definition of $\rhd$.

Let us prove $\forall e \in (H \cup Y) \setminus (H \cup X)\,.\ \mathrm{ic}(e) \setminus \mathrm{dc}(H \cup X,e) \subseteq H \cup X$ for $e \in (H \cup Y) \setminus (H \cup X) = Y \setminus X$. Let $e' \in \mathrm{ic}(e) \setminus \mathrm{dc}(H \cup X,e)$, then $e' \in \mathrm{ic}(e)$ and $e' \notin \mathrm{dc}(H \cup X,e)$. But $e' \in \mathrm{ic}(e)$ means $e' \rightarrow e$. We have two possibilities, the first is $e' \in H$ then $e' \in H \cup X$. The second is $e' \notin H$, then $e' \in E'$, i.e. $(e',e) \in\ \rightarrow \cap E'^2$. But since $e' \notin \mathrm{dc}(H \cup X,e)$, then $e' \notin \mathrm{dc}(H,e)$. Then $e' \rightarrow' e$, i.e. $e' \in \mathrm{ic}'(e)$. Furthermore, $e' \notin \mathrm{dc}(H \cup X,e)$ means $e' \notin \mathrm{dc}(X,e)$, which means $e' \notin \mathrm{dc}'(X,e)$ since $\rhd' \subseteq \rhd \cap E'^3$. But we know that $\mathrm{ic}'(e) \setminus \mathrm{dc}'(X,e) \subseteq X$ for $e \in Y \setminus X$ since $X \rightarrow_{\mathrm{s}}' Y$, then $e' \in X$ i.e. $e' \in H \cup X$.

By taking $X = \emptyset$ and applying transitivity we conclude:

$$\forall X \subseteq E'\,.\,X \in \mathrm{C}(\sigma[H]) \implies H \rightarrow_{\mathrm{s}}^{*} X \cup H \tag{A.3}$$

Similarly we can prove $\forall X,Y \subseteq E'\,.\,H \cup X \rightarrow_{\mathrm{s}} H \cup Y \implies X \rightarrow_{\mathrm{s}}' Y$, which yields:

$$\forall X \subseteq E'\,.\,X \in \mathrm{C}(\sigma[H]) \Longleftarrow H \rightarrow_{\mathrm{s}}^{*} X \cup H \tag{A.4}$$

From A.3 and A.4 we conclude: $\forall X \subseteq E'\,.\,X \in \mathrm{C}(\sigma[H]) \iff H \rightarrow_{\mathrm{s}}^{*} H \cup X$. □

Lemma A.2.2. *For each SES σ there is a DES δ, namely $\delta = \mathrm{des}(\sigma)$, such that $\mathrm{T}(\sigma) = \mathrm{T}(\delta)$ and $\mathrm{C}(\sigma) = \mathrm{C}(\delta)$.*

Proof of Lemma A.2.2. Let $\sigma = (E, \#, \to, \rhd)$ be a SES. By Definitions 3.3.1 and 3.2.1, $\# \subseteq E^2$ is irreflexive and symmetric. Hence, by Definitions 2.6.1 and 3.3.8, $\delta = \mathrm{des}(\sigma)$ is a DES.

Let $t = e_1, \ldots, e_n$. By Definition 3.3.2, $t \in \mathrm{T}(\sigma)$ iff $\bar{t} \subseteq E$, $\neg\left(e_i \# e_j\right)$, and $(\mathrm{ic}(e_i) \setminus \mathrm{dc}(\overline{t_{i-1}}, e_i)) \subseteq \overline{t_{i-1}}$ for all $1 \leq i, j \leq n$. Since $\mathrm{dc}(H, e) = \{e' \mid \exists d \in H \,.\, [e' \to e] \rhd d\}$ and $\mathrm{ic}(e) = \{e' \mid e' \to e\}$, we have $\left(\mathrm{ic}(e_i) \setminus \mathrm{dc}(\overline{t_{i-1}}, e_i)\right) \subseteq \overline{t_{i-1}}$ iff $\forall e' \in E \,.\, e' \to e_i \implies e' \in \overline{t_{i-1}} \vee \exists d \in \overline{t_{i-1}} \,.\, [e' \to e_i] \rhd d$ for all $1 \leq i \leq n$. By Definition 3.3.8, then $t \in \mathrm{T}(\sigma)$ iff $\bar{t} \subseteq E$, $\neg\left(e_i \# e_j\right)$, and $X \mapsto e_i \implies \overline{t_{i-1}} \cap X \neq \emptyset$ for all $1 \leq i, j \leq n$ and all $X \subseteq E$. Hence, by the definition of traces in Section 2.6, $t \in \mathrm{T}(\sigma)$ iff $t \in \mathrm{T}(\delta)$, i.e. $\mathrm{T}(\sigma) = \mathrm{T}(\delta)$.

By Lemma 3.3.5, Section 2.6, and Definition 3.3.2, then also:

$$\mathrm{C}(\delta) = \mathrm{C}_{\mathrm{Tr}}(\sigma) = \mathrm{C}(\sigma)$$

□

The most discriminating behavioral semantics of DESs used in literature are families of posets. Thus the translation should also preserve posets. Authors in [48] show that the early causality and trace equivalence coincide for Dual ESs. Thus we concentrate on there early causality. The remaining intentional partial order semantics are discussed in the Appendix B. To capture causal ambiguity we have to consider all traces of a configuration to obtain its posets.

Definition A.2.3. *Let $\delta = (E, \#, \mapsto)$ be a DES, $t = e_1, \ldots, e_n$ one of its traces, $1 \leq i \leq n$, and $X_1 \mapsto e_i, \ldots, X_m \mapsto e_i$ all bundles pointing to e_i. A set U is a cause of e_i in t if $\forall e \in U \,.\, \exists 1 \leq j < i \,.\, e = e_j$, $\forall 1 \leq k \leq m \,.\, X_k \cap U \neq \emptyset$, and U is the earliest set satisfying the previous two conditions. Let $\mathrm{P_d}(t)$ be the set of posets obtained this way for t.*

Theorem 3.3.10 states:

For each SES σ there is a DES $\delta = \mathrm{des}(\sigma)$, such that $\sigma \simeq_{\mathrm{p}} \delta$.

Proof. Let $\sigma = (E, \#, \to, \rhd)$ be a SES.
By Lemma A.2.2, $\delta = \mathrm{des}(\sigma) = (E, \#, \mapsto)$ is a DES such that $\mathrm{T}(\sigma) = \mathrm{T}(\delta)$ and $\mathrm{C}(\sigma) = \mathrm{C}(\delta)$.

Let $t = e_1, \ldots, e_n \in \mathrm{T}(\sigma)$, $1 \leq i \leq n$, and the bundles $X_1 \mapsto e_i, \ldots, X_m \mapsto e_i$ all bundles pointing to e_i. For U to be a cause for e_i Definition 3.3.9 requires $(\mathrm{ic}(e_i) \setminus \mathrm{dc}(U, e_i)) \subseteq U$. Since $\mathrm{dc}(H,e) = \{e' \mid \exists d \in H \,.\, [e' \to e] \triangleright d\}$ and $\mathrm{ic}(e) = \{e' \mid e' \to e\}$, this condition holds iff the condition $e' \to e_i \implies e' \in U \vee \exists d \in U \,.\, [e' \to e_i] \triangleright d$ holds for all $e' \in E$. By Definition 3.3.8, then $(\forall 1 \leq k \leq n \,.\, X_k \cap U \neq \emptyset) \iff ((\mathrm{ic}(e_i) \setminus \mathrm{dc}(U, e_i)) \subseteq U)$. So, by Definitions A.2.3 and 3.3.9, $\sigma \simeq_{\mathrm{p}} \delta$. □

The opposite direction of the translation from DESs to SESs:
Lemma 3.3.12 states:

> There are DESs $\delta = (E, \#, \mapsto)$, e.g. $\delta = (\{a,b,c,d,e\}, \emptyset, \mapsto)$ with $\mapsto = \{\{x,y\} \mapsto e \mid x,y \in \{a,b,c,d\} \wedge x \neq y\}$, that cannot be translated into a SES $\sigma = (E, \#', \to, \triangleright)$ such that $\mathrm{T}(\delta) = \mathrm{T}(\sigma)$.

Proof of Lemma 3.3.12. Assume a SES $\sigma = (E, \#, \to, \triangleright)$ such that $E = \{a, b, c, d, e\}$ and $\mathrm{T}(\sigma) = \mathrm{T}(\delta)$. According to Section 2.6, $\mathrm{T}(\delta)$ contains all sequences of distinct events of E such that e is not the first, second, or third event, i.e. for e to occur in a trace it has to be preceded by at least three of the other events. Since by Definition 3.3.2 conflicts cannot be dropped, $\mathrm{T}(\sigma) = \mathrm{T}(\delta)$ implies $\# = \emptyset$. Moreover, since e has to be preceded by at least three other events that can occur in any order, $\to$ has to contain at least three initial causes for e. W.l.o.g. let $a \to e$, $b \to e$, and $c \to e$. Because of the traces $abd, acd \in \mathrm{T}(\delta)$, we need the droppers $[b \to e] \triangleright d$ and $[c \to e] \triangleright d$. Then $ad \in \mathrm{T}(\sigma)$ but $ad \notin \mathrm{T}(\delta)$. In fact if we fix $E = \{a,b,c,d,e\}$ there only finitely many different SESs $\sigma = (E, \#, \to, \triangleright)$ and for none of them $\mathrm{T}(\delta) = \mathrm{T}(\sigma)$ holds. □

Lemma A.2.4. *For each DES δ there is a SES σ, namely $\sigma = \mathrm{ses}(\delta)$, such that*

$$\mathrm{T}(\delta) = \mathrm{T}(\sigma) \quad \textit{and} \quad \mathrm{C}(\delta) = \mathrm{C}(\sigma).$$

Proof of Lemma A.2.4. Let $\delta = (E, \#, \mapsto)$ be a DES. By the definition of DESs in 2.6, $\# \subseteq E^2$ is irreflexive and symmetric. Hence, by Definitions 3.3.1, 3.2.1, and 3.3.11, $\sigma = \mathrm{ses}(\delta) = (E', \#, \to, \triangleright)$ is a SES.

Let $t = e_1, \ldots, e_n$. Then, by Definition 3.3.2, $t \in \mathrm{T}(\sigma)$ iff $\bar{t} \subseteq E$, $\neg(e_i \# e_j)$, and $\left(\mathrm{ic}(e_i) \setminus \mathrm{dc}(\overline{t_{i-1}}, e_i)\right) \subseteq \overline{t_{i-1}}$ for all $1 \leq i,j \leq n$. Note that we have $\bar{t} \subseteq E$ instead of $\bar{t} \subseteq E'$, because all events in t have to be distinct and for all events in $E' \setminus E$ there is an initial self-loop but no dropper. Since $\mathrm{dc}(H,e) = \{e' \mid \exists d \in H \,.\, [e' \to e] \triangleright d\}$ and $\mathrm{ic}(e) = \{e' \mid e' \to e\}$, we have $\left(\mathrm{ic}(e_i) \setminus \mathrm{dc}(\overline{t_{i-1}}, e_i)\right)$

$\subseteq \overline{t_{i-1}}$ iff $\forall e' \in E\,.\, e' \to e_i \implies e' \in \overline{t_{i-1}} \vee \exists d \in \overline{t_{i-1}}\,.\, [e' \to e_i] \rhd d$ for all $1 \leq i \leq n$. By Definition 3.3.11, then $t \in \mathrm{T}(\sigma)$ iff $\bar{t} \subseteq E$, $\neg(e_i \# e_j)$, and $X \mapsto e_i \implies \overline{t_{i-1}} \cap X \neq \emptyset$ for all $1 \leq i,j \leq n$ and all $X \subseteq E$. Hence, by the definition of traces in Section 2.6, $t \in \mathrm{T}(\sigma)$ iff $t \in \mathrm{T}(\delta)$, i.e. $\mathrm{T}(\sigma) = \mathrm{T}(\delta)$.

By Lemma 3.3.5, the definition of configurations in Section 2.6, and Definition 3.3.2, then also $\mathrm{C}(\delta) = \mathrm{C}_{\mathrm{Tr}}(\sigma) = \mathrm{C}(\sigma)$. □

Moreover the DES and its translation have exactly the same posets.
Theorem 3.3.13 states:

For each DES δ there is a SES $\sigma = \mathrm{ses}(\delta)$, such that $\delta \simeq_{\mathrm{p}} \sigma$.

Proof of Theorem 3.3.13. Let $\delta = (E, \#, \mapsto)$ be a DES. By Lemma A.2.4, $\sigma = \mathrm{ses}(\delta) = (E, \#, \to, \rhd)$ is a SES such that $\mathrm{T}(\delta) = \mathrm{T}(\sigma)$ and $\mathrm{C}(\delta) = \mathrm{C}(\sigma)$.
Let $t = e_1, \ldots, e_n \in \mathrm{T}(\delta)$, $1 \leq i \leq n$, and the bundles $X_1 \mapsto e_i, \ldots, X_m \mapsto e_i$ all bundles pointing to e_i. For U to be a cause for e_i Definition 3.3.9 requires $(\mathrm{ic}(e_i) \setminus \mathrm{dc}(U, e_i)) \subseteq U$. Since $\mathrm{dc}(H,e) = \{e' \mid \exists d \in H\,.\, [e' \to e] \rhd d\}$ and $\mathrm{ic}(e) = \{e' \mid e' \to e\}$, this condition holds iff the condition $e' \to e_i \implies e' \in U \vee \exists d \in U\,.\, [e' \to e_i] \rhd d$ holds for all $e' \in E$. By Definition 3.3.11, then $(\forall 1 \leq k \leq n\,.\, X_k \cap U \neq \emptyset)$ iff $\Big((\mathrm{ic}(e_i) \setminus \mathrm{dc}(U, e_i)) \subseteq U\Big)$. So, by Definitions A.2.3 and 3.3.9, $\delta \simeq_{\mathrm{p}} \sigma$. □

Thus SESs and DESs have the same expressive power.

Proof of Theorem 3.3.14. By Theorems 3.3.10 and 3.3.13. □

Theorem 3.3.16 states:

Let σ, σ' be two SESs. Then $\sigma \simeq_{\mathrm{p}} \sigma' \iff \sigma \simeq_{\mathrm{t}} \sigma' \iff \mathrm{T}(\sigma) = \mathrm{T}(\sigma')$.

Proof of Theorem 3.3.16. By Corollary 3.3.15, $\sigma \simeq_{\mathrm{p}} \sigma'$ iff $\mathrm{T}(\sigma) = \mathrm{T}(\sigma')$.

