Event Structure Semantics of Reversible Process Calculi

Eva Graversen, Department of Computing, Imperial College London
PhD thesis

July 27, 2021

Statement of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at Imperial College London or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at Imperial College London or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Copyright

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC). Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose. When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law

Abstract

In reversible computing a program can undo a set of past actions. This is useful in many areas including debugging, state-space exploration, and modelling naturally reversible systems. Concurrent reversible programs do not necessarily have a defined "most recent" action, and therefore rely on cause-respecting reversal, where an action can be reversed if it has not caused other actions. For this reason analysing the causal relationships between actions in reversible process calculi is important.

One framework used to describe causal relationships is event structures, which are a well-established model of true concurrency. There exist a number of forms of event structures, including prime, asymmetric, bundle, flow, stable, and general event structures. More recently, reversible forms of some these types of event structure have been defined. We formulate corresponding categories as well as functors and in some cases adjunctions between them. We show that products and coproducts exist in many cases.

CCSK is a reversible variant of CCS, which uses static reversibility, meaning a process retains the same structure throughout its computation and simply annotates past actions with keys. We use reversible bundle event structures to define denotational semantics of the reversible process calculus CCSK. CCSK is an uncontrolled reversible calculus, meaning there is no control on whether or when an action reverses and a process can therefore do and undo the same action indefinitely. We therefore modify CCSK to control the reversibility with a rollback primitive, which reverses a specific action and all actions caused by it. To define the event structure semantics of rollback, we exploit event structures' capacity for non-causal reversibility.

The π-calculus is a widely used process calculus, which models communications between processes and allows the passing of communication links. Various operational semantics of the π-calculus have been proposed, which can be classified according to whether transitions are unlabelled (so-called reductions) or labelled. With labelled transitions, we can distinguish early and late semantics. The early version allows a process to receive names it already knows from the environment, while the late semantics do not. All existing reversible versions of the π-calculus use reduction or late semantics, despite the early semantics of the (forward-only) π-calculus being more widely used than the late. We define πIH and πIK, the first reversible early π-calculi. The new calculi are a reversible form of the internal π-calculus, which is a subset of the π-calculus where every link sent by an output is private, yielding greater symmetry between inputs and outputs. We also define denotational event structure semantics of πIK and operational event structure semantics of πIH and prove an equivalence between the two.

Acknowledgements

During my PhD I have met and worked with many wonderful people who have contributed to my growth as a researcher and person over the past four years. I would like to thank both my supervisors, Nobuko Yoshida and Iain Phillips for their advice and support. They taught me much about research and how to communicate my work.

I also want thank Thomas Hildebrandt and Håkon Normann for welcoming me when I visited them. The discussions I had with them contributed greatly to Chapter 6 of this thesis.

I will miss all the friends I made through my time in MRG; Assel Altayeva, Simon Castellan, David Castro-Perez, Francisco Ferreira, Juju Gabet, Lorenzo Gheri, Raymond Hu, Sung-Shik Jongmans, Nicolas Lagaillardie, Julien Lange, Rumyana Neykova, Nick Ng, Domenico Ruoppolo, Alceste Scalas, Bernardo Toninho, and Fangyi Zhou. Their kindness, humour, and insistence on going to the pub every Friday kept me sane during the tough times. Thank you in particular to Francisco for sharing my basement and keeping me company when I was writing my thesis during lockdown.

Finally I want to thank my family: Lisbeth, Michael, Anna and Leica. Without them this PhD would never have gotten started.

My work was supported by the EPSRC DTP award.

Contents

1 Introduction **11**
- 1.1 Reversibility . . . 11
- 1.2 Process Calculi . . . 12
- 1.3 Event Structures . . . 14
- 1.4 Categories . . . 15
- 1.5 Contributions . . . 16
- 1.6 Publications . . . 16

2 Reversible Process Calculi **19**
- 2.1 Reversible CCS . . . 19
- 2.2 Reversible π-calculus . . . 21

3 Event Structures and Domains **25**
- 3.1 Domains . . . 26
- 3.2 Prime Event Structures . . . 27
- 3.3 Asymmetric Event Structures . . . 28
- 3.4 Bundle Event Structures . . . 31
- 3.5 Extended Bundle Event Structures . . . 34
- 3.6 Flow Event Structures . . . 36
- 3.7 General Event Structures . . . 38
- 3.8 Conclusion . . . 39

4 Reversible Event Structures and Configuration systems **41**
- 4.1 Configuration Systems . . . 43
- 4.2 Reversible Prime Event Structures . . . 49
- 4.3 Reversible Asymmetric Event Structures . . . 55
- 4.4 Reversible Bundle Event Structures . . . 58
- 4.5 Reversible Extended Bundle Event Structures . . . 63
- 4.6 Reversible Flow Event Structures . . . 67
- 4.7 Reversible General Event Structures . . . 71
- 4.8 Reversible Stable Event Structures . . . 77
- 4.9 Cause-Respecting and Causal RESs and CSs . . . 83
- 4.10 Conclusion . . . 86

5 Event structure semantics of (controlled) reversible CCS **87**
- 5.1 Event Structure Semantics of CCSK . . . 88
- 5.2 Roll-CCSK . . . 93
- 5.3 Event Structure Semantics of Roll-CCSK . . . 99
- 5.4 Conclusion . . . 107

6 Event structures for the reversible early internal Pi-calculus **109**
6.1 πIK-Calculus . . . 112
6.2 πIH-Calculus . . . 116
6.3 Mapping Between πIK-Calculus and πIH-Calculus . . . 124
6.4 Denotational Event Structure Semantics of πIK-Calculus . . . 127
6.5 Operational Event Structure Semantics of πIH-Calculus . . . 131
6.6 Correspondence Between Denotational and Operational Event Structure Semantics . . . 134
6.7 Conclusion . . . 140

7 Conclusion **143**
7.1 Future Work . . . 145

8 Notation **147**

Bibliography **150**

A Proofs from Chapter 3 **157**
A.1 Proposition 3.3.9 . . . 157
A.2 Proposition 3.4.6 . . . 157
A.3 Proposition 3.4.7 . . . 157
A.4 Proposition 3.4.10 . . . 158
A.5 Proposition 3.4.14 . . . 159
A.6 Propositions 3.5.5 . . . 159
A.7 Proposition 3.5.7 . . . 159
A.8 Proposition 3.7.8 . . . 160

B Proofs from Chapter 4 **161**
B.1 Proposition 4.1.4 . . . 161
B.2 Proposition 4.1.6 . . . 161
B.3 Proposition 4.1.8 . . . 162
B.4 Proposition 4.1.10 . . . 162
B.5 Proposition 4.1.12 . . . 163
B.6 Proposition 4.1.16 . . . 163
B.7 Proposition 4.1.18 . . . 164
B.8 Proposition 4.2.3 . . . 164
B.9 Proposition 4.2.5 . . . 165
B.10 Proposition 4.2.9 . . . 165
B.11 Proposition 4.2.15 . . . 166
B.12 Proposition 4.3.8 . . . 166
B.13 Proposition 4.3.3 . . . 167
B.14 Proposition 4.3.11 . . . 168
B.15 Proposition 4.3.16 . . . 168
B.16 Proposition 4.3.20 . . . 169
B.17 Proposition 4.4.5 . . . 169
B.18 Proposition 4.4.15 . . . 170
B.19 Proposition 4.4.18 . . . 171
B.20 Proposition 4.7.4 . . . 171
B.21 Proposition 4.7.7 . . . 171
B.22 Proposition 4.7.9 . . . 172
B.23 Proposition 4.7.12 . . . 172
B.24 Proposition 4.7.15 . . . 173
B.25 Theorem 4.7.17 . . . 174
B.26 Proposition 4.7.18 . . . 175
B.27 Proposition 4.7.20 . . . 175
B.28 Proposition 4.7.22 . . . 175