If $\mathrm{C}(\sigma) \neq \mathrm{C}(\sigma')$ then, by Lemma 3.3.5 and Definitions 3.3.9 and 3.3.3, $\sigma \not\simeq_{\mathrm{p}} \sigma'$ and $\sigma \not\simeq_{\mathrm{t}} \sigma'$. Hence assume $\mathrm{C}(\sigma) = \mathrm{C}(\sigma')$. Note that, by Definition 3.3.2 and Lemma 3.3.5, for all $C \in \mathrm{C}(\sigma)$ there is a trace $t \in \mathrm{T}(\sigma)$ such that $\bar{t} = C$. Moreover for every trace $t \in \mathrm{T}(\sigma)$ except the empty trace there is a sub-trace $t' \in \mathrm{T}(\sigma)$ and a sequence of events $e_1, \ldots, e_m$ such that $t = t'e_1, \ldots, e_m$ and $\forall e \in \{e_1, \ldots, e_m\}\,.\, \left(\mathrm{ic}(e) \setminus \mathrm{dc}\left(\overline{t'}, e\right)\right) \subseteq \overline{t'}$. Thus, by Lemma A.2.1, $\mathrm{T}(\sigma) = \mathrm{T}(\sigma')$ iff $\sigma \simeq_{\mathrm{t}} \sigma'$. □

Theorem 3.3.17 states:

SESs and EBESs are incomparable.

Proof of Theorem 3.3.17.
Let $\sigma_\xi = (\{a,b,c\}, \emptyset, \{a \to b\}, \{[a \to b] \triangleright c\})$ be the SES that is depicted in Figure A.1. Assume there is some EBES $\xi = (E, \rightsquigarrow, \mapsto)$ such that $\mathrm{T}(\sigma_\xi) = \mathrm{T}(\xi)$. By Definition 3.3.2, $\mathrm{T}(\sigma_\xi) = \{\epsilon, a, c, ab, ac, ca, cb, abc, acb, cab, cba\}$, i.e. b cannot occur first. By Definition 2.5.2, a disabling $x \rightsquigarrow y$ implies that y can never precedes x. Thus we have $\rightsquigarrow \cap \{a,b,c\}^2 = \emptyset$, because within $\mathrm{T}(\sigma_\xi)$ each pair of events of $\{a,b,c\}$ occur in any order. Similarly we have $\mapsto \cap \{X \mapsto e \mid e \in \{a,b,c\} \wedge X \cap \{a,b,c\} = \emptyset\} = \emptyset$, because $x \mapsto y$ implies that x always has to precede y. Moreover, by Definition 2.5.2, adding impossible events as causes or using them within the disabling relation does not influence the set of traces. Thus there is no EBES ξ with the same traces as σ_ξ. By Definition 2.5.2 and the definition of posets in EBESs, then there is no EBES ξ with the same configurations or posets as σ_ξ.

Let $\xi_\sigma = (\{e,f\}, \{e \rightsquigarrow f\}, \emptyset)$ be the EBES that is depicted in Figure A.1. Assume there is some SES $\sigma = (E, \#, \to, \triangleright)$ such that $\mathrm{T}(\xi_\sigma) = \mathrm{T}(\sigma)$. According to Section 2.5, $\mathrm{T}(\xi_\sigma) = \{\epsilon, e, f, ef\}$. By Definition 3.3.2 and because of the traces e and f, there are no initial causes for e and f, i.e. $\to \cap \{x \to y \mid y \in \{e,f\}\} = \emptyset$. Moreover, $\# \cap \{e,f\}^2 = \emptyset$, because of the trace ef and because conflicts cannot be dropped. Thus $fe \in \mathrm{T}(\sigma)$ but $fe \notin \mathrm{T}(\xi_\sigma)$, i.e. there is no SES σ with the same traces as ξ_σ. Then by Definitions 3.3.2 and 3.3.9, there is no SES σ with the same configurations or families of posets as ξ_σ. □

Lemma 3.3.18 states:

For each SES σ there is a RCES ρ, such that $\sigma \simeq_{\mathrm{t}} \rho$.

Proof. By Definitions 3.3.3 and 3.3.4, $X \to_{\mathrm{s}} Y$ implies $X \subseteq Y$ for all $X, Y \in \mathrm{C}(\sigma)$.

Assume $X \subseteq X' \subseteq Y' \subseteq Y$. Then, by Definition 3.3.3, $X \to_{\mathrm{s}} Y$ implies $\forall e, e' \in Y \,.\, \neg\left(e \# e'\right)$ and $\forall e \in Y \setminus X \,.\, \left(\mathrm{ic}(e) \setminus \mathrm{dc}(X, e)\right) \subseteq X$. Then $X \subseteq X'$ implies $\left(\mathrm{ic}(e) \setminus \mathrm{dc}(X', e)\right) \subseteq \left(\mathrm{ic}(e) \setminus \mathrm{dc}(X, e)\right)$. Then $\forall e, e' \in Y' \,.\, \neg\left(e \# e'\right)$ and $\forall e \in Y' \setminus X' \,.\, \left(\mathrm{ic}(e) \setminus \mathrm{dc}(X', e)\right) \subseteq X'$, because of $Y' \subseteq Y$. By Definition 3.3.3, then $X' \to_{\mathrm{s}} Y'$.

Thus σ satisfies the conditions of Definition 2.7.5. Then by Lemma 2.7.6, $\rho = \mathrm{rces}(\sigma)$ is a RCES such that $\sigma \simeq_{\mathrm{t}} \rho$. □

Lemma 3.3.19 states:

There is no transition-equivalent SES to the RCES

$$\rho_\sigma = \Big(\{e,f\}, \big\{\emptyset \vdash \{e\}, \emptyset \vdash \{f\}, \{f\} \vdash \{e,f\}\big\}\Big).$$

Proof. Assume a SES $\sigma = (E, \#, \rightarrow, \rhd)$ such that $\sigma \simeq_t \rho_\sigma$. Then $C(\sigma) = C(\rho_\sigma)$. By Definition 3.3.2 and Lemma 3.3.5 and because of the configuration $\{e,f\} \in C(\rho_\sigma)$, the events e and f cannot be in conflict with each other, i.e. $\# \cap \{e,f\}^2 = \emptyset$. Moreover, because of the configurations $\{e\}, \{f\} \in C(\rho_\sigma)$, there are no initial causes for e and f, i.e. $\rightarrow \cap \{x \rightarrow y \mid y \in \{e,f\}\} = \emptyset$. Note that the relation $\rhd$ cannot disable events. Thus we have $\forall a,b \in \{e,f\} \,.\, \neg(a\#b)$ and $(\mathrm{ic}(e) \setminus \mathrm{dc}(\{f\},e)) = \emptyset \subseteq \{f\}$. But then, by Definition 3.3.3, $\{f\} \rightarrow_s \{e,f\}$. Since $\{f\} \rightarrow_{rc} \{e,f\}$ does not hold, this violates our assumption, i.e. there is no SES which is transition equivalent to ρ_σ. □

Theorem 3.3.20 states:

SESs are strictly less expressive than RCESs.

Proof of Theorem 3.3.20. By Lemmas 3.3.19 and 3.3.18. □

A.3. Proofs of Section 3.4: Growing Causality

Lemma 3.4.5 states:

Let γ be a GES. Then $C_{Tr}(\gamma) = C(\gamma)$.

Proof. Let $\gamma = (E, \#, \rightarrow, \blacktriangleright)$.

By Definition 3.4.2, $C \in C_{Tr}(\gamma)$ implies that there is some $t = e_1, \ldots, e_n$ such that $\bar{t} \subseteq E$, $\forall 1 \leq i,j \leq n \,.\, \neg(e_i \# e_j)$, $\forall 1 \leq i \leq n \,.\, \big(\mathrm{ic}(e_i) \cup \mathrm{ac}(\overline{t_{i-1}}, e_i)\big) \subseteq \overline{t_{i-1}}$, and $C = \bar{t}$. Hence, by Definition 3.4.3, $\overline{t_i} \rightarrow_g \overline{t_{i+1}}$ for all $1 \leq i \leq n$ and $\emptyset \rightarrow_g \{e_1\}$. Thus, by Definition 3.4.4, $C \in C(\gamma)$.

By Definition 3.4.4, $C \in C(\gamma)$ implies that there are $X_1, \ldots, X_n \subseteq E$ such that $\emptyset \rightarrow_g X_1 \rightarrow_g \ldots \rightarrow_g X_n$ and $X_n = C$. Then, by Definition 3.4.3, we have:

$$\emptyset \subseteq X_1 \subseteq X_2 \subseteq \ldots \subseteq X_n \subseteq E \tag{D1}$$

$$\forall e,e' \in X_n \,.\, \neg(e\#e') \tag{D2}$$

$$\forall e \in X_1 \,.\, \big(\mathrm{ic}(e) \cup \mathrm{ac}(\emptyset, e)\big) \subseteq \emptyset \tag{D3}$$

$$\forall 1 \leq i < n \,.\, \forall e \in X_{i+1} \setminus X_i \,.\, \big(\mathrm{ic}(e) \cup \mathrm{ac}(X_i, e)\big) \subseteq X_i \tag{D4}$$

$$\forall 1 \leq i < n \,.\, \forall t,m \in X_{i+1} \setminus X_i \,.\, \forall c \in E \,.\, m \blacktriangleright [c \rightarrow t] \implies c \in X_i \tag{D5}$$

Let $X_1 = \{e_{1,1},\ldots,e_{1,m_1}\}$ and $X_i \setminus X_{i-1} = \{e_{i,1},\ldots,e_{i,m_i}\}$ for all $1 < i \leq n$. Then, by Definition 3.4.2:
$t = e_{1,1},\ldots,e_{1,m_1},\ldots,e_{n,1},\ldots,e_{n,m_n} = e'_1,\ldots,e'_k$ is a trace such that $\overline{t} \subseteq E$ (because of (D1)), $\neg\left(e'_i \# e'_j\right)$ for all $1 \leq i,j \leq k$ (because of (D1) and (D2)), for all $1 \leq i \leq k$ and all $1 \leq j \leq m_i$ we have $\left(\mathrm{ic}(e_{i,j}) \cup \mathrm{ac}(\overline{t_{i-1}}, e_{i,j})\right) \subseteq \overline{t_{i-1}}$ (because of (D3), (D4), and, by (D5), $\mathrm{ac}(\overline{t_{i-1}} \cup X_i, e_{i,j}) = \mathrm{ac}(\overline{t_{i-1}}, e_{i,j})$), and $\overline{t} = C$ (because $X_n = C$). Thus $C \in \mathrm{C_{Tr}}(\gamma)$. □

Lemma 3.4.7 states:

Let γ be a GES, and $H \in \mathrm{C}(\gamma)$. Then:

$$\forall X, Y \subseteq E \setminus H \,.\, X \in \mathrm{C}(\gamma[H]) \iff H \rightarrow_{\mathrm{g}}^{*} H \cup X.$$

Proof. Similar to the proof of Lemma 3.3.7, w.r.t. to ac and Condition 3.4.3 of Definition 3.4.3. □

For the incomparability result between GESs and EBESs we consider two counterexamples, and show that there is no equivalent EBES or GES respectively.

Lemma A.3.1. *There is no configuration-equivalent GES to β_γ (cf. Figure A.1).*

Proof. Assume a GES $\gamma = (E, \#', \rightarrow, \blacktriangleright)$ such that $\mathrm{C}(\gamma) = \mathrm{C}(\beta_\gamma)$. According to Section 2.4, $\mathrm{C}(\beta_\gamma) = \{\emptyset, \{a\}, \{b\}, \{a,c\}, \{b,c\}\}$. Because $\{c\} \notin \mathrm{C}(\beta_\gamma)$, $\{a,c\} \in \mathrm{C}(\beta_\gamma)$, and by Definition 3.4.2 and Lemma 3.4.5, a has to be an initial cause of c in γ, i.e. $a \rightarrow c$. But then, by Definition 3.4.2 and Lemma 3.4.5, $\{b,c\} \notin \mathrm{C}(\gamma)$ although $\{b,c\} \in \mathrm{C}(\beta_\gamma)$. This violates our assumption, i.e. no GES can be configuration-equivalent to β_γ. □

Lemma A.3.2. *There is no trace-equivalent EBES to γ_ξ (cf. Figure A.1).*

Proof. Assume a EBES $\xi = (E, \#, \mapsto)$ such that $\mathrm{T}(\xi) = \mathrm{T}(\gamma_\xi)$. By Definition 3.4.2, $a, c, ca, bac \in \mathrm{T}(\gamma_\xi)$ and $ac \notin \mathrm{T}(\gamma_\xi)$. Because of $a, c \in \mathrm{T}(\gamma_\xi)$ and by Definition 2.5.2, a and c have to be initially enabled in ξ, i.e.
$\mapsto \cap \{X \mapsto y \mid y \in \{a,c\}\} = \emptyset$. Moreover, because of $ca, bac \in \mathrm{T}(\gamma_\xi)$, a cannot disable c, i.e. $\neg(a \rightsquigarrow c)$. But then $ac \in \mathrm{T}(\xi)$. This violates our assumption, i.e. there is no trace-equivalent EBES to γ_ξ. □

Theorem 3.4.8 states:

GESs are incomparable to BESs and EBESs.

Proof of Theorem 3.4.8. By Lemma A.3.1, there is no GES that is configuration equivalent to the BES β_γ. Thus no GES can have the same families of posets as the BES β_γ, because two BES with different configurations cannot have the same families of posets (cf. Section 2.4). Moreover, by Definitions 2.4.1 and 2.5.1, each BES is also an EBES. Thus no GES can have the same families of posets as the EBES β_γ.

By Lemma A.3.2, there is no EBES and thus also no BES that is trace-equivalent to the GES γ_ξ. By Definition 3.4.3, two GES with different traces cannot have the same transition graphs. Thus no EBES or BES can be transition-equivalent to γ_ξ. □

For the incomparability between GESs and SESs, we study a GES counterexample, such that no SES is trace-equivalent.

Lemma A.3.3. *There is no trace-equivalent SES to γ_σ (cf. Figure A.1).*

Proof. Assume a SES $\sigma = (E, \#, \rightarrow, \triangleright)$ such that $\mathrm{T}(\sigma) = \mathrm{T}(\gamma_\sigma)$. By Definition 3.4.2, $\mathrm{T}(\gamma_\sigma) = \{\epsilon, a, b, ab\}$. Because of the trace $ab \in \mathrm{T}(\gamma_\sigma)$ and by Definition 3.3.2, a and b cannot be in conflict, i.e. $\neg(a\#b)$ and $\neg(b\#a)$. Moreover, because of the traces $a, b \in \mathrm{T}(\gamma_\sigma)$, there are no initial cases for a or b, i.e. $\rightarrow \cap \{x \rightarrow y \mid y \in \{a,b\}\} = \emptyset$. Thus, by Definition 3.3.2, $ba \in \mathrm{T}(\sigma)$ but $ba \notin \mathrm{T}(\gamma_\sigma)$. This violates our assumption, i.e. no SES can be trace equivalent to γ_σ. □

Theorem 3.4.9 states:

GESs and SESs are incomparable.

Theorem 3.4.9. By Lemma A.3.3, no SES is trace-equivalent to the GES γ_σ. By Definition 3.4.3, two GES with different traces cannot have the same transition graphs. Thus no SES is transition-equivalent to the GES γ_σ.

By [47], BESs are less expressive than EBESs and by [48], BESs are less expressive than DESs. By Theorem 3.4.8, BESs and GESs are incomparable an by Theorem 3.3.14 DESs are as expressive as SESs. Thus GESs and SESs are incomparable. □

To show that GESs are strictly less expressive than RCESs, we give a translation for one direction and a counterexample for the other.

Lemma A.3.4. *For each GES γ there is an RCES ρ, such that $\gamma \simeq_t \rho$.*

Proof. Let $\gamma = (E, \#, \rightarrow, \blacktriangleright)$. By Definition 3.4.3, $X \rightarrow_g Y$ implies $X \subseteq Y$.

Assume $X \subseteq X' \subseteq Y' \subseteq Y$ and $X \rightarrow_g Y$. By Definition 3.4.3, we have first $\forall e, e' \in Y' . \neg\left(e \# e'\right)$, second $\forall e \in \left(Y' \setminus X'\right) . (\mathrm{ic}(e) \cup \mathrm{ac}(X, e)) \subseteq X$, and third $\forall t, m \in Y \setminus X . \forall c \in E . m \blacktriangleright [c \rightarrow t] \implies c \in X$. Moreover, because $\forall t, m \in Y \setminus X . \forall c \in E . m \blacktriangleright [c \rightarrow t] \implies c \in X$, $\mathrm{ac}(X, e) = \mathrm{ac}\left(X', e\right)$ for all $e \in Y' \setminus X'$. Hence $\forall e \in \left(Y' \setminus X'\right) . \left(\mathrm{ic}(e) \cup \mathrm{ac}\left(X', e\right)\right) \subseteq X'$ and $\forall t, m \in Y' \setminus X' . \forall c \in E . m \blacktriangleright [c \rightarrow t] \implies c \in X'$. Thus, by Definition 3.4.3, $X' \rightarrow_g Y'$.

By Lemma 2.7.6, $\rho = \mathrm{rces}\left(\gamma\right)$ is a RCES and $\gamma \simeq_t \rho$. □

Lemma A.3.5. *There is no transition-equivalent GES to $\rho_\gamma = (\{a, b, c\}, \vdash)$.*

Proof. Assume a GES $\gamma = (E, \#, \rightarrow, \blacktriangleright)$ such that $\gamma \simeq_t \rho_\gamma$. Then $\mathrm{C}\left(\gamma\right) = \mathrm{C}\left(\rho_\gamma\right)$. By Definition 3.4.3 and because of the configuration $\{a, b, c\} \in \mathrm{C}\left(\rho_\gamma\right)$, the events a, b, and c cannot be in conflict with each other, i.e. $\# \cap \{a, b, c\}^2 = \emptyset$. Moreover, because of the configurations $\{a\}, \{b\}, \{c\} \in \mathrm{C}\left(\rho_\gamma\right)$, there are no initial causes for a, b, or c, i.e. $\rightarrow \cap \{x \rightarrow y \mid y \in \{a, b, c\}\} = \emptyset$. Finally, because of the configurations $\{a, c\}, \{b, c\} \in \mathrm{C}\left(\rho_\gamma\right)$, neither a nor b can add a cause (except of themselves) to c, i.e. $a \blacktriangleright [e \rightarrow c] \implies e = a$ and $b \blacktriangleright [e \rightarrow c] \implies e = b$ for all $e \in E$. Thus we have $\forall e, e' \in \{a, b, c\} . \neg\left(e \# e'\right)$ and $(\mathrm{ic}(c) \cup \mathrm{ac}(\{a, b\}, c)) = \emptyset \subseteq \{a, b\}$. But then, by Definition 3.4.3, $\{a, b\} \rightarrow_g \{a, b, c\}$. Since $\neg(\{a, b\} \rightarrow_{rc} \{a, b, c\})$, this violates our assumption, i.e. there is no GES that is transition equivalent to ρ_γ. □

Theorem 3.4.10 states:

GESs are strictly less expressive than RCESs.

Proof of Theorem 3.4.10. By Lemmas A.3.4 and A.3.5. □

A.4. Proofs of Section 3.5: Fully Dynamic Causality

Lemma 3.5.8 states:

> Let ϱ be a SSDC. Then for the causal-state function cs of any state $(C, cs) \in \mathrm{S}(\varrho)$ it holds $cs(e) = \big(\mathrm{ic}(e) \cup \mathrm{ac}(C,e)\big) \setminus \big(\mathrm{dc}(C,e) \cup C\big)$.

Proof. If $C = \emptyset$ the equation follows directly from the definitions of cs, ic, ac, and dc.

Assume $(C, \mathrm{cs}) \rightarrow_{\mathrm{d}} (C', \mathrm{cs}')$. By induction, we have $cs(e) = \big(\mathrm{ic}(e) \cup \mathrm{ac}(C,e)\big) \setminus \big(\mathrm{dc}(C,e) \cup C\big)$. We prove for each $e \in E \setminus C'$ by a doubled case distinction $\mathrm{cs}'(e) = \big(\mathrm{ic}(e) \cup \mathrm{ac}(C',e)\big) \setminus \big(\mathrm{dc}(C',e) \cup C\big)$. Let us first assume $e' \notin \mathrm{cs}(e)$ but $e' \in \mathrm{cs}'(e)$, then by Condition 5 (the *if* condition) we have $\exists a \in C' \setminus C \,.\, a \blacktriangleright [e' \rightarrow e]$ and since $\mathrm{ac}(C',e) = \{e' \mid \exists a \in C' \,.\, a \blacktriangleright [e' \rightarrow e] \wedge a \notin \{e, e'\}\}$ we have $e' \in \mathrm{ac}(C',e)$, because ϱ is a SSDC it follows $e' \notin \mathrm{dc}(C',e)$ and because $e \in E \setminus C'$ it follows $e \notin C$. Then in this case $e' \in \big(\mathrm{ic}(e) \cup \mathrm{ac}(C',e)\big) \setminus \big(\mathrm{dc}(C',e) \cup C\big)$ holds. Let now still $e' \notin \mathrm{cs}(e)$ but $e' \notin \mathrm{cs}'(e)$, then by contraposition of Condition 5 we have $\nexists a \in C' \setminus C \,.\, a \blacktriangleright [e' \rightarrow e]$, and so $e' \notin \big(\mathrm{ic}(e) \cup \mathrm{ac}(C',e)\big) \setminus \big(\mathrm{dc}(C',e) \cup C\big)$. Let us now consider the case $e' \in \mathrm{cs}(e)$ and here first $e' \in \mathrm{cs}'(e)$. Then by Condition 4 (the *only if* condition) it follows $\nexists d \in C' \setminus C \,.\, [e' \rightarrow e] \rhd d$. Then $e' \notin \mathrm{dc}(C',e)$ and because $e \in E \setminus C'$ it follows $e \notin C$ and so $e' \notin \big(\mathrm{ic}(e) \cup \mathrm{ac}(C',e)\big) \setminus \big(\mathrm{dc}(C',e) \cup C\big)$. In the last case we consider $e' \in \mathrm{cs}(e)$ and $e' \notin \mathrm{cs}'(e)$. By Condition 4 (the *if* condition) we have $\exists d \in C' \setminus C \,.\, [e' \rightarrow e] \rhd d$ and so $e' \in \mathrm{dc}(C',e)$. Thus $e' \notin \big(\mathrm{ic}(e) \cup \mathrm{ac}(C',e)\big) \setminus \big(\mathrm{dc}(C',e) \cup C\big)$. So in each case $\mathrm{cs}'(e) = \big(\mathrm{ic}(e) \cup \mathrm{ac}(C',e)\big) \setminus \big(\mathrm{dc}(C',e) \cup C\big)$ holds. □

Lemma 3.5.9 states:

> Let ρ be a SSDC and let (C, cs) and (C', cs') be two states of ρ with $C \subseteq C'$, then Conditions 4 and 5 of $\rightarrow_{\mathrm{d}}$ hold for those two states.

Proof. Let us prove Condition 4. Let $e' \in E \setminus C'$ with $e' \in \mathrm{cs}(e) \setminus \mathrm{cs}'(e)$. Since $\mathrm{ac}(C,e) = \{e' \mid \exists a \in C \,.\, a \blacktriangleright [e' \rightarrow e] \wedge a \notin \{e, e'\}\}$ and $\mathrm{dc}(C,e) = \{e' \mid \exists d \in C \,.\, [e' \rightarrow e] \rhd d\}$ and $C \subseteq C'$, we have: $\mathrm{ac}(C,e) \subseteq \mathrm{ac}(C',e)$ and $\mathrm{dc}(C,e) \subseteq \mathrm{dc}(C',e)$ and by the previous Lemma 3.5.8 we have:

$$e' \in \Big(\big(\mathrm{ic}(e) \cup \mathrm{ac}(C,e)\big) \setminus \big(\mathrm{dc}(C,e) \cup C\big)\Big) \setminus \Big(\big(\mathrm{ic}(e) \cup \mathrm{ac}(C',e)\big) \setminus \big(\mathrm{dc}(C',e) \cup C'\big)\Big)$$

In other words:

$$e' \in \big(\mathrm{ic}(e) \cup \mathrm{ac}(C,e)\big) \wedge e' \notin \big(\mathrm{dc}(C,e) \cup C\big)$$
$$\wedge\ e' \notin \Big(\big(\mathrm{ic}(e) \cup \mathrm{ac}\big(C',e\big)\big) \setminus \big(\mathrm{dc}\big(C',e\big) \cup C'\big)\Big)$$

But if $e' \in \mathrm{ic}(e) \cup \mathrm{ac}(C,e)$ then $e' \in \mathrm{ic}(e) \cup \mathrm{ac}\big(C',e\big)$, thus $e' \in \big(\mathrm{dc}\big(C',e\big) \cup C'\big)$ and so $e' \in \mathrm{dc}\big(C',e\big)$, but $e' \notin \mathrm{dc}(C,e)$, which yields $[e' \to e] \rhd \cap \big(C' \setminus C\big) \neq \emptyset$, so the *if* condition of 4 holds. Let now $e,e' \in E \setminus C'$ with $[e' \to e] \rhd \cap \big(C' \setminus C\big) \neq \emptyset$, then $e' \in \mathrm{dc}\big(C',e\big)$ so $e' \notin \mathrm{cs}'(e)$ follows, which is exactly the *only if* condition of 4.

Condition 5 is proved similarly. □

Lemma 3.5.10 states:

> Let ρ be a SSDC and $(X,\mathrm{cs}_X) \to_{\mathrm{d}} (Y,\mathrm{cs}_Y)$ a transition in ρ. Then for all X',Y' with $X \subseteq X' \subseteq Y' \subseteq Y$, there is a transition $\big(X',\mathrm{cs}'_X\big) \to_{\mathrm{d}} \big(Y',\mathrm{cs}'_Y\big)$ in ρ, where $\mathrm{cs}_{X'}(e) = \Big(\big(\mathrm{ic}(e) \cup \mathrm{ac}\big(X',e\big)\big) \setminus \big(\mathrm{dc}\big(X',e\big) \cup X'\big)\Big)$ and $\mathrm{cs}_{Y'}(e) = \Big(\big(\mathrm{ic}(e) \cup \mathrm{ac}\big(Y',e\big)\big) \setminus \big(\mathrm{dc}\big(Y',e\big) \cup Y'\big)\Big)$.

Proof. By assumption Conditions 1 and 2 of Definition 3.5.3 of the transition relation holds for the two states $\big(X',\mathrm{cs}_{X'}\big)$ and $\big(Y',\mathrm{cs}_{Y'}\big)$. Conditions 4, and 5 follow from Lemma 3.5.9. Condition 6 holds because of Definition 3.5.7 and ρ is a SSDC. Condition 7 holds because it is a special case of the same conditions for (X,cs_X) and (Y,cs_Y). Let now $e \in Y' \setminus X'$, such that $\mathrm{cs}(e)_{X'} \neq \emptyset$, then there is $a \in X' \setminus X$ and a $c \in E$ with $a \blacktriangleright [c \to e]$, but this is a contradiction with Condition 7 of $(X,\mathrm{cs}_X) \to_{\mathrm{d}} (Y,\mathrm{cs}_Y)$, so Condition 3 holds. Thus $\big(X',\mathrm{cs}'_X\big) \to_{\mathrm{d}} \big(Y',\mathrm{cs}'_Y\big)$ holds. □

Lemma 3.5.12 states:

> Let σ be a SSDC, $H \in \mathrm{C}(\sigma)$. Then:
>
> $$\forall X \subseteq E \setminus H\,.\, X \in \mathrm{C}(\sigma[H]) \iff H \to_{\mathrm{c}}^{*} H \cup X.$$

Proof. Let $\Delta[H] = \big(E',\#',\to',\rhd',\blacktriangleright'\big)$.

First: $X \in \mathrm{C}(\sigma[H]) \implies H \to_{\mathrm{c}}^{*} H \cup X$ for $X \subseteq E \setminus H$.
To prove that, let (X_1,cs_1), (X_2,cs_2) be two states in $\Delta[H]$ for $X_1,X_2 \subseteq E'$.