B.29 Theorem 4.7.23 . . . 176
B.30 Proposition 4.8.6 . . . 176
B.31 Proposition 4.8.9 . . . 176
B.32 Proposition 4.8.12 . . . 177
B.33 Proposition 4.8.13 . . . 178
B.34 Proposition 4.8.15 . . . 178
B.35 Proposition 4.8.20 . . . 179
B.36 Proposition 4.9.6 . . . 179
B.37 Proposition 4.9.7 . . . 180
B.38 Proposition 4.9.11 . . . 180

C Proofs from Chapter 5 **183**
C.1 Proposition 5.1.4 . . . 183
C.2 Proposition 5.1.8 . . . 183
C.3 Proposition 5.1.14 . . . 183
C.4 Proposition 5.1.15 . . . 184
C.5 Lemma 5.1.17 . . . 184
C.6 Theorem 5.1.18 . . . 185
C.7 Theorem 5.1.19 . . . 190
C.8 Lemma 5.2.8 . . . 192
C.9 Theorem 5.2.12 . . . 192
C.10 Lemma 5.3.4 . . . 192
C.11 Lemma 5.3.6 . . . 193
C.12 Lemma 5.3.9 . . . 195
C.13 Lemma 5.3.10 . . . 195
C.14 Lemma 5.3.11 . . . 196
C.15 Lemma 5.3.12 . . . 196
C.16 Lemma 5.3.13 . . . 197
C.17 Lemma 5.3.14 . . . 197
C.18 Proposition 5.3.15 . . . 198
C.19 Theorem 5.3.16 . . . 201
C.20 Theorem 5.3.17 . . . 204
C.21 Theorem 5.3.25 . . . 205

D Proofs from Chapter 6 **207**
D.1 Proposition 6.1.5 . . . 207
D.2 Proposition 6.1.6 . . . 208
D.3 Proposition 6.1.7 . . . 209
D.4 Proposition D.4 . . . 209
D.5 Proposition 6.2.6 . . . 210
D.6 Proposition 6.2.10 . . . 211
D.7 Lemma 6.2.12 . . . 211
D.8 Lemma 6.3.4 . . . 211
D.9 Theorem 6.3.8 . . . 212
D.10 Proposition 6.4.9 . . . 213
D.11 Theorem 6.4.11 . . . 214
D.12 Theorem 6.4.12 . . . 215
D.13 Lemma 6.6.3 . . . 217
D.14 Proposition 6.6.6 . . . 218
D.15 Lemma 6.6.7 . . . 218
D.16 Theorem 6.6.9 . . . 219
D.17 Theorem 6.6.13 . . . 224

Chapter 1

Introduction

This thesis uses categories of event structures to define true concurrency semantics of a number of reversible process calculi. We start in this chapter by motivating and giving background on the fields of reversibility (Section 1.1), process calculi (Section 1.2), event structures (Section 1.3), and category theory (Section 1.4). We then describe our contributions in more detail in Section 1.5, as well as in which publications these contributions appear in Section 1.6. An overview of notation is available in Chapter 8.

1.1 Reversibility

In a reversible setting, programs are able to undo a set of past actions. The first to consider the usefulness of such programs was Landauer in [55], where he speculated that reversible actions might consume less energy than irreversible because only irreversible computations generate heat. More recently, a common motivation for using reversible frameworks is to model systems which are naturally reversible, such as biochemical systems where reactions can go both ways depending on various factors, as seen in e.g. [37, 83, 84]. Another obvious use for reversible programs is quantum computing, where actions are naturally reversible [2].

Reversibility can also be useful in more traditional programming and debugging. There exist reversible programming languages, including the functional language RFUN [87] and the imperative language Janus [97]. However, there also exist reverse debuggers, which add reversibility to non-reversible languages for the purpose of debugging. Reverse debugging, also known as time travel debugging, allows a usually non-reversible program to go through the past actions of the program in order to find the root of a problem without needing to rerun the entire program again for each action. Reverse debuggers dealing with sequential languages already exist, such as Windows Time Travel debugger [72]. Reverse debugging is even more useful in concurrent programs, which may additionally exhibit transient errors which only occur when threads interleave in a particular way, and are therefore difficult to replicate in a debugger. However, concurrent reverse debugging also comes with additional challenges, since concuurency makes it significantly more difficult to determine what order actions happened in and how actions in one process affected other processes. If a sequential program did a then b then c, it is easy to see that we must reverse c before we can reverse a or b. If a concurrent program did a in parallel with b in parallel with c, then we do not know which order the actions were done in. If the actions are independent it does not matter, and we can undo them in any order we choose, but if for example a wrote something to the memory which c read, then we need to reverse c before we reverse a. Existing reversible programming languages therefore do not include concurrency. UndoDB [86] is a commercial concurrent debugger, which solves this problem by serialising the computation, so we know which order the actions happened in but not whether they depend on one another. This approach fails to capture the real concurrency and causality of actions and creates time overhead. A different approach based on the way concurrency is handled by reversible process calculi was proposed by [56].

1.2 Process Calculi

Process calculi such as the Calculus of Communicating Systems (CCS) [73] and π-calculus [74, 75] model programs which run in parallel and interact with one another. This is particularly relevant to reversible computing because concurrent computations do not have an easily defined "most recent" action, which can be reversed first, but instead a last action of each subprocess, which may or may not have affected another process. Reversible process calculi can be used to model these computations and interactions, and use *causal* reversibility to determine the order in which actions can be reversed. In a causally reversible process, an action can be reversed if it has not caused any other actions. The π-calculus deals with two types of causation: structural causation is created directly by the structure of the process, such as by a prefix operator, while link causation is created by one action changing which names are available to another through scope extrusion. The same notions of causality in concurrent programs can also be used in causal-consistent debugging [62], which allows a user to replay only the parts of the program which caused a specific action.

One important feature of causally reversible calculi is their ability to distinguish true concurrency in a way forward-only processes cannot [79]. For instance, using CCS notation, doing a and b in parallel $(a|b)$ and choosing to do either a and then b or b and then a $(a.b + b.a)$ are equivalent under interleaving semantics; however in a reversible setting we can distinguish them by noting that $a|b$ allows us to perform a followed by b and then to reverse a, which is impossible for $a.b + b.a$. This distinction can also be seen in true concurrency semantics, which leads to our interest in true concurrency semantics of reversible process calculi. In particular we are interested in a model of true concurrency which makes the causal relationships between actions explicit, since the causal relationships is what determines reversal order. True concurrency semantics will also help us define equivalences between processes, such as the ones described later in this section, including processes from different calculi whose semantics use the same true concurrency model.

Two different approaches have been used to remember past actions in reversible process calculi. Dynamically reversible calculi store past actions as memories separate from future actions, either centrally or distributed among the processes. Reversible CCS (RCCS) [35] was the first reversible process calculus to use dynamic reversibility. It did this by equipping every parallel subprocess with a stack of memories, $m \rhd P$, which stores all previous actions. Statically reversible calculi do not remove past actions from their place in the process, but instead use annotations of actions and other operators to denote which changes has occurred in the process. This approach was first used in [80], where CCS with communicating keys (CCSK) where past actions rather than being moved to a memory are given a key, so $a[k] \mid \overline{a}[k]$ means a communication between the two actions has taken place and been given the communication key k. Some work has already been done comparing the two approaches in CCS [57, 68], but we believe having a shared framework for modelling and reasoning about different reversible calculi will be useful for future comparisons between calculi.

The π-calculus is more complicated to reverse than CCS for two main reasons. Actions in the π-calculus cause changes to not just the actions directly involved, but also substitutions in the rest of the process. And unlike CCS, the π-calculus has link causation in addition to structural causation, which complicates causal reversibility. This link causation is further complicated by the fact that it is *non-stable*, meaning the causes of actions may be ambiguous, such as in the process $(\nu\ a)(\overline{b}\langle a\rangle \mid \overline{c}\langle a\rangle \mid a(x))$ where once all the actions have been performed, either of the two output actions $\overline{b}\langle a\rangle$ and $\overline{c}\langle a\rangle$ may have opened the restriction $(\nu\ a)$, which was preventing the third action $a(x)$ from receiving on the restricted channel, and therefore the input can be said to have been caused by either output. In a reversible π-calculus, one would need to decide which of these outputs can or cannot reverse before the input. This presents an issue for both reversible variants and true concurrency semantics of the π-calculus. There exist alternative definitions of causation in the π-calculus. In [18] an extrusion causes any subsequent action in which the extruded name is used, so in the above process whichever of $\overline{b}\langle a\rangle$ and $\overline{c}\langle a\rangle$ happened first would cause both the input and the other output action. In [42] on the other hand the output that happened first would only cause the input, since extrusions only cause actions which have the extruded name as subject.