Let $\mathrm{cs}'_1 : E' \setminus X_1 \to E' \setminus X_1$ be a causality state function such that $\mathrm{cs}'_1(e) = \mathrm{cs}_1(e) \setminus \{e\}$ for $e \in \{e \mid \exists e' \in H \,.\, e\#e' \land \neg(e \to e)\}$, and $\mathrm{cs}'_1(e) = \mathrm{cs}_1(e)$ otherwise. Let cs'_2 be defined similarly according to cs_2. Let us prove the following auxiliary implication:

$$(X_1, \mathrm{cs}_1) \to'_{\mathrm{d}} (X_2, \mathrm{cs}_2) \land (X_1, \mathrm{cs}_1) \in \mathrm{S}(\Delta') \implies \left(H \cup X_1, \mathrm{cs}'_1\right) \to_{\mathrm{d}} \left(H \cup X_2, \mathrm{cs}'_2\right) \quad \text{(A.5)}$$

Let us prove first that $\left(H \cup X_1, \mathrm{cs}'_1\right)$ is a state in Δ.
Since $E' \setminus X_1 = E \setminus (H \cup X_1)$ then $\mathrm{cs}'_1 : E \setminus (H \cup X_1) \to \mathcal{P}(E \setminus (H \cup X_1))$.
Let us prove also that $\mathrm{cs}'_1(e) \subseteq \mathrm{mc}'(H \cup X_1, e)$ where
$\mathrm{mc}(H \cup X_1, e) = \{e' \in E \setminus (H \cup X_1) \mid e' \to e \lor \exists a \in H \cup X_1 \,.\, \left(e', a, e\right) \in \blacktriangleright\}$ is the maximal causality function in Δ. Let $e' \in \mathrm{cs}'_1(e)$, then $e' \in \mathrm{cs}_1(e)$ by definition of cs'_1. But $\mathrm{cs}_1(e) \subseteq \mathrm{mc}'(X_1, e)$ since (X_1, cs_1) is a state in $\Delta[S]$, where:

$$\mathrm{mc}'(X_1, e) = \{e' \in E' \setminus X_1 \mid e' \to' e \lor \exists a \in X_1 \,.\, \left(e', a, e\right) \in \blacktriangleright'\}$$

Then $e' \in \mathrm{mc}'(X_1, e)$. Then $e' \to' e$ or $\exists a \in X_1 \,.\, \left(e', a, e\right) \in \blacktriangleright'$. But $e' \to' e$ either means $e' \in \mathrm{cs}_H(e)$ which means $e' \in \mathrm{mc}(H, e)$, then $e' \in \mathrm{mc}(H \cup X_1, e)$ since $\mathrm{mc}(H, e) \subseteq \mathrm{mc}(H \cup X_1, e)$ by the definition of the maximal causality function; or it means $e' = e$ and $\exists e'' \in H \,.\, e''\#e$ which means $e' \to' e$ according to the definition of cs'_1, which in turn means $e' \in \mathrm{mc}(H \cup X_1, e)$. On the other hand $\exists a \in X \,.\, \left(e', a, e\right) \in \blacktriangleright'$ means $\exists a \in H \cup X_1 \,.\, \left(e', a, e\right) \in \blacktriangleright'$, which also means $\exists a \in H \cup X_1 \,.\, \left(e', a, e\right) \in \blacktriangleright$ since $\blacktriangleright' \subseteq \blacktriangleright \cap E'^3$; i.e. $e' \in \mathrm{mc}(H \cup X_1, e)$. Hence $\mathrm{cs}'_1(e) \subseteq \mathrm{mc}(H \cup X_1, e)$. So $(H \cup X_1, \mathrm{cs}_1)$ is a state in Δ. The same applies for $\left(H \cup X_2, \mathrm{cs}'_2\right)$.

Next, to prove the transition, conflict-freeness of Condition 2 is proved similar to Lemma 3.3.7 with the absense of droppers for cycles of events conflicting with H, since (X_1, cs_1) is reachable. Conditions 3 to 5 are proved straightforward and similarly, as well as Condition 7. Condition 6 holds for all SSDCs anyway.Then $\left(\emptyset, \mathrm{cs_i}'\right) \to^*_{\mathrm{d}} (X, \mathrm{cs}_x) \implies \left(H, \mathrm{cs}'\right) \to^*_{\mathrm{d}} \left(H \cup X, \mathrm{cs}'_x\right)$ for $X \subseteq E' = E \setminus H$. Then, by the definition of configurations and $\to_{\mathrm{c}}$ in DCES, we have:

$$\forall X \subseteq E \setminus H \,.\, X \in \mathrm{C}(\sigma[H]) \implies H \to^*_{\mathrm{c}} H \cup X \quad \text{(A.6)}$$

Second: similarly we prove the other condition:

$$(H \cup X_1, \mathrm{cs}_1) \to_{\mathrm{d}} (H \cup X_2, \mathrm{cs}_2) \implies \left(X_1, \mathrm{cs}'_1\right) \to'_{\mathrm{d}} \left(X_2, \mathrm{cs}'_2\right) \text{ for:}$$

- $X_1, X_2 \subseteq E \setminus H$

- $\mathrm{cs}_1 : E \setminus (H \cup X_1) \to \mathcal{P}(E \setminus (H \cup X_1))$
- $\mathrm{cs}_2 : E \setminus (H \cup X_2) \to \mathcal{P}(E \setminus (H \cup X_2))$

Furthermore, cs'_1 has the same domain and co-domain of cs_1 such that $\mathrm{cs}'_1(e) = \mathrm{cs}_1(e) \cup \{e\}$ for $e \in \{e \mid \exists e' \in H.\, e' \# e$, and $\mathrm{cs}'_1(e) = \mathrm{cs}_1(e)$ otherwise. cs'_2 is defined similarly. Then we conclude:

$$\forall X \subseteq E \setminus H.\, (H, \mathrm{cs}_H) \to_{\mathrm{d}}^{*} (H \cup X, \mathrm{cs}_x) \implies \left(\emptyset, \mathrm{cs}'_1\right) \to_{\mathrm{d}}^{\prime *} \left(X, \mathrm{cs}'_x\right)$$

But we can notice that in this case $\mathrm{cs}'_1 = \mathrm{cs}_{\mathrm{i}}{}'$, which is the causality state function for the initial state in $\Delta[H]$. Then:

$$\forall X \subseteq E \setminus H.\, H \to_{\mathrm{c}}^{*} H \cup X \implies X \in \mathrm{C}(\Delta[H]) \tag{A.7}$$

Hence, from A.6 and A.7 we conclude: $X \in \mathrm{C}(\sigma[H]) \iff H \to_{\mathrm{c}}^{*} H \cup X$ holds for $X \subseteq E \setminus H$. □

Lemma 3.5.14 states:

> Let σ be a SES and $\mathrm{i}(\sigma)$ its embedding. Then we have for each state $\left(C, \mathrm{cs}\right)$ of $\mathrm{i}(\sigma)$, $\mathrm{cs}(e) = \mathrm{ic}(e) \setminus \left(\mathrm{dc}(C, e) \cup C\right)$.

Proof. By Lemma 3.5.8 and because $\mathrm{ac}(C, e) = \emptyset$ in $\mathrm{i}(\sigma)$ for all configurations C and events e. □

Lemma 3.5.15 states:

> Let γ be a GES and $\mathrm{i}(\gamma)$ its embedding. Then we have for each state (C, cs) of $\mathrm{i}(\gamma)$, $\mathrm{cs}(e) = \left(\mathrm{ic}(e) \cup \mathrm{ac}(C, e)\right) \setminus C$.

Proof. By Lemma 3.5.8 and because $\mathrm{dc}(C, e) = \emptyset$ in $\mathrm{i}(\gamma)$ for all configurations C and events e. □

Lemma 3.5.16 states:

> Let μ be a GES or SES, then we have $\mathrm{i}(\mu) \simeq_{\mathrm{t}} \mu$.

Proof. Let μ be a SES and $C \to_{\mathrm{s}} C'$ a transition in μ, from Lemma 3.5.14 we have that for a configuration C the causality state function is defined as $\mathrm{cs} : E \setminus X \to \mathcal{P}(E \setminus X)$ where $\mathrm{cs}(e) = \mathrm{ic}(e) \setminus \left(\mathrm{dc}(X, E) \cup X\right)$. Based on Definition 3.3.3 C' is conflict free and $C \subseteq C'$ since $C \to_{\mathrm{s}} C'$, so Conditions 1 and 2 of Definition 3.5.3 are satisfied. Moreover in the configuration C we

have $\mathrm{cs}(e) = \mathrm{ic}(e) \setminus (\mathrm{dc}(C,E) \cup C)$ but $\mathrm{ic}(e) \setminus \mathrm{dc}(C,E) \subseteq C$, so Condition 3 holds. Conditions 4 and 5 hold since $\mathrm{i}(\mu)$ is a SSDC. Conditions 6, and 7 are trivially satisfied since $\blacktriangleright = \emptyset$, so $(C,\mathrm{cs}) \rightarrow_{\mathrm{d}} (C',\mathrm{cs}')$. Let now $(C,\mathrm{cs}) \rightarrow_{\mathrm{d}} (C',\mathrm{cs}')$ in $\mathrm{i}(\mu)$, then by Definitions 3.5.3, 3.3.3 in combination with Lemma 3.5.14 there is a transition $C \rightarrow_{\mathrm{s}} C'$ in μ.
The proof is similar for a GES. □

Embedding EBESs into DCESs

Lemma 3.5.19 states:

> Let ϑ be a EBDC, $C \in \mathcal{C}(\vartheta)$, and let $e, e' \in C \,.\, e <_C e'$. Let also $(C_0, cs_0) \rightarrow_{\mathrm{d}} \ldots \rightarrow_{\mathrm{d}} (C_n, cs_n)$ with $C_0 = \emptyset$ and $C_n = C$ be the transition sequence of C, then $\exists C_i \in \{C_0, \ldots, C_n\} \,.\, e \in C_i \wedge e' \notin C_i$.

Proof. Let (C_f, cs_f) be the first occurrence of e in the sequence $(C_0, cs_0) \rightarrow_{\mathrm{d}} \ldots \rightarrow_{\mathrm{d}} (C_n, cs_n)$, so according to Condition 1 of Definition 3.5.3 it is enough to prove that $e' \notin C_f$. First, assume that $e \rightarrow e'$, then $e \in cs_0(e')$ according to the definition of $\mathrm{cs_i}$. Then according to Definition 3.5.3 the only situation where $e \notin cs_{f-1}(e')$ is that there is a dropper $e'' \in C_{f-1}$ for it according to Condition 4, but that is impossible since e and e' will be in conflict according to Condition 2 of Definition 3.5.17. So $e \in cs_{f-1}(e')$ and thus $e' \notin C_f$ according to Condition 3 of Definition 3.5.3.

Second assume that $e' \blacktriangleright [e \rightarrow e]$. If $e' \in C_{f-1}$ then according to Condition 5 of Definition 3.5.3 $e \in cs_{f-1}(e)$ which means $e \notin C_f$ according to Condition 3 of Definition 3.5.3, which is a contradiction to the definition of C_f. Then according to Condition 7 of Definition 3.5.3, if $e' \in C_f$, it follows $e \in C_{f-1}$, which again contradicts the definition of C_f. So because $e' \in C$, there is an $h > f$, such that $e' \in C_h$ but $e' \notin C_{h-1}$.

Third, assume $\exists c \in E \,.\, [c \rightarrow e'] \triangleright e$. Then since EBDC are a subclass of SSDC we have $\nexists a \in E \,.\, a \blacktriangleright [c \rightarrow e']$ according to Definition 3.5.7. Then $c \rightarrow e'$ according to Condition 1 of Definition 3.5.1, which means $c \in cs_0(e')$ according to definition of $\mathrm{cs_i}$ in Definition 3.5.3. Let us assume that $e' \in C_f$ then either c or another dropper $d \,.\, [c \rightarrow e'] \triangleright d$ occurred before e', which is impossible because of the mutual conflict in Condition 2 of Definition 3.5.17. So $e' \notin C_f$. □

Lemma 3.5.20 states:

> $\leq_C$ is a partial order over C.

Proof. Let $e, e' \in C\,.\, e <_C e'$ and let $(\emptyset = C_0, cs_0) \ldots (C_n = C, cs_n)$ be the transition sequence of C. Let also C_h, C_j be the configurations where e, e' first occur, respectively, then according to Lemma 3.5.19, $h < j$. Since $\leq_C$ is the reflexive and transitive closure of $<_C$, then $e \leq_C e' \implies h \leq j$. For antisymmetry, assume that $e' \leq_C e$ also then according to Lemma 3.5.19: $j \leq h$, but $h \leq j$, then $h = j$. The only possibility for $h = j$ is that $e = e'$ because otherwise $h < j$ and $j < h$, which is a contradiction. □

Lemma 3.5.21 states:

> Let ϑ be a EBDC and $(C, \mathrm{cs}), (C', \mathrm{cs}') \in \mathrm{S}(\vartheta)$ with $C \subseteq C'$. Then: $\big(\forall e, e' \in C'\,.\, e \neq e' \wedge e' \leq_{C'} e \implies e' \in C\big) \iff (C, \mathrm{cs}) \rightarrow_{\mathrm{d}} (C' \mathrm{cs}')$.

Proof. First let us assume $\forall e, e' \in C'\,.\, e \neq e' \wedge e' \leq_{C'} e \Rightarrow e' \in C$ and show that all the conditions of Definition 3.5.3 hold for (C, cs) and (C', cs'). Condition 1, $C \subseteq C'$, holds by assumption. Because C' is a configuration it is conflict free, i.e. Condition 2 holds. Conditions 4 and 5 follow immediately from Lemma 3.5.9. Condition 6 follows from Definition 3.5.7 of SSDC, since ϑ is an EBDC which is a subclass of SSDC. To prove Condition 3, let $f \in \big(C' \setminus C\big)$, then we have from Lemma 3.5.8 $\mathrm{cs}(f) = \big(\mathrm{ic}(f) \cup \mathrm{ac}(C, f)\big) \setminus \big(\mathrm{dc}(C, f) \cup C\big)$. Assume $\mathrm{cs}(f) \neq \emptyset$, i.e. $\exists f' \in \mathrm{cs}(f)$. So either $f' \in \mathrm{ic}(f)$ or $f \in \cup\,\mathrm{ac}(C, f)$. We can ignore the case that $f' \in \mathrm{ac}(C, f)$, because in EBDC the added causality for f can only be f, which would make f impossible, but this cannot be the case since $f \in C'$. So let us consider the remaining option: $f' \in \mathrm{ic}(f)$. Then $f' \leq_{C'} f$ by the definition of $\leq_{C'}$. Then by assumption, $f' \in C$ and therefore $f' \notin \mathrm{cs}(f)$, which is a contradiction. Then $\forall f \in (C' \setminus C)\,.\, \mathrm{cs}(f) = \emptyset$. For Condition 7 we show $\forall t, m \in C' \setminus C\,.\, \forall c \in E\,.\, m \blacktriangleright [c \rightarrow t] \implies c \in C$. The only growing causality is of the form $m \blacktriangleright [c \rightarrow c]$ and according to Definition 3.5.18, $m \blacktriangleright [c \rightarrow c]$ means $c \leq_{C'} m$, then $c \in C$.

Let us now assume $(C, \mathrm{cs}) \rightarrow_{\mathrm{d}} (C', \mathrm{cs}')$, and $e, e' \in C'$ with $e \neq e'$ and $e' \leq_{C'} e$, so by Lemma 3.5.19 it follows $e' \in C$. □

Lemma 3.5.23 states:

> Let ξ be an EBES. Then $\mathrm{dces}(\xi)$ is an EBDC.

Proof. First $\mathrm{dces}(\xi)$ is a DCES. The definition of $\rightarrow$ in Definition 3.3.11 ensures Conditions 3.5.1(1) and 3.5.1(2). According to the definition of $\triangleright$ in Definition 3.3.11, the only dropped causes are the fresh events, which cannot be added by $\blacktriangleright$ according to Definition 3.5.22(3). So Condition 3.5.1(3) also holds.

Second, dces(ξ) is a SSDC, since the only dropped events are the fresh ones which are never added by $\blacktriangleright$, so Definition 3.5.7 holds.

Third, dces(ξ) is a EBDC. Definition 3.5.17(1) holds by definition. Bundle members in ξ mutually disable each other, then according to the definition of $\#'$ Condition 3.5.17(2) holds. Therefore dces(ξ) is a EBDC. $\square$

Before comparing an EBES with its translation according to posets, we make use of the following lemma.

Lemma A.4.1. *Let* $\xi = (E, \rightsquigarrow, \mapsto)$ *be an EBES. Then* $\mathrm{C}(\xi) = \mathrm{C}(\mathrm{dces}(\xi))$.

Proof. First, $\forall c \subseteq E \,.\, c \in \mathrm{C}(\xi) \implies c \in \mathrm{C}(\mathrm{dces}(\xi))$. According to Section 2.5, $c \in \mathrm{C}(\xi)$ means there is a trace $t = e_1, \ldots, e_n$ in ξ such that $c = \bar{t}$. Let us prove that t corresponds to a transition sequence in dces(ξ) leading to c. i.e. let us prove that there exists a transition sequence $(\emptyset = c_0, cs_0) \to_{\mathrm{d}} \ldots \to_{\mathrm{d}} (c_n = c, cs_n)$ such that $c_i = c_{i-1} \cup \{e_i\}$ for $1 \leq i \leq n$, and cs_i is defined according to Lemma 3.5.8. This means we have to prove that $(c_{i-1}, cs_{i-1}) \to_{\mathrm{d}} (c_i, cs_i)$ for $1 \leq i \leq n$.

c_i is conflict-free since it is a configuration in ξ which means that it does not contain any mutual disabling according to Definition 2.5.2. Second, it is clear that $c_{i-1} \subseteq c_i$ by definition. Next, let us prove $\forall (e \in c_i \setminus c_{i-1}) \,.\, cs_{i-1} = \emptyset$, i.e. $\big(\mathrm{ic}(e) \cup \mathrm{ac}(c_{i-1}, e)\big) \setminus \mathrm{dc}(c_{i-1}, e) \subseteq c_{i-1}$ according to Lemma 3.5.8. ic(e) contains only fresh events according to the definition of dces(ξ), and the members of bundles $X_i \mapsto e$ are droppers of these fresh events. But since each of these bundles is satisfied, then each of these fresh events in ic(e) is dropped. Furthermore, there cannot be added causality in dces(ξ) for e except for e itself which makes it an impossible event, but it is not an impossible event since it occurs in a configuration. Therefore $\big(\mathrm{ic}(e) \cup \mathrm{ac}(c_{i-1}, e)\big) \setminus \big(\mathrm{dc}(c_{i-1}, e) \cup c_{i-1}\big) = \emptyset$ for all $e \in c_i$ and all $1 \leq i \leq n$. On the other hand, conditions 4) and 5) of 3.5.3 hold according to Lemma 3.5.9. Condition 6 of Definition 3.5.3 holds by Definition 3.5.7. Since in the transition $(c_{i-1}, cs_{i-1}) \to_{\mathrm{d}} (c_i, cs_i)$, only one event—namely e_i—occurs, then Definition 3.5.3(7) also holds.

In that way we proved that $\mathrm{C}(\xi) \subseteq \mathrm{C}(\mathrm{dces}(\xi))$. In a similar way, and with the help of Lemma 3.5.10, we can prove that $\mathrm{C}(\mathrm{dces}(\xi)) \subseteq \mathrm{C}(\xi)$ which means that $\mathrm{C}(\xi) = \mathrm{C}(\mathrm{dces}(\xi))$. $\square$

Lemma 3.5.24 states:

For each EBES ξ there is a DCES dces(ξ) such that

$$\mathrm{P}(\xi) = \mathrm{P}(\mathrm{dces}(\xi)).$$

Proof. First, $\forall p \in \mathrm{P}(\xi) . p \in \mathrm{P}(\mathrm{dces}(\xi))$. Let $p = (C, \leq_C)$, then $C \in \mathrm{C}(\xi)$ by the definition of posets of EBESs. Then according to Theorem A.4.1: $C \in \mathrm{C}(\mathrm{dces}(\xi))$. On the other hand, let $\leq'_C$ be the partial order defined for C in dces(ξ) as in Definition 3.5.18. This means that we should prove that $\leq_C = \leq'_C$. But since $\leq_C, \leq'_C$ are the reflexive and transitive closures of $\prec_C, <_C$ respectively, then it is enough to prove that $\prec_C = <_C$. In other words we have to prove $\forall e, e' \in C . e \prec_C e' \iff e' <_C e$.

Let us start with $e \prec_C e' \implies e <_C e'$. According to Definition 2.5.3 $e \prec_C e'$ means $\exists X \subseteq E . e \in X \mapsto e' \vee e \rightsquigarrow e'$. If $\exists X \subseteq E . e \in X \mapsto e'$ then $\exists c \in E' . \left[c \rightarrow e'\right] \rhd e$ by the definition of dces(ξ) Definition 3.5.22. This means $e <_C$ according to the definition of $<_C$ Definition 3.5.18. If $e \rightsquigarrow e'$ then $\neg e' \rightsquigarrow e$ since otherwise e, e' are in conflict. This means $e' \blacktriangleright [e \rightarrow e]$ according to Definition 3.5.22, which means $e <_C e'$ according to Definition 3.5.18.

Let us consider the other direction: $e <_C e' \implies e \prec_C e'$. $e <_C e'$ means $\exists c \in E' . \left[c \rightarrow e'\right] \rhd e \vee e' \blacktriangleright [e \rightarrow e]$ according to the definition of $<_C$ in Definition 3.5.18. The third option where $e \rightarrow e'$ is rejected since the only initial causes that exist in dces(ξ) are the fresh impossible events. If $\exists c \in E' . \left[c \rightarrow e'\right] \rhd e$ then $\exists X \subseteq E . e \in X \rightsquigarrow e'$ according to the definition of $\rhd$ in dces(ξ). This means $e \prec_C e'$ according to Definition 2.5.3. If on the other hand $e' \blacktriangleright [e \rightarrow e]$ then $e \rightsquigarrow e'$ according to the definition of dces(ξ), which means $e \prec_C$ according to Definition 2.5.3.

In that way we have proved that $\prec_C = <_C$, which means that $\leq_C = \leq'_C$. In a similar way we can prove that $\forall p \in \mathrm{P}(\mathrm{dces}(\xi)) . p \in \mathrm{P}(\xi)$, which means $\mathrm{P}(\xi) = \mathrm{P}(\mathrm{dces}(\xi))$. □

Expressiveness Of DCESs

Lemma A.4.2. *There is a DCES such that no EBES with the same configurations exits.*

Proof. We consider the embedding $\mathrm{i}(\sigma_\xi)$ (cf. Figure A.1) of the SES σ_ξ, which models disjunctive causality. According to Definitions 3.3.3 and 3.3.4, because $\neg(a \# c)$ and $\mathrm{ic}(a) = \mathrm{ic}(c) = \emptyset$, it holds $\emptyset \rightarrow_{\mathrm{s}} \{a, c\}$ and so $\{a, c\} \in$

$\mathrm{C}(\sigma_\xi)$. Further there is no transition $\emptyset \rightarrow_{\mathrm{s}} \{b\}$, because $\mathrm{ic}(b) = \{a\}$, but there are transitions $\{a\} \rightarrow_{\mathrm{s}} \{a,b\}$ and $\{c\} \rightarrow_{\mathrm{s}} \{c,b\}$, because $\mathrm{ic}(b) \setminus \mathrm{dc}(\{a\},b) \subseteq \{a\}$ ($\mathrm{ic}(b) \setminus \mathrm{dc}(\{c\},b) \subseteq \{c\}$ resp.). The transitions are translated to the embedding according to Lemma 3.5.16 and Definition 3.5.6 the same holds for the configurations.

If we now assume there is a EBES ξ with the configurations $\emptyset, \{a\}, \{c\}$, $\{a,c\}, \{a,b\}, \{b,c\}$ and $\{a,b,c\}$ then according to Definition 2.5.1 because there is no configuration $\{b\}$ there must be a non-empty bundle $X \mapsto b$ and caused by the the configurations $\{a,b\}, \{b,c\}$ this bundle X must contain a and c. Now the stability condition of Definition 2.5.1 implies $a \rightsquigarrow c$ and $c \rightsquigarrow a$, so a and c are in mutual conflict contradicting to the assumption $\{a,c\} \in \mathrm{C}(\xi)$. Thus there is no EBES with the same configurations as $\mathrm{i}(\sigma_\xi)$. □

Theorem 3.5.25 states:

DCESs are strictly more expressive than EBESs.

Proof of Theorem 3.5.25. Follows directly from Lemmas A.4.2 and 3.5.24. □

For the incomparability result between DCESs and RCESs, we give an RCES counterexample, which cannot be modeled by a DCES.

Lemma A.4.3. *There is no transition-equivalent DCES to ρ_γ (cf. Figure 3.6).*

Proof of Lemma A.4.3. Assume $\Delta = (E, \#, \rightarrow, \triangleright, \blacktriangleright)$ such that $\Delta \simeq_{\mathrm{t}} \rho_\gamma$. Then $\mathrm{C}(\Delta) = \mathrm{C}(\rho_\gamma)$. By Definition 3.5.3 and because of the configuration $\{a,b,c\} \in \mathrm{C}(\rho_\gamma)$, the events a, b, and c cannot be in conflict with each other, i.e. $\# \cap \{a,b,c\}^2 = \emptyset$. Moreover, because of the configurations $\{a\}, \{b\}, \{c\} \in \mathrm{C}(\rho_\gamma)$, there are no initial causes for a, b, or c, i.e. $\rightarrow \cap \{x \rightarrow y \mid y \in \{a,b,c\}\} = \emptyset$. Note that the relation $\triangleright$ cannot disable events. Finally, because of the configurations $\{a,c\}, \{b,c\} \in \mathrm{C}(\rho_\gamma)$, neither a nor b can add a cause (except of themselves) to c, i.e. $a \blacktriangleright [e \rightarrow c] \implies e = a$ and $b \blacktriangleright [e \rightarrow c] \implies e = b$ for all $e \in E$. Thus we have $\forall e, e' \in \{a,b,c\} \,.\, \neg(e \# e')$ and in the state $(\{a,b\}, cs)$ it follows $\mathrm{cs}(c) = \emptyset \subseteq \{a,b\}$. But then, by Definition 3.5.3, $(\{a,b\}, \mathrm{cs}) \rightarrow_{\mathrm{d}} (\{a,b,c\}, \mathrm{cs}')$ for some causal state functions cs and cs′. Since $\neg(\{a,b\} \rightarrow_{\mathrm{rc}} \{a,b,c\})$, this violates our assumption, i.e. there is no DCES that is transition equivalent to ρ_γ. □

Theorem 3.5.26 states:

DCESs and RCESs are incomparable.

Proof of Theorem 3.5.26. The Theorem follows from Lemma A.4.3 and the order insensitivity of the RCESs: There is no RCES with the same behavior as the DCES $\big(\{a,t,c,d\},\ \emptyset,\ \emptyset,\ \{[c\to t]\rhd d\},\ \{a\blacktriangleright[c\to t]\}\big)$, since the transition behavior of RCESs is configuration based (Definition 2.7.2) and therefore it cannot behave differently w.r.t. the events in the configuration $\{a,d\}$. □

Theorem 3.5.27 states:

DCESs are strictly more expressive than GESs and SESs.

Proof of Theorem 3.5.27. By Theorems 3.5.26, 3.4.10, and 3.3.20 and Lemmas 3.5.16 and 3.5.16. □

A.5. Proofs of Section 4.4: Inferring Changes from Evolution

Lemma 4.4.2 states:

The internalization inter(v,δ_1,H,δ_2) is a SSDC.

Proof. First, inter(v,δ_1,H,δ_2) is a DCES.
Let us prove that $(E,\#,\to)$ is a PES according to Definition 3.2.1. Since $C_1\subseteq\#_1$ and $\#_1\subseteq E_1^2$ and $E_1\subseteq E$ then $C_1\subseteq E^2$. The same applies for C_2 and C_3. Hence, $\#\subseteq E^2$. To prove that # is irreflexive, assume the opposite, i.e. $e\#e$ for $e\in E$, which means $(e,e)\in\#$. This means that either $(e,e)\in C_1$, or $(e,e)\in C_2$ or $(e,e)\in C_3$. But $(e,e)\in C_1$ or $(e,e)\in C_2$ means $(e,e)\in\#_1$ from the definition of C_1 and C_2. Besides, $(e,e)\in C_3$ means $(e,e)\in\#_2$. This means that either $e\#_1e$ or $e\#_2e$. But that is a contradiction since both $\#_1$ and $\#_2$ are irreflexive. Hence $\neg(e\#e)$, i.e. # is irreflexive.

To prove that # is symmetric, assume $e\#e'$ and let us prove $e'\#e$. But $e\#e'$ means either $(e,e')\in C_1$, or $(e,e')\in C_2$ or $(e,e')\in C_3$.
$(e,e')\in C_1$ means $e\#_1e'\wedge(e,e')\in(E_1\setminus E_2)^2$. Then $e'\#_1e\wedge(e',e)\in(E_1\setminus E_2)^2$ since $\#_1$ is symmetric. Then $(e',e)\in C_1$ according to the definition of C_1. Then $(e',e)\in\#$.

$(e,e') \in C_2$ means $e \#_1 e' \wedge e \#_2 e'$. Then $e' \#_1 e \wedge e' \#_2 e$ since $\#_1$, $\#_2$ are symmetric. Then $(e',e) \in C_2$, which means $(e',e) \in \#$.
$(e,e') \in C_3$ means $e \#_2 e' \wedge (e,e') \notin E_1^2$. Then $e' \#_2 e \wedge (e',e) \notin (e',e) \notin E_1^2$. Then $(e',e) \in C_3$, i.e. $e' \# e$. Hence $\#$ is symmetric.
Since $\rightarrow_1 \subseteq E_1^2$, $v \in E$, $H \subseteq E$ and $E_1, E_2 \subseteq E$ then $\rightarrow \subseteq E^2$. Hence, $(E, \#, \rightarrow)$ is a PES. Additionally, it can be easily seen that $\triangleright, \blacktriangleright \subseteq E^3$.

Now let us prove that Property 1 of Definition 3.5.1 holds. Assume that $[c \rightarrow t] \triangleright d$ for $c,d,t \in E$. Then $d = v$ and $(e,e') \in (\rightarrow_1 \setminus \rightarrow_2)$ according to the definition of $\triangleright$. Then $e \rightarrow_1 e'$, which means that $e \rightarrow e'$ according the definition of $\rightarrow$. By that we have proved the following:

$$\forall c,d,t \in E \,.\, [c \rightarrow t] \triangleright d \implies c \rightarrow t \tag{A.8}$$

This means Property 1 holds.