Like CCS, there exist both dynamic and static reversible variants of the π-calculus. The first reversible π-calculus, reversible higher-order π-calculus ($\rho\pi$) [59], considered memories shared by the entire process and stored them in parallel with subprocesses, an approach copied by the reversible session-based π-calculus (ReS-π) [88]. The later reversible π-calculus (Rπ) [33] took an approach more similar to RCCS by giving each subprocess its own stack of memories. A parametric framework for statically reversible π-calculi was created by [67], using communication keys like CCSK. This required using the keys to keep track of not just

communication but also substitution and opening of restrictions.

There are three different versions of the operational semantics of forward-only π-calculus. Reduction semantics only allow communications and have unlabelled transitions. These are the most common in the forward-only setting and the ones used by $\rho\pi$. Late semantics have labelled transitions and perform substitutions on input actions only when they synchronise with an output action, meaning free names cannot be received from the environment. Rπ and the parametric framework for statically reversible π-calculi use late semantics. Early semantics are also labelled but allow input actions to perform substitutions without synchronising with an output and therefore do allow input actions to receive existing free names from the environment. This creates additional causation in the early semantics between output actions making bound names free and inputs receiving these formerly bound names e.g. in the process $a(x)|(\nu\ c)\overline{b}\langle c\rangle$, the event $a(c)$ will be caused by $\overline{b}\langle c\rangle$ making c a free name. Early semantics are more common than late in the forward-only setting, but so far no reversible early π-calculus has been defined. We are therefore interested in defining a reversible early π-calculus.

Stable non-interleaving operational semantics for the π-calculus were defined by [52], who used extrusion histories to keep track of link causation. They described how early semantics create link causation which does not appear in reduction or late semantics. For example, in the process $(\nu\ a)(\nu\ b)(\overline{c}\langle a\rangle \mid \overline{c}\langle b\rangle \mid a(x))$, after both outputs have been performed we can have an input action $a(b)$ which depends on them both because it requires both restrictions to have been opened. These semantics also resolved the non-stable causation present in π-calculus processes such as $(\nu\ a)(\overline{b}\langle a\rangle \mid \overline{c}\langle a\rangle \mid a(x))$ by labelling transitions with their causes, so that the input action $a(x)$ caused by $\overline{b}\langle a\rangle$ is a different action from the input action $a(x)$ caused by $\overline{c}\langle a\rangle$, and causation becomes unambiguous. This is also how $\rho\pi$, Rπ, and ReS-π resolve the non-stable causation. The parametric framework for statically reversible π-calculi contains three different semantics, which resolve the non-stable causation in three different ways. One is the same as the other calculi, one has whichever output occurred first as the cause of the input, and one has whichever output occurred first as the cause of both the input and other output. Since we are primarily interested in the link causation created by the semantics being early, we would prefer to avoid overcomplicating our calculus by also tackling the problem of non-stable causation, to which solutions already exist.

The internal π-calculus, or πI-calculus, is a subset of the π-calculus where every link sent by an output is bound (private), yielding greater symmetry between inputs and outputs. This avoids situations such as $(\nu\ a)(\overline{b}\langle a\rangle \mid \overline{c}\langle a\rangle \mid a(x))$, since in the πI-calculus, the a sent by b is bound, and therefore different from the a sent by c, which is bound separately and different from the a receiving x. The early πI-calculus is therefore stable, but still contains causal relationships created by early semantics seen in $a(x)|(\nu\ c)\overline{b}\langle c\rangle$, which in πI-calculus is expressed as $a(x)|\overline{b}(c)$, since the restriction on the output is implied. It has also been shown that the asynchronous π-calculus can be encoded in the asynchronous form of the πI-calculus [17]. This leads us to focus our effort on reversible early semantics on the πI-calculus. Since our main interest is modelling the link causation present in early but not late or reduction semantics, restricting ourselves to the internal π-calculus is a way to avoid the added complexity of a full reversible π-calculus which has already been studied by [33, 59, 67, 88]. We see that restricting ourselves to πI-calculus simplifies our semantics in that we can define a statically reversible πI-calculus without explicitly keeping track of the creation and opening on restrictions, unlike the full statically reversible π-calculus in [67]. This also simplifies comparing the statically and dynamically reversible πI-calculi. No previous work has attempted to show a correspondence between different reversible π-calculi, and we consider our work a stepping stone towards more detailed comparisons between dynamically and statically reversible full π-calculi.

So far, we have discussed uncontrolled reversibility. In uncontrolled reversible process calculi, an action can reverse whenever it has not caused any other actions, letting processes potentially do and undo the same actions without making progress. For this reason, some calculi use controlled reversibility, either forcing or preventing reversing at specific points. A simple way to prevent the process from reversing once it has reached a desired state is with irreversible actions, introduced in RCCS in [36]. An irreversible action can be used to commit a transaction or to indicate that a process is on a safe state. Another way to control reversibility is by forcing a process to reverse to a specific state using rollback, introduced in $\rho\pi$ in [58]. This is used when the process needs to escape an undesirable state or has reached an error. Sometimes, reversing may be controlled directly by a user, such as in a reversible debugger where the user may decide to roll back to the last time a given memory cell was changed, or reverse back to the last time the process was in a safe

state. Since controlled variants of reversible calculi are common, we want to implement a control operator as part of our true concurrency semantics of a reversible process calculus. Irreversible actions are easy to implement, being just a class of actions which the rules for reversing do not apply to, so we will focus on rollback.

There exists some work on using contracts to model and reason about processes with rollback capabilities. Contracts for binary interactions where either participant can roll back at any time were investigated in [11, 12]. In retractable contracts [13], the client and server only return to the previous decision point if the transaction is stuck. However, contracts only model reduction semantics of binary interactions between a client and server, whereas we intend our true concurrency semantics to be compositional, and therefore easier to apply to larger systems with many participants.

1.3 Event Structures

As mentioned, we are interested in true concurrency semantics for reversible calculi, in particular true concurrency semantics which describe causal relationships between actions. This leads to our interest in event structures. Event structures [77] are a common model of true concurrency. An event structure consists of a set of events and relations describing their causal relationships. Since the introduction of (prime) event structures, many other variations have been defined, including asymmetric event structures [8], bundle event structures [64], extended bundle event structures [64], flow event structures [21,23], and general event structures [95]. These all have different forms of causality and conflict, which make them more or less expressive and more or less suitable for modelling different systems. Causal relationships between actions are also used in the concurrent reversible setting to determine whether an action, or event, can reverse in any given state, which makes event structures a natural choice for defining true concurrency semantics of reversible process calculi. Reversible variants of prime [82], asymmetric [82], and general [84] event structures have therefore been defined for the purpose of modelling concurrent programs [82] and biochemical systems [84]. When modelling biochemical systems, [84] used reversible event structures' capability for non-causal reversibility, since chemical bonds do not always dissolve in the same causal order as they are created. We will use this non-causal reversibility to model rollback, as it lets us create and describe the causal relationships of a rollback event.

Another model of concurrency frequently studied in relation to reversibility is Petri nets [14, 26, 44, 78]. There are two competing definitions of what makes a Petri net reversible. In [44], a Petri net is called reversible if its initial state is reachable from every reachable state. Another definition comes from [26], where a Petri net is considered reversible if every transition has an inverse transition. This definition is much closer to the one used in programming languages and process calculi. A more recent notion of reversible Petri nets comes from [78], which distinguishes base tokens, existing at the beginning of a computation, from bond tokens, a new type of tokens created by transitions. Transitions here are reversed by removing bonds between tokens and moving participants back to their previous placement. Like reversible general event structures, this was motivated by a desire to model biochemical systems with non-causal reversibility, and the authors therefore give three different semantics for backtracking, causal reversibility, and non-causal reversibility.