To prove that Property 2 of Definition 3.5.1 holds, assume that $a \blacktriangleright [c \rightarrow t]$ for $a,c,t \in E$. Then either $(c,a,t) \in D$ or $(c,a,t) \in N$, or $(c,a,t) \in R$.
$(c,a,t) \in D$ means $c = t$ and $c \in (E_1 \setminus E_2) \wedge \neg(c \rightarrow_1 t)$. Then $\neg(c \rightarrow_1 t)$ and $t \neq v$ and $c \neq v$ since $v \notin E_1$ and $c = t \in E_1$. Then $(c,t) \notin \rightarrow_1$ and $(c,t) \notin I_1$ and $(c,t) \notin I_2$. Then $\neg(c \rightarrow t)$.
Assume that $(c,a,t) \in N$, then $(c,t) \in (\rightarrow_2 \setminus \rightarrow_1)$. In other words $\neg(c \rightarrow_1 t)$ and $c,t \in E_2$. Then $\neg(c \rightarrow_1 t)$ and $t \neq v$ and $c \neq v$. Then $\neg(c \rightarrow t)$.
Assume that $(c,a,t) \in R$, then $c = v$ and $(a,t) \in \#_1 \setminus \#_2$, which means that $c = v$ and $a,t \in E_1$. Then clearly $\neg(c \rightarrow_1 t)$ since $\rightarrow_1 \subseteq E_1^2$ and $v \notin E_1$. On the other hand, $(c,t) \notin I_1$ since $c = v$ and $v \notin H$. Similarly we find that $(c,t) \notin I_2$, which means $\neg(c \rightarrow t)$. Hence, we conclude the following:

$$\forall c,d,t \in E \,.\, a \blacktriangleright [c \rightarrow t] \implies \neg(c \rightarrow t) \tag{A.9}$$

This means Property 2 holds.

To prove Property 3 of Definition 3.5.1, assume $m \blacktriangleright [c \rightarrow t]$ for $c,m,t \in E$. Then $\neg(c \rightarrow t)$ according to A.9. Then $\neg([c \rightarrow t] \triangleright d)$ for all $d \in E$, according to A.8. In other words:

$$\forall c,a,t \in E \,.\, a \blacktriangleright [c \rightarrow t] \implies \nexists d \in E \,.\, [c \rightarrow t] \triangleright d \tag{A.10}$$

Then $\neg([c \rightarrow t] \triangleright m)$. Then Property 3 holds.

On the other hand, if we assume $[c \rightarrow t] \triangleright m$ for $c,m,t \in E$, then we would have $c \rightarrow t$ according to A.8. Then $\neg(a \blacktriangleright [c \rightarrow t])$ for all $a \in E$, according to A.9. In other words:

$$\forall c,d,t \in E \,.\, [c \rightarrow t] \triangleright d \implies \nexists a \in E \,.\, a \blacktriangleright [c \rightarrow t] \tag{A.11}$$

From (A.10) and (A.11), and according to Definition 3.5.7 of SSDC, we conclude that $\mathrm{inter}(v,\delta_1,H,\delta_2)$ is a SSDC. □

Lemma 4.4.3 states:

> Let $\Delta = \mathrm{inter}(v,\delta_1,H,\delta_2) = (E,\#,\rightarrow,\triangleright,\blacktriangleright)$ be an internalization, and let $\rightarrow_{\mathrm{c}}$ be its configuration-transition relation. Let $\rightarrow_{\mathrm{p}_1}$ be the transition relation of δ_1. Then:
>
> $$\rightarrow_{\mathrm{p}_1} \cap [\mathrm{C}(\delta_1)]^2 = \rightarrow_{\mathrm{c}} \cap [\mathcal{P}(E \setminus \{v\})]^2 \cap [\mathrm{C}(\Delta)]^2$$

Proof. **First:** $\rightarrow_{\mathrm{p}_1} \cap[\mathrm{C}(\delta_1)]^2 \subseteq \rightarrow_{\mathrm{c}} \cap[\mathcal{P}(E \setminus \{v\})]^2 \cap [\mathrm{C}(\Delta)]^2$.
To prove that, let us define the causality state function $\mathrm{cs} : E \setminus X \to \mathcal{P}(E \setminus X)$ such that $\mathrm{cs}(e) = \left(\mathrm{ic}(e) \cup \mathrm{ac}(X,e)\right) \setminus \left(\mathrm{dc}(X,e) \cup X\right)$, and prove the following:

$$C \rightarrow_{\mathrm{p}_1} C' \implies (C,\mathrm{cs}) \rightarrow_{\mathrm{d}} \left(C',\mathrm{cs}'\right).$$

$C \rightarrow_{\mathrm{p}_1} C'$ means that $C \subseteq C'$ and $\forall e,e' \in C' .\ \neg\left(e \#_1 e'\right)$ according to Definition 3.2.3. Then $\left(e,e'\right) \notin C_1$ and $\left(e,e'\right) \notin C_2$ (cf. C_1, C_2, C_3 of Definition 4.4.1). Since $e,e' \in C' \subseteq E_1$, then $\left(e,e'\right) \notin C_3$. Hence, $\left(e,e'\right) \notin \#$. So Conditions 1 and 2 of Definition 3.5.3 hold.

Let us prove that $\forall e \in C' \setminus C .\ \mathrm{cs}(e) = \emptyset$.
By definition, $\mathrm{ic}(e) = \{e' \mid e' \rightarrow e\}$. From the defininition of $\rightarrow$ and since $e \in E_1$, then $\mathrm{ic}(e) = \{e' \mid e' \rightarrow_1 e\}$. But from Definition 3.2.3 we know that $e' \rightarrow_1 e$ means $e' \in C$ for all $e' \in E_1$. This means $\mathrm{ic}(e) \subseteq C$ for all $e \in C' \setminus C$. By definition, $\mathrm{ac}(C,e) = \{e' \mid \exists a \in C .\ a \blacktriangleright [e' \rightarrow e]\}$. Assume $\exists a \in C .\ a \blacktriangleright [e' \rightarrow e]$, then by looking at the definition of $\blacktriangleright$ in Definition 4.4.1, we conclude that $a \notin D \cup N$ since $a \in C \subseteq E_1$ and $v \notin E_1$. Then $\left(e',a,e\right) \in R$, but C' is conflict free, i.e. $\forall e,e' \in C' .\ \neg\left(e \#_1 e'\right)$, which means $\forall e \in C' .\ \nexists e' \in C .\ e' \#_1 e$. Then $\mathrm{ac}(C,e) = \emptyset$. Then $\forall e \in C' \setminus C .\ \left(\mathrm{ic}(e) \cup \mathrm{ac}(C,e)\right) \subseteq C$. Then $\forall e \in C' \setminus C .\ \mathrm{cs}(e) = \emptyset$. Then Condition 3 of Definition 3.5.3 holds. So far, we proved that there are no adders in C.

Similarly we prove that there are no adders in C'. Thus Conditions 6 and 7 of Definition 3.5.3 aslo hold. Similar to the proof of Lemma 3.5.9, other conditions of Definition 3.5.3 hold. Thus we conclude that $\rightarrow_{\mathrm{p}_1} \subseteq \rightarrow_{\mathrm{c}}$, and since $E_1 \subseteq E \setminus \{v\}$ then $\rightarrow_{\mathrm{p}_1} \subseteq \rightarrow_{\mathrm{c}} \cap [\mathcal{P}(E \setminus \{v\})]^2$.

Accordingly, configurations that are reachable from $\emptyset$ by $\rightarrow_{\mathrm{p}_1}$ are reachable from S_0 under $\rightarrow_{\mathrm{d}}$, since $\mathrm{ic}(e) = \mathrm{cs}(e)$ for all $e \in E$. Hence:

$$\rightarrow_{\mathrm{p}_1} \cap [\mathrm{C}(\delta_1)]^2 \subseteq \rightarrow_{\mathrm{c}} \cap [\mathcal{P}(E \setminus \{v\})]^2 \cap [\mathrm{C}(\Delta)]^2 \tag{A.12}$$

Second: $\rightarrow_{\mathrm{c}} \cap [\mathcal{P}(E \setminus \{v\})]^2 \cap [\mathrm{C}(\Delta)]^2 \subseteq \rightarrow_{\mathrm{p}_1} \cap [\mathrm{C}(\delta_1)]^2$.
Assume:

$$(C, C') \in \rightarrow_{\mathrm{c}} \cap [\mathcal{P}(E \setminus \{v\})]^2 \cap [\mathrm{C}(\Delta)]^2 \tag{A.13}$$

Let us start with proving $C, C' \subseteq E_1$, second we prove that $C \rightarrow_{\mathrm{p}_1} C'$ and finally we address reachability of C_1, C_2.

(A.13) means $\exists \mathrm{cs}, \mathrm{cs}'$ such that $(C, cs) \rightarrow_{\mathrm{d}} (C', cs')$, where $(C, cs), (C', cs') \in \mathrm{S}(\Delta)$ and $v \notin C \cup C'$. According to Lemma 3.5.8 and Condition 3 of Definition 3.5.3 we have $\big(\mathrm{ic}(e) \cup \mathrm{ac}(C, e)\big) \setminus \big(\mathrm{dc}(C, e) \cup C\big) = \emptyset$. Assume $\exists e \in E_2 \setminus E_1 \,.\, e \in C'$. Then since $v \notin C$ we have $\mathrm{dc}(C, e) = \emptyset$ according to the definition of $\triangleright$ in Definition 4.4.1. Then $\mathrm{ic}(e) \cup \mathrm{ac}(C, e) \subseteq C$, i.e. $\mathrm{ic}(e) \subseteq C$. But according to the definition of $\mathrm{ic}(e)$ and I_2 we have $v \in \mathrm{ic}(e)$, i.e. $v \in C$, which is a contradiction. So $C, C' \subseteq E_1$, i.e. $\rightarrow_{\mathrm{c}} \cap [\mathcal{P}(E \setminus \{v\})]^2 \cap [\mathrm{C}(\Delta)]^2 \subseteq [\mathcal{P}(E_1)]^2$.

Let us prove that $C \rightarrow_{\mathrm{p}_1} C'$. Since $(C, cs) \rightarrow_{\mathrm{d}} (C', cs')$ then $C \subseteq C'$. Additionally, we have $\forall e, e' \in C' \,.\, \neg\big(e \# e'\big)$, then $(e, e') \notin C_1$ and $(e, e') \notin C_2$ and $(e, e') \notin C_3$. Assume $e \#_1 e'$, then either $e \#_2 e'$ or $(e, e') \notin \#_2$. If $e \#_2 e'$ then $(e, e') \in C_2$ and $e \# e'$ which is a contradiction. If $(e, e') \notin \#_2$ then (v, e, e') , $(v, e', e) \in R$. This means $v \in \mathrm{ac}(C, e)$, and since $\mathrm{cs}(e) = \emptyset$ then $v \in \mathrm{dc}(C, e) \cup C$. And since $\mathrm{dc}(C, e) = \emptyset$, then $v \in C$, which is a contradiction. Then $\neg(e \#_1 e')$. Regarding causality, let us prove $\forall e \in C' \setminus C \,.\, \{e' \mid e' \rightarrow_1 e\} \subseteq C$. We proved that $\mathrm{dc}(C, e) = \emptyset$ for all $e \in E \setminus C$ and $\mathrm{ic}(e) \subseteq C$. From the definition of ic and since $\rightarrow_1 \subseteq \rightarrow$ we conclude $\forall e \in C' \setminus C \,.\, \{e' \mid e' \rightarrow_1 e\} \subseteq C$. Hence $C \rightarrow_{\mathrm{p}_1} C'$

Finally, configurations reachable from S_0 under $\rightarrow_{\mathrm{d}}$ are reachable from $\emptyset$ by $\rightarrow_{\mathrm{p}_1}$, since $\mathrm{ic}(e) = \mathrm{cs}(e)$ for all $e \in E$. Hence:

$$\rightarrow_{\mathrm{c}} \cap [\mathcal{P}(E \setminus \{v\})]^2 \cap [\mathrm{C}(\Delta)]^2 \subseteq \rightarrow_{\mathrm{p}_1} \cap [\mathrm{C}(\delta_1)]^2 \tag{A.14}$$

From (A.12) and (A.14) we conclude:

$$\rightarrow_{\mathrm{p}_1} \cap [\mathrm{C}(\delta_1)]^2 = \rightarrow_{\mathrm{c}} \cap [\mathcal{P}(E \setminus \{v\})]^2 \cap [\mathrm{C}(\Delta)]^2$$

□

Lemma 4.4.4 states:

> Let $\mathrm{inter}(v, \delta_1, H, \delta_2)$ be an internalization with a configuration-transition relation $\rightarrow_{\mathrm{c}}$. Then $H \rightarrow_{\mathrm{c}} H \cup \{v\}$.

Proof. Given that H is reachable from the last lemma, let (H, cs) be the state with H. Since Δ is a SSDC, then according to Lemma 3.5.8 we have

$\text{cs}(e) = \big(\text{ic}(e) \cup \text{ac}(H,e)\big) \setminus \big(\text{dc}(H,e) \cup H\big)$. Let $H' = H \cup \{v\}$, and let us define the function cs$'$ such that $\text{cs}'(e) = \big(\text{ic}(e) \cup \text{ac}\big(H',e\big)\big) \setminus \big(\text{dc}\big(H',e\big) \cup H'\big)$ and prove $(H,\text{cs}) \to_{\text{d}} \big(H',\text{cs}'\big)$. Clearly $H \subseteq H'$. Besides, from definition of # in Definition 4.4.1 we find that H' is conflict-free, given that H is conflict-free. $\text{ic}(v) = H$ according to $\to$. Since H is conflict-free w.r.t. $\#_1$, then $\nexists e,e' \in H\,.\,(v,e,e') \in R$. Then $\text{ac}(H,v) = \emptyset$. Then $\text{cs}(v) = \emptyset$ and Condition 3 from Definition 3.5.3 holds. Since $\big|H' \setminus H\big| = 1$, then Conditions 6 and 7 hold. The rest of conditions are proved to hold with such definition of cs, cs$'$ as in the proof of Lemma 3.5.9. Then $(H,\text{cs}) \to_{\text{d}} (H',\text{cs}')$, and thus $H \to_{\text{c}} H \cup \{v\}$. □

Lemma 4.4.6 states:

Let $\Delta = \text{inter}(v,\delta_1,H,\delta_2)$ be an internalization. Then:

$$\Delta[H \cup \{v\}] \simeq_{\text{t}} \delta_2[H].$$

Proof. Let $\to_{\text{p}_2}$ be the transition relation of $\delta_2' = \delta_2[H]$ and let $\to_{\text{c}}$ be the configuration-transition relation of $\Delta' = \Delta[H']$, where $H' = H \cup \{v\}$. We need to prove $\to_{\text{p}_2} \cap [\text{C}(\delta_2')]^2 = \to_{\text{c}} \cap [\text{C}(\Delta')]^2$. Let $\Delta = (E,\#,\to,\rhd,\blacktriangleright)$, and $\Delta' = (E',\#',\to',\rhd',\blacktriangleright')$. According to Definition 3.5.11, $E' = E \setminus H'$. Let also $\delta_2 = (E_2,\#_2,\to_2)$, and $\delta_2' = \big(E_2',\#_2',\to_2'\big)$, such that $E_2' = E_2 \setminus H$ according to Definition 3.2.4.