Event structures have been used to define true concurrency semantics for forward-only process calculi including CCS [21,22,94], π-calculus [31,32], CCSP [92], LOTOS [64], and CSP [66]. Other true concurrency semantics have been studied [20, 24, 40, 41], but event structures are a common method. True concurrency is even more important in a reversible setting, since causal relationships are so important and, as mentioned above, reversibility naturally leads to distinctions which would not be seen without true concurrency in a forward-only setting. In addition, having a shared framework which multiple calculi can be mapped to allows us to compare processes from different calculi by comparing their event structures. Having event structure semantics of multiple reversible process calculi also lets us define ways to reason about event structures, such as logics, and apply those to any process which can be represented by an event structure. Two logics exist for reasoning about causally reversible systems: Event Identifier Logic [81] and Logic for True Concurrency [9]. Event identifier logic resembles Hennessy-Milner Logic, but adds a reverse primitive $\langle\langle x \rangle\psi$, which denotes that x can be reversed, reaching a configuration where ψ holds. Logic for true concurrency, on the other hand, uses $(\mathbf{x}, \overline{\mathbf{y}} < a\ z)$ to denote that there exists an a-labelled event z, which is not in conflict with any of

the events of the current state, such that all events in $\mathbf{x}$, cause z and all the events of y are concurrent with z, defining concurrent as neither causes the other and they are not in conflict.

Both of these logics characterise hereditary history-preserving bisimulation [16]. A history-preserving bisimulation is a relation $\mathcal{R} \subseteq C_{\mathcal{E}_1} \times C_{\mathcal{E}_2} \times \mathcal{P}(E_1 \times E_2)$ on the configurations of two event structures, $\mathcal{E}_1$ and $\mathcal{E}_2$ and isomorphisms between triples of a configuration with a causal order and labelling of the events in the configuration, $(X, <_{1_X}, \lambda_{1_X})$ and $(Y, <_{2_Y}, \lambda_{2_Y})$, such that if $\mathcal{R}(X, Y, f)$ and $X \xrightarrow{e} X'$ for some a-labelled event e then there exists an a-labelled event e' such that $Y \xrightarrow{e'} Y'$ and $\mathcal{R}(X', Y', f[e \mapsto e'])$. And vice versa if $Y \xrightarrow{e} Y'$. Note that these logics cannot be directly translated to a non-causally reversible setting. Nor do these logics include ways to reason about controlled reversibility. In [60], a weak barbed bisimulation which does not distinguish between forward and reverse action was used. This is applicable to calculi with controlled reversibility but is very coarse. A much more fine-grained notion of bisimulation was the strong bisimulation distinguishing between forward and reverse transitions called forward-reverse bisimulation by [80]. However, there is an interest in exploring notions more fine-grained than the definition put forth in [60] and less fine-grained than forward-reverse bisimulation [3]. If we can describe the semantics of control operators using non-causally reversible event structures, then we can use a logic for non-causally reversible event structures to reason about the behaviours of controlled reversible systems.

There exist true concurrency semantics of reversible process calculi, such as using rigid families to model the semantics of the reversible π-calculus (Rπ) [34], and using configuration structures to model the semantics of a subset of Reversible CCS processes [4]. However, existing semantics first reverse all past actions to find the original process, then map this process to a rigid family or configuration structure, and then apply each of the reversed memories in order to reach the current state of the process. We prefer to map directly from a process with past actions to an event structure with an initial state. This is significantly easier to do from a statically reversible calculus, since the structure of the process does not change as the process evolves, making the structural causation in a process with past actions easier to determine. As mentioned, methods for mapping between dynamic and static calculi have already been studied, which leads to us being able to define event structure semantics of a dynamically reversible calculus by first mapping to a statically reversible calculus. We would like our true concurrency semantics to include a method of controlling reversibility, and in particular would like to use reversible event structures' capacity for non-causal reversibility to describe how a rollback can be considered as just another action, which has its own causal relationship with other actions. However, since no statically reversible calculus controlled by rollback exists, we will need to add rollback to a statically reversible calculus before we can model it. Once we have described event structure semantics of rollback in a specific statically reversible calculus, we can use this as a basis for adding rollback to anything else we can model with event structures.

1.4 Categories

Category theory is a framework used for describing a variety of mathematical structures, including forward-only event structures [95]. In category theory, structures are described by *objects* and *morphisms* between them. Some attempts have been made at using specific kinds of categories such as dagger [1] and inverse [6] categories to model reversibility by having morphisms which reverse one another. Our interest in categories however is more in constructing categories of reversible event structures and relating them to one another.

In the context of event structures and other models of concurrency, morphisms act as partial synchronisations. Event structure morphisms are partial functions mapping events from one event structure to events from another in such a way that (1) any configuration of events from the first structure is also a configuration in the second, and (2) if an event e_1 from the first structure is enabled in such a configuration and mapped to an event e_2 in the second structure, then applying the morphism to the configuration gives us a configuration in which e_2 is enabled. This means that we can say processes are equivalent if the event structures they generate are isomorphic. Additionally, having categories allows us to construct products and coproducts, which we can use when defining parallel composition and choice between event structures representing processes.

With this in mind we are defining categories of reversible event structures, so that these can be used to describe event structure semantics of reversible CCS, rollback, and an early reversible π-calculus. When

defining event structure semantics of CCS, we will be focussing on CCSK because it is easier to create denotational semantics of a statically reversible process calculus than a dynamically reversible calculus, and we will therefore add rollback to CCSK before defining its event structure semantics. To avoid the non-stable causation present in the π-calculus, we will be restricting our early semantics to the internal π-calculus, which we will define a statically and dynamically reversible variant of along with event structure semantics of both, which we will compare.

1.5 Contributions

Categories of event structures We give categories of previously defined varieties of reversible event structures: Prime (Section 4.2), Asymmetric (Section 4.3), and general (Section 4.7). We also introduce reversible variants of bundle (Section 4.4), extended bundle (Section 4.5) and flow (Section 4.6) event structures and categories of these. We define products and coproducts in these categories and functors between the different categories. We also show that some of these functors form adjunctions. This leads to a more complete picture of how these event structures relate to each other and to us being able to compare event structure semantics which use different event structures as their target. We show a tight correspondence between reversible general event structures and finitely enabled configuration systems in Theorems 4.7.17 and 4.7.16.

Roll-CCSK We introduce Roll-CCSK, a variant of CCSK in which reversal is controlled by a rollback operator, in Section 5.2. In Roll-CCSK each rollback operator is associated with a previous action, and once the rollback is reached the process reverses everything caused by that action. We define both centralised and distributed operational semantics of Roll-CCSK and prove an operational correspondence between them in Theorem 5.2.17. These semantics are defined such that the roll can always be undone by a sequence of forwards actions (Theorem 5.2.9)and any subset of activated rolls can be executed reversing only the effects of the actions they refer to (Theorem 5.2.12).

Reversible πI-calculi We introduce two early reversible variants of πI-calculus, a restricted variant of the π-calculus. One, πIK (Section 6.1), uses static reversibility, and the other, πIH (Section 6.2), uses dynamic reversibility. By restricting ourselves to the πI-calculus we focus on the link causation created by being early and keep the causation stable. We define a mapping between them in Section 6.3 and prove operational correspondence in Theorem 6.3.8.

Event structure semantics We define event structure semantics of CCSK (Section 5.1), Roll-CCSK (Section 5.3), πIK (Section 6.4), and πIH (Section 6.5). We define the event structures of πIH operationally and the other three denotationally on the structure of processes. We show that in the denotational semantics, the event structures generated by a process before and after an action are isomorphic, but with different initial states (Theorems 5.1.18, 5.1.19,5.3.16, 5.3.17, 5.3.25, 5.3.26, 6.4.11 and 6.4.12). To get operational event structure semantics, we prove that our operational semantics generate a labelled asynchronous transition system (Theorem 6.5.3), and use an existing mapping [96] to get a labelled prime event structure, which we map to a causal reversible prime event structure. Our denotational semantics are all defined inductively on the structure of processes. We prove operational correspondence between the event structure and operational semantics of each of these calculi and describe a correspondence between the event structures generated be πIK and πIH in Section 6.6, where we show that the event structure generated by the denotational semantics is isomorphic with a restriction of the event structure generated by the operational semantics in Theorem 6.6.9.

1.6 Publications

The contents of this thesis appear in [48–51] of which I am the first and main author.

Section 4.1 and Chapter 4 are based on [48, 50], which defined categories of configuration structures and reversible prime, asymmetric, and general reversible event structures.

Sections 3.4, 3.5, 4.4, and 4.5 as well as Chapter 5 are based on [49], which defined categories of reversible bundle and extended bundle event structures and used those to define event structure semantics of CCSK and Roll-CCSK.

Chapter 6 is based on [51], which described two reversible πI-calculus semantics and two approaches to defining reversible event structure semantics of πI-calculus.

Chapter 2

Reversible Process Calculi

Reversing in a sequential setting is well-understood [65]: One simply recursively undoes the last action of the program. But in a concurrent setting defining the last action is not as easy. It is possible that multiple actions were performed at the same time, or that one wants to undo an action performed by a process, that has not done anything in a while, but would prefer not to undo all the subsequent independent actions of the other processes. This is where the idea of *causal* reversibility arises in [35]. In causal reversibility an action α can be reversed if and only if all the other actions caused by α have been reversed. A number of different approaches to defining causal reversible process calculi exist, and we will describe some of them here.

2.1 Reversible CCS

RCCS [35] is a reversible variant of CCS, which keeps track of past actions by assigning each process P a memory m, creating the construct $m \rhd P$ and referring to these processes with memories as monitored processes. RCCS processes have the following syntax:

$$\begin{array}{lll} R & ::= m \rhd P \mid R_0 \mid R_1 \mid (a)R \\ P & ::= 0 \mid \sum \alpha_i.P_i \mid P_0|P_1 \mid (a)P \\ \alpha & ::= a \mid \overline{a} \mid \tau \\ m & ::= \langle\rangle \mid \langle 1\rangle \cdot m \mid \langle 2\rangle \cdot m \mid \langle *, \alpha, P\rangle \cdot m \mid \langle m, \alpha, P\rangle \cdot m \end{array}$$

RCCS uses $\langle 1\rangle$ and $\langle 2\rangle$ to denote that the process was split into two parallel sub processes, meaning the rest of the memory belongs to the larger split process. A memory of the form $\langle *, \alpha, P\rangle$ denotes that the process has interacted with an unknown process, performing the action α and rejecting the option of behaving as the process P, while $\langle m, \alpha, P\rangle$ means the same but interacting with a known process, having the memory m. A process can only undo the most recent memory in the stack, but, aside from synchronising with other processes involved in the action being reversed, can do so independently of other parallel processes undoing their most recent memories.

Table 2.1 shows the transition rules of RCCS. RCCS transitions are of the form $R \xrightarrow{\mu:\zeta} R'$ where μ is one memory if a single action is being performed and two if the action is a synchronisation, and ζ is either a forwards action α or a backwards action α_*. The synchronisation uses $R_{m_1@m_0}$ to replace all memories of the form $\langle *, \alpha, Q\rangle \cdot m_0$ with $\langle m_1, \alpha, Q\rangle \cdot m_0$. RCCS uses structural congruence to split monitored processes when necessary. Structural congruence for monitored processes consists of the following rules: $m \rhd (P \mid Q) \equiv m \cdot \langle 1\rangle \rhd P \mid m \cdot \langle 2\rangle \rhd Q$ and $m \rhd (a)P \equiv (a)(m \rhd P)$.

In [37], Danos and Krivine extend RCCS to create CCS-R. The main differences between the two languages are that (1) instead of using the entire memory of a process as a tag, each process is now equipped with a separate memory and tag, where each parallel subprocess has a tag from the beginning, which it keeps throughout the computation, and (2) CCS-R includes recursion. Another extension comes from [69], which adds a broadcasting mechanic.

One major problem with RCCS and CCS-R is that they are not compositional. Since processes are only considered reachable if reachable from a single process and memory $\langle\rangle \rhd P$, if two reachable processes are put in parallel the result is non-reachable, being created from a process $\langle\rangle \rhd P \mid \langle\rangle \rhd P'$ where the two processes

$$m \rhd \alpha.P + Q \xrightarrow{m:\alpha} \langle *, \alpha, Q\rangle \cdot m \rhd P \qquad \langle *, \alpha, Q\rangle \cdot m \rhd P \xrightarrow{m:\alpha_*} m \rhd \alpha.P + Q$$

$$\frac{R \xrightarrow{\mu:\zeta} R'}{R \mid S \xrightarrow{\mu:\zeta} R' \mid S} \qquad \frac{S \xrightarrow{\mu:\zeta} S'}{R \mid S \xrightarrow{\mu:\zeta} R \mid S'} \qquad \frac{R \xrightarrow{\mu:\zeta} R' \quad a \notin \zeta}{(a)R \xrightarrow{\mu:\zeta} (a)R'} \qquad \frac{R \equiv S \xrightarrow{\mu:\zeta} S' \equiv R'}{R \xrightarrow{\mu:\zeta} R'}$$

$$\frac{R \xrightarrow{m_0:\alpha} R' \quad S \xrightarrow{m_1:\overline{\alpha}} S'}{R \mid S \xrightarrow{m_0,m_1:\tau} R'_{m_1@m_0} \mid S'_{m_0@m_1}} \qquad \frac{R \xrightarrow{m_0:\alpha_*} R' \quad S \xrightarrow{m_1:\overline{\alpha}_*} S'}{R_{m_1@m_0} \mid S_{m_0@m_1} \xrightarrow{m_0,m_1:\tau} R' \mid S'}$$

Table 2.1: Semantics of RCCS [35]

have no memories of being created by splitting up parallel processes.

CCSK was defined in [80], and distinguishes itself from RCCS by retaining the structure of the process when actions are performed, and annotating past actions with keys instead of generating memories. This means we get $a.P \mid \overline{a}.Q \xrightarrow{\tau[n]} a[n].P \mid \overline{a}[n].Q$, with the shared key n denoting that a and $\overline{a}$ have previously communicated, and we therefore cannot reverse one without reversing the other.

We call the set of action of CCSK $\mathcal{A}$ and let a, b, c range over $\mathcal{A}$, α, β range over $\mathcal{A} \cup \overline{\mathcal{A}}$, and μ range over $\mathcal{A} \cup \overline{\mathcal{A}} \cup \{\tau\}$. We let $\mathcal{K}$ be an infinite set of communication keys and let m, n range over $\mathcal{K}$.

CCSK then has the following syntax, very similar to CCS:

$$P ::= \alpha.P \mid \alpha[n].P \mid P_0 + P_1 \mid P_0 | P_1 \mid P \setminus A \mid P[f]$$

Here $P \setminus A$ restricts communication on actions in $A \cup \overline{A}$ and $P[f]$ applies a function $f : \mathcal{A} \to \mathcal{A}$ to actions done by P.