First: $\to_{\text{c}} \cap [\text{C}(\Delta')]^2 \subseteq \to_{\text{p}_2} \cap [\text{C}(\delta_2')]^2$.

Let us prove initially that $\forall (C,C') \in \to_{\text{c}} \cap [\text{C}(\Delta')]^2\,.\,C,C' \in \mathcal{P}(E_2')$. By deduction, let us assume $C \in \mathcal{P}(E_2')$ and prove $C' \in \mathcal{P}(E_2')$, then check the base case. To prove $C' \in \mathcal{P}(E_2')$ assume the opposite i.e. $\exists e \in C'\,.\,e \notin E_2'$. This means $\exists e \in C' \setminus C\,.\,e \notin E_2'$ according to the assumption, i.e. $\exists e \in C' \setminus C\,.\,e \notin E_2 \setminus H$, which implies $e \notin E_2 \vee e \in H$. But $C' \subseteq \big(E \setminus H'\big) = (E_1 \cup E_2) \setminus H$ by Definition 3.5.11, and we have $e \in C'$. So $e \notin E_2 \vee e \in H$ means $e \in E_1 \setminus E_2$, i.e. a dropped event. It can be proven easily that the remainder of a SSDC is a SSDC itself, i.e. Δ' is a SSDC. Then since $e \in C' \setminus C$, we have $\big(\text{ic}'(e) \cup \text{ac}'(C,e)\big) \setminus \big(\text{dc}'(C,e) \cup C\big) = \emptyset$ according to Condition 3 of Definition 3.5.3, i.e. $\big(\text{ic}'(e) \cup \text{ac}'(C,e)\big) \setminus \text{dc}'(C,e) \subseteq C$. Since $\rhd' \subseteq \rhd \cap E'^3$ by the definition of the remainder of a DCES, and since $v \notin E'$, then $\rhd' = \emptyset$. So $\text{dc}'(C,e) = \emptyset$, and hence we conclude the following:

$$\forall \big(C,C'\big) \in \to_{\text{c}} \cap [\text{C}(\Delta')]^2\,.\,\forall e \in C' \setminus C\,.\,\text{ic}'(e) \subseteq C \tag{A.15}$$

But $\mathrm{ic}'(e) = \{e' \in E' \mid e' \to' e\}$, and from the definition of $\to'$ in Definition 3.5.11 we know that $\mathrm{cs}(e) \subseteq \mathrm{ic}'(e)$, where $\mathrm{cs}(e) = \big(\mathrm{ic}(e) \cup \mathrm{ac}(H', e)\big) \setminus \big(\mathrm{dc}(H', e) \cup H'\big)$. Then from A.15 we conclude the following:

$$\forall (C, C') \in \big(\to_{\mathrm{c}} \cap [\mathrm{C}(\Delta')]^2\big) \,.\, \forall e \in C' \setminus C \,. \\ \big(\mathrm{ic}(e) \cup \mathrm{ac}(H', e)\big) \setminus \big(\mathrm{dc}(H', e) \cup H'\big) \subseteq C \quad \text{(A.16)}$$

From the definition of D and $\to$ in Definition 4.4.1 w.r.t. Δ we can find that $e \in \mathrm{ic}(e) \cup \mathrm{ac}(H', e)$, i.e. a self-loop. But $\mathrm{dc}(H', e) = \emptyset$ since $e \in E_1 \setminus E_2$, and we have $e \notin H'$, then $e \in \mathrm{cs}(e)$, i.e. $e \in \mathrm{ic}'(e)$. Then according to A.15 we have $e \in C$, which is a contradiction. Then $C' \subseteq \mathcal{P}(E'_2)$ and we know that $\emptyset \in \mathcal{P}(E'_2)$ as a base case.

Now let us prove $C \to_{\mathrm{c}} C' \wedge C, C' \in \mathrm{C}(\Delta') \implies C \to_{\mathrm{p}_2} C' \wedge C, C' \in \mathrm{C}(\delta'_2)$ for all $C, C' \subseteq E'$. First, $C \to_{\mathrm{c}} C'$ means $C \subseteq C'$ by the definition of $\to_{\mathrm{c}}$. Next, let us prove that $\forall e \in C' \setminus C, e' \in E'_2 \,.\, e' \to'_2 e \implies e' \in C$. If $e' \to'_2 e$ then $e' \to_2 e$, then $e' \notin \mathrm{dc}(H', e)$ and $e' \in \mathrm{ic}(e) \cup \mathrm{ac}(H', e)$ by the definition of $\to'_2$, $\triangleright$, $\to$, and $\blacktriangleright$ resp. But $e' \notin H'$, then according to A.16 we find $e' \in C$, for all $e' \to'_2 e$ and $e \in C' \setminus C$. Finally, let us prove conflict-freeness w.r.t. $\#'_2$. We have already $\forall e, e' \in C' \,.\, \neg\big(e \#' e'\big)$ since $C \to_{\mathrm{c}} C'$. But $\#' = \# \cap E'^3$ then $\neg\big(e \# e'\big)$ since $e, e' \in E'$. But $\neg\big(e \# e'\big)$ means $\big(e, e'\big) \notin C_3$, which means $\big(e, e'\big) \notin \#_2 \vee \big(e, e'\big) \in E_1^2$. Assume $\big(e, e'\big) \in \#_2$, then $\big(e, e'\big) \in E_1^2$ should hold, then by Limitation 4.1 assumption we have $e \#_1 e'$ since $e, e' \notin H$ and $e, e' \in E_1 \cap E_2$. But then we will have $e \#_1 e' \wedge e \#_2 e'$ which means $\big(e, e'\big) \in C_2$, i.e. $e \# e'$ by the definition of $\#$, and that is a contradiction. Then $\big(e, e'\big) \notin \#_2$, hence $\big(e, e'\big) \notin \#'_2$ for all $e, e' \in C'$. Then according to Definition 3.2.3 we have $C \to_{\mathrm{p}_2} C'$. Hence, if $C \in \mathrm{C}(\delta'_2)$ then $C' \in \mathrm{C}(\delta'_2)$ for all $C, C' \subseteq E'$ such that $C \to_{\mathrm{c}} C' \wedge C, C' \in \mathrm{C}(\Delta')$. Additionally, we know that $\emptyset \in \mathrm{C}(\Delta')$ and $\emptyset \in \mathrm{C}(\delta'_2)$, then by deduction $C, C' \in \mathrm{C}(\delta'_2)$, i.e. $C \to_{\mathrm{c}} C' \wedge C, C' \in \mathrm{C}(\Delta') \implies C \to_{\mathrm{p}_2} C' \wedge C, C' \in \mathrm{C}(\delta'_2)$ for all $C, C' \subseteq E'$.

Second: $\to_{\mathrm{p}_2} \cap\, [\mathrm{C}(\delta'_2)]^2 \subseteq\, \to_{\mathrm{c}} \cap\, [\mathrm{C}(\Delta')]^2$
can be proved similarly to Lemma 4.4.3 with taking into consideration the remainders. Hence, and according to the definition of $\simeq_{\mathrm{t}}$, we conclude that $\Delta' \simeq_{\mathrm{t}} \delta'_2$. □

B. Appendix: Alternative Partial Order Semantics for Dual Event Structures and Shrinking-Causality Event Structures

To show that DESs and SESs are not only behavioral equivalent ES models but are also closely related at the structural level, we consider the remaining four intentional partial order semantics for DESs of [48].

Liberal causality is the least restrictive notion of causality in [48]. Here each set of events from bundles pointing to an event e that satisfies all bundles pointing to e is a cause.

Definition B.1 (Liberal Causality). *Let $\delta = (E, \#, \mapsto)$ be a DES, $e_1, \ldots, e_n$ one of its traces, $1 \leq i \leq n$, and $X_1 \mapsto e_i$, …, $X_m \mapsto e_i$ all bundles pointing to e_i. A set U is a cause of e_i in $e_1, \ldots, e_n$ iff*

- $\forall e \in U \,.\, \exists 1 \leq j < i \,.\, e = e_j$
- $U \subseteq (X_1 \cup \ldots \cup X_m)$ *and*
- $\forall 1 \leq k \leq m \,.\, X_k \cap U \neq \emptyset$.

Let $\mathrm{P_{lib}}(t)$ *be the set of posets obtained this way for a trace t.*

Bundle satisfaction causality is based on the idea that for an event e in a trace each bundle pointing to e is satisfied by exactly one event in a cause of e.

Definition B.2 (Bundle Satisfaction Causality). *Let $\delta = (E, \#, \mapsto)$ be a DES, $e_1, \ldots, e_n$ one of its traces, $1 \leq i \leq n$, and $X_1 \mapsto e_i, \ldots, X_m \mapsto e_i$ all bundles pointing to e_i. A set U is a cause of e_i in $e_1, \ldots, e_n$ iff*

- $\forall e \in U \,.\, \exists 1 \leq j < i \,.\, e = e_j$ *and*

- *there is a surjective mapping $f : \{X_k\} \to U$ such that $f(X_k) \in X_k$ for all $1 \leq k \leq m$.*

Let $\mathrm{P}_{\mathrm{bsat}}(t)$ *be the set of posets obtained this way for a trace t.*

Minimal causality requires that there is no subset that is a cause itself.

Definition B.3 (Minimal Causality). *Let $\delta = (E, \#, \mapsto)$ be a DES and let $e_1, \ldots, e_n$ be one of its traces, $1 \leq i \leq n$, and $X_1 \mapsto e_i, \ldots, X_m \mapsto e_i$ all bundles pointing to e_i. A set U is a cause of e_i in $e_1, \ldots, e_n$ iff*

- $\forall e \in U \,.\, \exists 1 \leq j < i \,.\, e = e_j$
- $\forall 1 \leq k \leq m \,.\, X_k \cap U \neq \emptyset$ *and*
- *there is no proper subset of U satisfying the previous two conditions.*

Let $\mathrm{P}_{\mathrm{min}}(t)$ *be the set of posets obtained this way for a trace t.*

Late causality contains the latest causes of an event that form a minimal set.

Definition B.4 (Late Causality). *Let $\delta = (E, \#, \mapsto)$ be a DES, $e_1, \ldots, e_n$ one of its traces, $1 \leq i \leq n$, and $X_1 \mapsto e_i, \ldots, X_m \mapsto e_i$ all bundles pointing to e_i. A set U is a cause of e_i in $e_1, \ldots, e_n$ if*

- $\forall e \in U \,.\, \exists 1 \leq j < i \,.\, e = e_j$
- $\forall 1 \leq k \leq m \,.\, X_k \cap U \neq \emptyset$
- *there is no proper subset of U satisfying the previous two conditions, and*
- *U is the latest set satisfying the previous three conditions.*

Let $\mathrm{P}_{\mathrm{late}}(t)$ *be the set of posets obtained this way for a trace t.*

As derived in [48], it holds that

$$\mathrm{P}_{\mathrm{late}}(t), \mathrm{P}_{\mathrm{d}}(t) \subseteq \mathrm{P}_{\mathrm{min}}(t) \subseteq \mathrm{P}_{\mathrm{bsat}}(t) \subseteq \mathrm{P}_{\mathrm{lib}}(t)$$

for all traces t. Moreover a behavioral partial order semantics is defined and it is shown that two DESs have the same posets w.r.t. to the behavioral partial order semantics iff they have the same posets w.r.t. to the early partial order semantics iff they have the same traces.

Bundle satisfaction causality is (as the name suggests) closely related to the existence of bundles. In SESs there are no bundles. Of course, as shown by the translation $\mathrm{des}(\cdot)$ in Definition 3.3.8, we can transform the initial and dropped causes of an event into a bundle. Besides if we do so, a SES σ and its translation $\mathrm{des}(\sigma)$ have exactly the same families of posets obviously. On the other hand, because bundles are no native concept of SESs, we cannot directly map the definition of posets w.r.t. to bundle satisfaction to SESs.

To adapt the definitions of posets in the other three cases we have to replace the condition $U \subseteq (X_1 \cup \ldots \cup X_m)$ by

$$U \subseteq \left(\{e \mid e \rightarrow e_i \vee \exists e' \in E \,.\, [e' \rightarrow e_i] \rhd e\}\right)$$

and replace the condition $\forall 1 \leq k \leq m \,.\, X_k \cap U \neq \emptyset$ by $(\mathrm{ic}(e_i) \setminus \mathrm{dc}(U, e_i)) \subseteq U$ (as in Definition 3.3.9). The remaining conditions remain the same with respect to traces as defined in Definition 3.3.2. Let $\mathrm{P}_{\mathrm{lib}}(t)$, $\mathrm{P}_{\mathrm{min}}(t)$, and $\mathrm{P}_{\mathrm{late}}(t)$ denote the sets of posets obtained this way for a trace $t \in \mathrm{T}(\sigma)$ of a SES σ w.r.t. liberal, minimal, and late causality. Moreover, let $\mathrm{P}_x(\delta) = \bigcup_{t \in \mathrm{T}(\delta)} \mathrm{P}_x(t)$ and $\mathrm{P}_x(\sigma) = \bigcup_{t \in \mathrm{T}(\sigma)} \mathrm{P}_x(t)$ for all $x \in \{\mathrm{lib}, \mathrm{bsat}, \mathrm{min}, \mathrm{late}\}$.

Since the definitions of posets in DESs and SESs are very similar again, the translations $\mathrm{des}(\cdot)$ and $\mathrm{ses}(\cdot)$ preserve families of posets. The proof is very similar to the proofs of Theorems 3.3.10 and 3.3.13.

Theorem B.5. *For each SES σ there is a DES δ, namely $\delta = \mathrm{des}(\sigma)$, and for each DES δ there is a SES σ, namely $\sigma = \mathrm{ses}(\delta)$, such that $\mathrm{P}_x(\sigma) = \mathrm{P}_x(\delta)$ for all $x \in \{\mathrm{lib}, \mathrm{min}, \mathrm{late}\}$.*

Proof. The definitions of posets in DESs and SESs w.r.t. to minimal and late causality differ in exactly the same condition and its replacement as the definitions of posets in DESs and SESs w.r.t. early causality. Thus the proof in these two cases is similar to the proofs of Theorems 3.3.10 and 3.3.13.