Table 2.2 shows the operational semantics of CCSK. CCSK uses $\rightsquigarrow$ to denote a reverse action, $\mathsf{std}(P)$ to denote that P is a standard process, meaning it contains no past actions, $\mathsf{key}(P)$ to denote the set of keys used in P, and $\mathsf{fsh}[n](P)$ to denote that the key n is fresh for P. We use $\rightarrowtail$ to denote that an action may be forwards or reverse. The rules are slightly reformulated compared to [80] in that we use structural congruence $\equiv$. The rules for structural congruence are:

$P \mid 0 \equiv P$	$P_0 \mid P_1 \equiv P_1 \mid P_0$	$P_0 \mid (P_1 \mid P_2) \equiv (P_0 \mid P_1) \mid P_2$
$P + 0 \equiv P$	$P_0 + P_1 \equiv P_1 + P_0$	$P_0 + (P_1 + P_2) \equiv (P_0 + P_1) + P_2$

We extend CCSK with recursion as follows. We add process constants $A\langle \tilde{b} \rangle$, together with definitions $A(\tilde{a}) = P_A$, where P_A is a standard process and $\tilde{a}$ is a tuple containing the actions of P_A. This leads us to expand our definition of structural congruence with $A\langle \tilde{b} \rangle \equiv P_A\{^{\tilde{b}}/_{\tilde{a}}\}$.

Definition 2.1.1. *A process P is* reachable *if there exists a standard process Q such that $Q \rightarrowtail^* P$, and* forwards-reachable *if there exists a standard process Q such that $Q \to^* P$.*

Since CCSK is causal all reachable processes are forwards-reachable ([80], Proposition 5.15; the proof still applies with recursion added).

CCSK employs what is known as *static* reversibility, since it keeps the structure of the process intact throughout the computation, to contrast with *dynamic* reversibility, such as in RCCS, which moves past actions out of their place in the process and into separate memories. CCSK and RCCS have been shown to be equivalent by [57, 68], who defined mappings between the two calculi and used those to transfer results about one to the other.

$$\frac{\mathsf{std}(P)}{\alpha.P \xrightarrow{\alpha[n]} \alpha[n].P} \qquad \frac{P \xrightarrow{\mu[m]} P' \quad m \neq n}{\alpha[n].P \xrightarrow{\mu[m]} \alpha[n].P'} \qquad \frac{P \equiv Q \xrightarrow{\mu[n]} Q' \equiv P'}{P \xrightarrow{\mu[n]} P'}$$

$$\frac{P_0 \xrightarrow{\mu[n]} P_0' \quad \mathsf{fsh}[n](P_1)}{P_0 \mid P_1 \xrightarrow{\mu[n]} P_0' \mid P_1} \qquad \frac{P_0 \xrightarrow{\alpha[n]} P_0' \quad P_1 \xrightarrow{\overline{\alpha}[n]} P_1'}{P_0 \mid P_1 \xrightarrow{\tau[n]} P_0' \mid P_1'}$$

$$\frac{P_0 \xrightarrow{\mu[n]} P_0' \quad \mathsf{std}(P_1)}{P_0 + P_1 \xrightarrow{\mu[n]} P_0' + P_1} \qquad \frac{P \xrightarrow{\mu[n]} P' \quad \mu, \overline{\mu} \notin A}{P \setminus A \xrightarrow{\mu[n]} P' \setminus A} \qquad \frac{P \xrightarrow{\mu[n]} P'}{P[f] \xrightarrow{f(\mu)[n]} P'[f]}$$

$$\frac{\mathsf{std}(P)}{\alpha[n].P \overset{\alpha[n]}{\rightsquigarrow} \alpha.P} \qquad \frac{P \overset{\mu[m]}{\rightsquigarrow} P' \quad m \neq n}{\alpha[n].P \overset{\mu[m]}{\rightsquigarrow} \alpha[n].P'} \qquad \frac{P \equiv Q \overset{\mu[n]}{\rightsquigarrow} Q' \equiv P'}{P \overset{\mu[n]}{\rightsquigarrow} P'}$$

$$\frac{P_0 \overset{\mu[n]}{\rightsquigarrow} P_0' \quad \mathsf{fsh}[n](P_1)}{P_0 \mid P_1 \overset{\mu[n]}{\rightsquigarrow} P_0' \mid P_1} \qquad \frac{P_0 \overset{\alpha[n]}{\rightsquigarrow} P_0' \quad P_1 \overset{\overline{\alpha}[n]}{\rightsquigarrow} P_1'}{P_0 \mid P_1 \overset{\tau[n]}{\rightsquigarrow} P_0' \mid P_1'}$$

$$\frac{P_0 \overset{\mu[n]}{\rightsquigarrow} P_0' \quad \mathsf{std}(P_1)}{P_0 + P_1 \overset{\mu[n]}{\rightsquigarrow} P_0' + P_1} \qquad \frac{P \overset{\mu[n]}{\rightsquigarrow} P' \quad \mu, \overline{\mu} \notin A}{P \setminus A \overset{\mu[n]}{\rightsquigarrow} P' \setminus A} \qquad \frac{P \overset{\mu[n]}{\rightsquigarrow} P'}{P[f] \overset{f(\mu)[n]}{\rightsquigarrow} P'[f]}$$

Table 2.2: Semantics of CCSK [83]

2.2 Reversible π-calculus

The π-calculus has two forms of causation. *Structural* causation, as one would find in CCS, comes directly from the structure of the process, e.g. in $a(b).c(d)$ the action $a(b)$ must happen before $c(d)$. *Link* causation, on the other hand, comes from one action making a name available for others to use, e.g. in the πI-calculus process $(\nu c)(c(x)|\overline{b}(c))$, the input action $c(x)$ will be caused by the output $\overline{b}(c)$ making c a free name.

Communication keys were also used in [67] to create a framework for reversible π-calculi. This required using the keys to not only denote past actions, but also which actions had caused substitutions and opening of restrictions. Instead of removing a restriction when opened, the restrictions in statically reversible π-calculi is annotated with the key of the action (or multiple potential actions) which opened the restriction. In addition, when a name is substituted due to an input, every instance of the substituted name is annotated with the associated key. Previous reversible π-calculi had used methods more similar to Danos and Krivine's, including $\rho\pi$ [59], Rπ [33], and ReSπ [88]. Like RCCS and CCS-R, they all use a combination of memories and tags to keep track of the previous actions of processes, but unlike RCCS, $\rho\pi$ and ReSπ both keep their memories separate from the processes that generated them, creating a global set of memories in parallel, with processes with keys $M_1 \mid \cdots \mid M_n \mid \kappa : P_1 \mid \cdots \mid \kappa : P_n$, instead of multiple stacks. This means that instead of a synchronisation between process P and Q adding a new memory to each of their memory stacks, in which they remember each other's tags, a new memory is added to a global set of memories, containing the old processes and the new tag generated for the new processes. The syntax of $\rho\pi$ is as follows:

$$\begin{aligned}
P, Q &::= 0 \mid X \mid \nu a.P \mid (P \mid Q) \mid \langle P \rangle \mid a(X) \triangleright P \\
M, N &::= 0 \mid \nu u.M \mid (M \mid N) \mid \kappa : P \mid [\mu : k] \\
\kappa &::= k \mid \langle h, \tilde{h} \rangle \dot{k} \\
\mu &::= ((\kappa_1 : a\langle P \rangle) \mid (\kappa_2 : a(X) \triangleright Q))
\end{aligned}$$

Where a is a name, h, k are keys, X is a process variable, and u is either a name or a key.

$\rho\pi$ is a higher-order asynchronous π-calculus where the memory $[((\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q)); k]$ records that an output $a\langle P\rangle$ with a key κ_1 sent the process P to a process $a(X) \triangleright Q$ with key κ_2, which after the input has the new key k. This memory can be used to reverse by replacing the thread $k : Q[^P/_X]$ with $(\kappa_1 : a\langle P\rangle) \mid)\kappa_2 : a(X) \triangleright Q)$. The key $\langle h, \tilde{h}\rangle\dot{k}$ denotes that a parallel composition has been split up using the structural congruence rule

$$k : \prod_{i=1}^{n} P_i \equiv \nu\tilde{h}.\prod_{i=1}^{n}(\langle h_i, \tilde{h}\rangle \ldots k : P_i) \quad \tilde{h} = \{h_1, \ldots h_n\}$$

The main difference between $\rho\pi$ and ReSπ, aside from $\rho\pi$ being based on HOπ and ReSπ being based on the standard π-calculus, is the way they handle forking: instead of using a special kind of keys, ReSπ creates a memory $\langle t \rightrightarrows (t_1, t_2)\rangle$ to denote the process tagged t splitting into two processes tagged t_1 and t_2. Rπ on the other hand, uses stacks of memory similarly to RCCS and CCS-R. ReSπ uses session types to structure communication between processes. This is taken further in [71], where monitors are used as memories, meaning the process does not need to store separate memories.

These process calculi all allow the computation to move forwards and backwards at will, but in many reversible systems, this is not what happens in practice. For example, in biochemical reactions, the direction of the process can be determined by temperature or pressure, while in computing there will often be certain actions which cannot be undone, such as sending a message to an external process. Reversible systems where actions cannot reverse whenever they have not caused other actions are called controlled reversible. In order to model controlled reversibility, many of the process calculi mentioned above have been extended with various methods of control. [36] adds irreversible actions to RCCS, so that if the process reaches a desirable state, or the end of a transaction, then it cannot reverse beyond this action. TransCCS [39], instead of making actions reversible, extends CCS with a transaction construct, $[P \lhd_k Q]$, which executes P until it can commit, unless the transaction is aborted, in which case the whole transaction is replaced by Q. Roll-π [58] does the inverse of irreversible actions, only allowing the process to reverse if it encounters the roll primitive. The idea here is that the process only reverses if it encounters an error. In Roll-π, every input action $a(X) \triangleright_\gamma P$ is given a tag γ, which may be associated with a rollback in P, $\mathsf{roll}\ \gamma$. When $a(X) \triangleright_\gamma P$ is involved in a communication,

$$(\kappa_1 : a\langle Q\rangle) \mid (\kappa_2 : a(X) \triangleright_\gamma P) \rightarrow \nu k.(k : P[^{Q,k}/_{X,\gamma}) \mid [(\kappa_1 : a\langle Q\rangle) \mid (\kappa_2 : a(X) \triangleright_\gamma P); k]$$

the roll becomes $\mathsf{roll}\ k$, now including a pointer to the memory which needs to be reversed to complete the roll. After this, more reductions can be executed until $\mathsf{roll}\ k$ is reached. To start the roll, the associated memory is marked for reversal.

$$(\kappa : \mathsf{roll}\ k) \mid [(\kappa_1 : a\langle Q\rangle) \mid (\kappa_2 : a(X) \triangleright_\gamma P); k] \rightarrow (\kappa : \mathsf{roll}\ k) \mid [(\kappa_1 : a\langle Q\rangle) \mid (\kappa_2 : a(X) \triangleright_\gamma P); k]^\bullet$$

This ensures multiple rolls can be in progress at the same time. The roll is the executed by finding all the keys associated with memories caused by $[(\kappa_1 : a\langle Q\rangle) \mid (\kappa_2 : a(X) \triangleright_\gamma P); k]$. The processes and memories associated with these keys are then removed in one move, thereby completing the rollback.

Rollback has also been defined for μKLAIM, a tuple-based language with shared memory [47]. This was done similarly to roll-π, giving the locations of shared memories keys, which change when a process interacts with the location. Another variant of CCS with rollback, CCS_{roll} was defined in [70]. CCS_{roll}, like roll-π, has memories in parallel with the process and equips the process with an ordering of keys, which it uses to determine which actions have been caused by the action being rolled back. This makes the semantics less compositional than CCSK. The causal-consistent reversible debugger for Erlang, CauDEr [56], allows the user to roll back not only to a checkpoint, but to other past events such as receiving or sending a specific message.

In [71], session types enriched with modalities information are used to control reversibility. Another way to use session types to control reversibility is presented in [43], where multiparty session types are given labels which the process may return to at any time after passing them. This was refined by [28] adding points where the rollback is triggered by a participant, and expanded with flexible choice in [29]. Another way to control reversibility is to have a separate process in charge of controlling the reversible process. This is the approach used in [46], where the controlling process is a debugger controlled by the user, and [83]

where a controller process has a different syntax from a reversible process and is not allowed to reverse. Others have attempted to include natural methods of controlling reversal. [7] used energy potential drive the computation. A method of control used specifically to model biochemical systems is the prefix operator in the calculus of covalent bonding [54], in which creating a bond between two concerted actions forces the process to dissolve, or reverse, one of the previous bonds.

Chapter 3

Event Structures and Domains

Event structures are a well-known model of true concurrency. They consist of a set of events and relations on them, describing the causes and conflict between events. Since the original introduction of prime event structures [77], related notions such as general event structures [95], often called simply event structures, which replace conflict with a set of consistent sets of events and causation with sets of enabling events, asymmetric event structures [8], which make the conflict asymmetric, flow event structures [21] which allow an event to have multiple possible conflicting causes, bundle event structures [64] which replace causes with bundles, and extended bundle event structures [64] with bundles and asymmetric conflict, have since been introduced. Winskel [94] defined a category of event structures, and used this to define event structure semantics of CCS.

In this chapter we recall the definitions of domains, prime event structures, asymmetric event structures, bundle event structures, extended bundle event structures, flow event structures, and general event structures. We also introduce categories of bundle event structures and extended bundle event structures, which have not previously had categorical definitions, and construct products and coproducts in these categories. We also define a functor from asymmetric to prime event structures based on the mapping introduced by [80], and show that this forms an adjunction with the functor from prime to asymmetric event structures defined by [8].

Section 3.1 recalls the category of domains, partial orders which can be used to describe the configurations of event structures. The rest of the chapter describes the categories of prime event structures in Section 3.2, asymmetric event structures in Section 3.3, bundle event structures in Section 3.4, extended bundle event structures in Section 3.5, flow event structures in Section 3.6, and general event structures in Section 3.7.

Definition 3.0.1 (Category [5])**.** *A category consists of the following data:*

- *Objects:* $A, B, C, \ldots$
- *Morphisms:* $f, g, h, \ldots$
- *For each morphism there are given objects* $\mathsf{dom}(f)$ *and* $\mathsf{cod}(f)$ *called the domain and codomain of* f*. We write* $f : A \to B$ *to indicate that* $A = \mathsf{dom}(f)$ *and* $B = \mathsf{cod}(f)$*.*
- *Given morphisms* $f : A \to B$ *and* $g : B \to C$*, there is a given morphism* $f \circ g : A \to C$ *called the composite of* f *and* g*.*
- *For any object* A *there is an identity arrow* $1_A : A \to A$*.*

These satisfy the following:

- *Associativity:*

$$f \circ (g \circ h) = (f \circ g) \circ h$$

  for all $f : A \to B$*,* $g : B \to C$*,* $h : C \to D$*.*

- *Unit:* $1_A \circ f = f = f \circ 1_B$ *for all* $f : A \to B$*.*

In the context of event structures and other models of concurrency, morphisms act as partial synchronisations. Event structure morphisms are partial functions mapping events from one event structure to events from another in such a way that (1) any configuration of events from the first structure is also a configuration in the second, and (2) if an event e_1 from the first structure is enabled in such a configuration and mapped to an event e_2 in the second structure, then applying the morphism to the configuration gives us a configuration in which e_2 is enabled.

A functor is a morphism in the category of categories. When two functors $F : B \to A$ and $G : A \to B$ between a category of more powerful event structures A and a category of less powerful event structures B form an adjunction, we can say that G is an approximation. If G is a right adjoint of F then G creates an event structure in B which underestimates the capabilities of the original event structure, and which fewer other event structures can synchronise with. Converesely, if G a left adjoint of F then it overestimates the capabilities of the event structure described by the more powerful framework A when translating to B. Products and coproducts will be useful when it comes to modelling reversible process calculi in future chapters.