If $\sigma = (E, \#, \rightarrow, \rhd)$ is a SES then, by Lemma A.2.2, $\delta = \mathrm{des}(\sigma) = (E, \#, \mapsto)$ is a DES such that $\mathrm{T}(\sigma) = \mathrm{T}(\delta)$ and $\mathrm{C}(\sigma) = \mathrm{C}(\delta)$. If $\delta = (E, \#, \mapsto)$ is a DES then, by Lemma A.2.4, $\sigma = \mathrm{ses}(\delta) = (E, \#, \rightarrow, \rhd)$ is a DES such that $\mathrm{T}(\delta) = \mathrm{T}(\sigma)$ and $\mathrm{C}(\delta) = \mathrm{C}(\sigma)$. In both cases let $t = e_1, \ldots, e_n \in \mathrm{T}(\sigma)$, $1 \leq i \leq n$, and $X_1 \mapsto e_i, \ldots, X_m \mapsto e_i$ be all bundles pointing to e_i.

In the case of liberal causality, for U to be a cause for e_i the definition of posets in SESs requires $U \subseteq \left(\{e \mid e \rightarrow e_i \vee \exists e' \in E \,.\, [e' \rightarrow e_i] \rhd e\}\right)$ and $(\mathrm{ic}(e_i) \setminus \mathrm{dc}(U, e_i)) \subseteq U$. The second condition holds iff $\forall 1 \leq k \leq m \,.\, X_k \cap U \neq \emptyset$ as shown in the proofs of Theorems 3.3.10 and 3.3.13. By Definitions 3.3.8 and 3.3.11, the first conditions holds iff $U \subseteq (X_1 \cup \ldots \cup X_m)$. So, by the definitions of posets in DESs and SESs w.r.t. to liberal causality, $\mathrm{P}_{\mathrm{lib}}(\sigma) = \mathrm{P}_{\mathrm{lib}}(\delta)$. □

C. Appendix: Application of Evolving Event Structures in Schema Evolution and Instance Migration in Business Processes

A schema might evolve while multiple instances of the old definition are running. Different ways to handle running instances of an evolving schema are available in the literature [80]. Mainly, such instances are either restarted with the new definition, or they are aborted, or migrated—when possible—to the new definition with a valid state based on their history. However, the migration is not always possible since an instance—with its state—might not fit to the new schema definition. Studies in the literature [65] provide different definitions of what is a valid state for the instance in the new definition. In this work, as in Chapter 4, we depend on history preservation as a criterion to migrate an instance to a new schema definition.

To that end, we define *schema-internalization* to model the trigger event v as an internal event in the schema, such that v is responsible for transforming the schema from its old state to its new one. The result is an ES Δ that is transition-equivalent to ES_1 before v occurs and to ES_2 after v occurs. Accordingly, ES_1 can be safely replaced by Δ for running instances as well as future ones. The difference to the internalization of Section 4.4 is that the event v in schema-internalization is independent from any history or system run of any instance. Additionally, v here can take place for a running instance if and only if it results in that instance migrating to ES_2.

Furthermore, in case v occurs initially in a schema-internalization, i.e. after the system run $\emptyset$ in Δ, then it represents a schema evolution that affects future system runs (i.e. future instances) where the remainder is transition-equivalent to δ_2. On the other hand, in case v takes place after a system run $H \neq \emptyset$ then it represents the migration of the instance (δ, H) to (δ_2, H), which is the evolution step of the running workflow (δ_1, H) addressed in Chapter 4. Therefore, by schema-internalization we generalize the two concepts of evolution, namely schema evolution and running workflow one.

We overload the notion of evolution steps to denote a schema evolution (step), such that $\delta_1 \xrightarrow{v} \delta_2$ denotes an evolution from δ_1 to δ_2 by v. Besides, we lift the internalization function inter to accept two independent ESs δ_1, δ_2 and an external event v representing the evolution trigger. We represent a schema evolution by the pair (δ_1, δ_2). Correctness of schema evolution is insured in the literature by maintaining certain properties e.g. deadlock freeness.

As in 4.4 we focus on PESs of Definition 3.2.1 to represent evolving structures. Besides, we add one contraint next to Constraint 4.1. Given two PESs $\delta_1 = (E_1, \#_1, \rightarrow_1)$, $\delta_2 = (E_2, \#_2, \rightarrow_2)$ and $\rightarrow_1'$ is the transitive closure of $\rightarrow_1$, it must hold:

$$\rightarrow_2 \cap E_1^2 \subseteq \rightarrow_1' \tag{C.1}$$

The last constraint states that no new causality between old events is added, unless it belongs to the transitive closure which does not change the set of configurations or even the transitions in a conjunctive-causality ES like a PES. Note that these limitations do not affect the idea of how to propagate changes to running instances in a unified way, each depending on its state.

Since events of a history H of an instance can have neither new conflicts in-between each other due to (4.1), nor new causality due to (C.1), it can be seen and proved that the only chances not to preserve H by δ_2 is:

1. Adding a new event as a causal predecessor to one of the events of H, or

2. Dropping one event of H

which are exactly the two cases where v should be disabled. Formally speaking, the internalization is defined as follows:

Definition C.1. *Let* $s = \left(\delta_1 \xrightarrow{v} \delta_2\right)$ *be a schema evolution step such that* $\delta_1 = (E_1, \#_1, \rightarrow_1)$, $\delta_2 = (E_2, \#_2, \rightarrow_2)$ *are two PESs satisfying (4.1) and (C.1), and* $v \notin E_1 \cup E_2$. *The* schema-internalization *of* v *into* δ_2 *w.r.t.* s *is defined as* $\text{sinter}(v, \delta_1, \delta_2) = (E, \#, \rightarrow, \triangleright, \blacktriangleright)$*, such that:*

- $E = E_1 \cup E_2 \uplus \{v\}$
- $\# = C_1 \cup C_2 \cup C_3$ *where:*
  - $C_1 := \#_1 \cap (E_1 \setminus E_2)^2$
  - $C_2 := \#_1 \cap \#_2$
  - $C_3 := \#_2 \setminus E_1^2$
- $\rightarrow = \rightarrow_1 \cup \{(v, e) \mid e \in E_2 \setminus E_1\}$
- $\triangleright = \left\{\left(e, v, e'\right) \mid \left(e, e'\right) \in \rightarrow_1 \setminus \rightarrow_2 \wedge e' \in E_2\right\}$
- $\blacktriangleright = D \cup N \cup R \cup F \cup O$ *where:*
  - $D := \left\{(e, v, e) \mid e \in (E_1 \setminus E_2) \wedge \neg(e \rightarrow_1 e)\right\}$ *to disable dropped events*
  - $N := \left\{\left(e, v, e'\right) \mid \left(e, e'\right) \in \rightarrow_2 \setminus \rightarrow_1\right\}$ *to add new causality*
  - $R := \left\{\left(v, e, e'\right) \mid \left(e, e'\right) \in \#_1 \setminus \#_2\right\}$ *to resolve conflicts*
  - $F := \left\{(v, e, v) \mid e \in E_1 \wedge \exists e' \in E_2 \setminus E_1 \,.\, e' \rightarrow e\right\}$ *to prevent adding fresh events to others that already happened*
  - $O := \{(v, e, v) \mid e \in E_1 \setminus E_2\}$ *to prevent dropping events that already happened*

Comparing to the internalization of Section 4.4 we have: E, $\#$, $\triangleright$ as well as D, N, R are defined similarly. On the other hand $\rightarrow$ matches I_2 only. However, the resulting structure of schema-internalization can be proved to be a SSDC (cf. Definition 3.5.7).

Lemma C.2. *The schema-internalization* $\text{sinter}(v, \delta_1, \delta_2)$ *is a SSDC.*

Proof. Similar to Lemma 4.4.2, taking into consideration that $\neg(v \rightarrow v)$ when proving (A.9). □

Next, we prove that the transitions of a schema-internalization resulting structure before v takes place are exactly the ones of δ_1, w.r.t. to reachable configurations.

Lemma C.3. *Let* $\Delta = \text{sinter}(v, \delta_1, \delta_2) = (E, \#, \rightarrow, \triangleright, \blacktriangleright)$ *be a schema-internalization, and let* $\rightarrow_{\mathrm{c}}$ *be its configuration-transition relation. Let* $\rightarrow_{\mathrm{p}_1}$ *be the transition relation of* δ_1*. Then:*

$$\rightarrow_{\mathrm{p}_1} \cap \, [\mathrm{C}(\delta_1)]^2 = \rightarrow_{\mathrm{c}} \cap \, [\mathcal{P}(E \setminus \{v\})]^2 \cap [\mathrm{C}(\Delta)]^2$$

Proof. Similar to Lemma 4.4.3, taking into consideration that v being initially enabled in $\rightarrow$ does not affect the proof since we do not consider v in the configurations here. The same applies for F, O. □

Furthermore, v can take place after a given system run $H \in \mathrm{C}(\Delta)$, if and only if H can be preserved by δ_2.

Lemma C.4. *Let* $\Delta = \mathrm{sinter}(v, \delta_1, \delta_2)$ *be a schema-internalization with a configuration-transition relation* $\rightarrow_{\mathrm{c}}$*, and let* $H \in \mathrm{C}(\Delta)$ *such that* $v \notin H$*. Then:*

$$H \rightarrow_{\mathrm{c}} H \cup \{v\} \iff H \in \mathrm{C}(\delta_2).$$

Proof. Proof of Lemma 4.4.4 covers $\Longleftarrow$ of this lemma. To prove the opposite direction $\Longrightarrow$, assume that $H \notin \mathrm{C}(\delta_2)$ then let us prove that $H \rightarrow_{\mathrm{c}} H \cup \{v\}$ does not hold. Let $\delta_2 = (E_2, \#_2, \rightarrow_2)$, and let (H, cs) be the state with $H \in \mathrm{C}(\Delta)$ as Δ is a SSDC according to Lemma C.2. In fact $H \notin \mathrm{C}(\delta_2)$ means that either $H \not\subseteq E_2$, or $H \subseteq E_2$ but $H \notin \mathrm{C}(\delta_2)$.

First, if $H \subseteq E_2$ and $H \notin \mathrm{C}(\delta_2)$, then according to Definition 3.2.2 H is conflict-free w.r.t. $\#_2$ since $H \in \mathrm{C}(\Delta)$ and there are no growing conflicts according to (4.1). Additionally, H is cycle-free due to (C.1) since $H \subseteq E_1$ according to Lemma C.3. Hence, H is not left closed w.r.t. $\rightarrow_2$. In other words, $\exists e' \in H \,.\, \exists e \in E_2 \,.\, e \rightarrow_2 e' \land e \notin H$. But $e \in E_1$ cannot hold due to (C.1), then $e' \in E_2 \setminus E_1$. Then according to the definition of F it holds that $e' \blacktriangleright [v \rightarrow v]$. Accordingly, $v \in \mathrm{cs}(v)$ since there is no dropper for $v \rightarrow v$ in $\triangleright$ of Definition C.1. Hence $\neg(H \rightarrow_{\mathrm{c}} H \cup \{v\})$ according to Property 3 of Definition 3.5.3.

Second, if $H \not\subseteq E_2$, then $\exists e \in H \,.\, e \in E_1 \setminus E_2$ since $H \in \mathrm{C}(\delta_1)$ according to Lemma C.3. Then according to the definition of O, it holds that $e \blacktriangleright [v \rightarrow v]$, which means again that $v \in \mathrm{cs}(v)$ and $\neg(H \rightarrow_{\mathrm{c}} H \cup \{v\})$. □

Additionally, if v can take place after a configuration $H \in \mathrm{C}(\Delta)$, then the remainder of Δ after $H \cup \{v\}$ is transition-equivalent to the remainder of δ_2 after H.

Lemma C.5. *Let* $\Delta = \mathrm{sinter}(v, \delta_1, \delta_2)$ *be a schema-internalization with a configuration-transition relation* $\rightarrow_{\mathrm{c}}$*. Let* $H \in \mathrm{C}(\Delta)$ *such that* $v \notin H$ *and* $H \rightarrow_{\mathrm{c}} H \cup \{v\}$*. Then:*

$$\Delta[H \cup \{v\}] \simeq_{\mathrm{t}} \delta_2[H].$$

Proof. Similar to Lemma 4.4.6. □

The next theorem shows that given a new schema definition δ_2, all instances of δ_1 can be migrated safely to schema Δ (the result of schema-internalization) which allows them to migrate to δ_2 if that does not contradict with their states. Otherwise, the instances will continue running as if they were in the old schema definition δ_1. Besides, Δ guarantees that new instances will run according to δ_2.

Theorem C.6. *Let $\delta_1 \xrightarrow{v} \delta_2$ be a schema evolution step, $\Delta = \mathrm{sinter}(v, \delta_1, \delta_2)$ be the internalization with the configuration-transition relation $\rightarrow_{\mathrm{c}}$. Let $\rightarrow_{\mathrm{p}_1}$ be the transition relation of δ_1, and let (δ_1, H) be an instance. Then:*

1. *(Δ, H) is an instance.*
2. *$\Delta[\{v\}] \simeq_{\mathrm{t}} \delta_2$.*
3. *$\delta_1 \xrightarrow[H]{v} \delta_2 \implies \Delta[H \cup \{v\}] \simeq_{\mathrm{t}} \delta_2[H]$.*
4. *$\neg\left(\delta_1 \xrightarrow[H]{v} \delta_2\right) \implies \rightarrow_{\mathrm{p}_1} = \left(\rightarrow_{\mathrm{c}} \cap [\mathcal{P}(E \setminus \{v\})]^2\right)$*

Proof. 1 and 4 are straightforward from Lemma C.3, while 2 and 3 are straightforward from Lemma C.5. □

Summary schema-internalization treats a schema evolution as an ordinary event for the instances of that schema. In this way, constraints over such an event can be placed regarding when it is enabled or disabled and how. This is then used to take a decision of when to migrate instances in a way that leads to no errors. Furthermore, this work helps building a complete transition graph for evolution of the Schema, and analyzes the dependency and conflict between different evolutions.

Universitätsverlag der TU Berlin

On the Foundations of Dynamic Coalitions

Dynamic Coalitions denote a temporary collaboration between different entities to achieve a common goal. This thesis studies workflows in Dynamic Coalitions, by analyzing their features, highlighting their unique characteristics and similarities to other workflows. For this purpose, we use the formal model of Event Structures and extend it in three different ways to faithfully model scenarios taken as use cases from healthcare. First, to cover the pre-planned changes in Dynamic-Coalition workflows, we present Dynamic Causality Event Structures which allows some events to change the causal dependencies of other events in a structure. Second, we present Evolving Event Structures that support ad-hoc and unforeseen changes in workflows and constrain such changes by the reachability of goals captured as events. Third, we add Priority to various kinds of Event Structures as a modeling feature for workflows in general that is posed as a requirement by the healthcare domain. Comparing to Adaptive Workflows, which are concerned with evolutions of workflows, this thesis shows that workflows in Dynamic Coalitions are not only Adaptive but also Goal-Oriented and that they might evolve due to the join of new members who contribute to goal satisfaction in a Dynamic Coalition.

ISBN 978-3-7983-2856-3 (print)
ISBN 978-3-7983-2857-0 (online)

ISBN 978-3-7983-2856-3

http://verlag.tu-berlin.de