3.1 Domains

Forward-only event structures are based on, and map into, a kind of partial order called a domain. Domains are a subcategory of partially ordered sets which we will use to describe the configurations of forward-only event structures. To define domains we first need some notation on partially ordered sets $\langle D, \sqsubseteq \rangle$ from [77,93]:

Definition 3.1.1. *Given $x \in D$, we let $\downarrow x = \{x' \in D \mid x' \sqsubseteq x\}$.*

We write $\uparrow X$ if $X \subseteq D$ is compatible*, meaning there exists a $y \in D$ such that for all $x \in X$, $x \sqsubseteq y$.*

$X \subseteq D$ is pairwise compatible *if for all $x, y \in X$, $\uparrow \{x, y\}$.*

$X \subseteq D$ is directed *if for any $x, y \in X$ there exists $z \in X$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$.*

$\langle D, \sqsubseteq \rangle$ is a complete partial order *(CPO) if for any directed subset $X \subseteq D$, there exists a least upper bound $\bigsqcup X \in D$.*

Element $y \in D$ is compact *if for any directed set $X \subseteq D$, if $y \sqsubseteq \bigsqcup X$ then there exists an $x \in X$ such that $y \sqsubseteq x$.*

$\mathsf{K}(D)$ is the set of compact elements of D.

A CPO $\langle D, \sqsubseteq \rangle$ is finitary *if for all $x \in \mathsf{K}(D)$, $\downarrow x$ is finite.*

A partial order $\langle D, \sqsubseteq \rangle$ is coherent *if for all pairwise compatible $X \subseteq D$, there exists a least upper bound $\bigsqcup X \in D$.*

A complete prime *of D is an element $y \in D$ such that for any compatible $X \subseteq D$, if $y \sqsubseteq \bigsqcup X$ then there exists an $x \in X$ such that $y \sqsubseteq x$.*

The set of complete primes of D is denoted $Pr(D)$.

Given $x \in D$, we let $Pr(x) = \downarrow x \cap Pr(D)$.

A partial order $\langle D, \sqsubseteq \rangle$ is prime algebraic *if for any $x \in D$, $x = \bigsqcup Pr(x)$.*

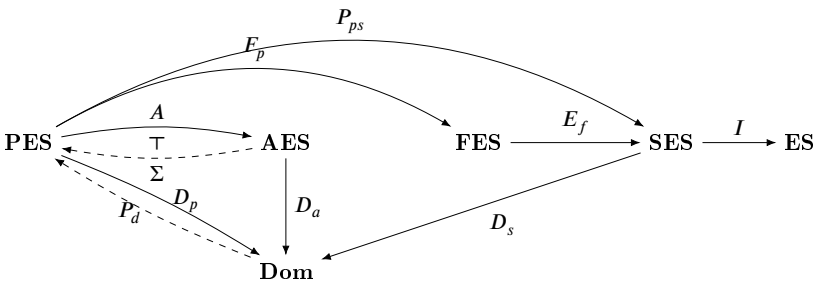


Figure 3.1: Some categories of forward-only event structures and functors between them: PES were introduced in [77], and defined categorically along with **Dom**, **SES**, **ES**, D_p, P_d, P_{ps}, and D_s in [95], **AES**, A, and D_a were introduced in [8], and Σ as a mapping in [82]. The adjunction between A and Σ is new to this work. Dashed lines indicate functors which do not commute with the rest of the diagram

Domains *are coherent, prime algebraic, finitary partial orders.*

Complete primes are considered to be atomic *events*, which cannot be expressed as the upper bound of other elements, while non-prime elements x of domains are the states reached when the events of which x is the least upper bound have happened.

To define the category **Dom**, we need a notion of domain morphism, which we present in Definition 3.1.2. Having a category of domains is necessary because ensuring our mappings from event structures to domains are functors ensures they preserve mophisms.

Definition 3.1.2 (Domain Morphism [95]). *Let $\langle D_0, \sqsubseteq_0 \rangle$ and $\langle D_1, \sqsubseteq_1 \rangle$ be domains. A domain morphism is a function $f : D_0 \to D_1$ such that:*

1. *for all $x_0, y_0 \in D_0$, if $x \preceq_0 y$ then $f(x) \preceq_1 f(y)$ where $\prec$ is defined as:*

$$x \prec x' \text{ if } x \sqsubseteq x' \text{ and } \forall x'' \in D.\ (x \sqsubseteq x'' \sqsubseteq x' \text{ implies } x = x'' \text{ or } x' = x'')\ ;$$

2. *for all $X \subseteq D_0$, if X is pairwise compatible then $f(\bigsqcup X) = \bigsqcup f(X)$;*

3. *for all $X \subseteq D_0$, if $X \neq \emptyset$ and is compatible then $f(\bigsqcap X) = \bigsqcap f(X)$.*

We also define prime intervals in Definition 3.1.3 for use in future mappings from event structures to domains.

Definition 3.1.3 (Prime Interval [95]). *Let $\langle D, \sqsubseteq \rangle$ be a domain. A prime interval is a pair $[x, x']$ of elements of D such that $x \prec x'$. We say $[x, x'] \leq [y, y']$ if $x = x' \sqcap y$ and $x \sqcup y = y'$, and $\sim$ is the transitive and symmetric closure of $\leq$.*

We furthermore define the mapping $[x, x']_\sim \mapsto p$, where p is the unique element in $Pr(x') \setminus Pr(x)$, which is an isomorphism between the $\sim$-classes of prime intervals of $\langle D, \subseteq \rangle$, and the complete primes of D, with the inverse function $p \mapsto [\bigsqcap \{x \in D \mid x \sqsubset p\}, p]_\sim$.

3.2 Prime Event Structures

As we will late define reversible variants of a number of event structures, we first need to introduce the forward-only variants, starting with *prime event structures.*

Definition 3.2.1 (Prime Event Structure [77]). *A prime event structure (PES) is a triple $\mathcal{E} = (E, <, \sharp)$, where E is the set of events, and* causality*, $<$, and* conflict*, $\sharp$, are binary relations on E such that*

1. *$\sharp$ is irreflexive and symmetric;*

2. *$<$ is an irreflexive partial order such that for every $e \in E$, $\{e' \mid e' < e\}$ is finite;*

3. *$\sharp$ is hereditary with respect to $<$, i.e. for all $e, e', e'' \in E$, if $e \sharp e'$ and $e < e''$ then $e'' \sharp e'$.*

A PES consists of a set of events, and causality and conflict relations describing when these events can occur. If $e < e'$ then e' cannot happen unless e has already happened. Moreover if $e \sharp e'$ then e and e' cannot both occur in the same computation. No event is allowed to have an infinite number of causes, as every event should be reachable in a finite number of steps. Furthermore, $\sharp$ is hereditary with respect to $<$, since if e and e' never appear in the same configuration, and e'' cannot appear in a configuration that does not contain e, then e'' and e' will never appear in the same configuration. This condition, along with $\sharp$ being irreflexive, also means that two events cannot be in conflict if they cause the same event.

Definition 3.2.2 (Configuration [77]). *For any PES $\mathcal{E} = (E, <, \sharp)$, we say that $X \subseteq E$ is a* configuration *of $\mathcal{E}$ if X is left-closed under $<$ and conflict-free, meaning no $e, e' \in X$ exist, such that $e \sharp e'$. We use $\mathsf{Conf}(\mathcal{E})$ to denote the set of all configurations of $\mathcal{E}$.*

Example 3.2.3 shows a PES modelled as a domain.



(a) PES $\mathcal{E}$ (b) Domain of PES $\mathcal{E}$

Figure 3.2: Example of a PES and corresponding domain.

Example 3.2.3. *The PES $\mathcal{E}$, shown in Figure 3.2a, with events a, b, c where $a < b$, $a < c$, and $b \sharp c$, has configurations $\emptyset$, $\{a\}$, $\{a, b\}$, and $\{a, c\}$, and therefore $D_p(\mathcal{E}) = C_{\mathcal{E}}$ is the domain seen in Figure 3.2b.*

We recall the definition of a PES morphism in Definition 3.2.4, giving us the category **PES**. Morphisms on event structures can be seen as a sort of synchronisation between the two structures, where if X is a configuration then $f(X)$ is too, and two events e and e' can only synchronise with the same image event $f(e) = f(e')$ if they are in conflict.

Definition 3.2.4 (PES morphism [95])**.** *Let $\mathcal{E}_0 = (E_0, <_0, \sharp_0)$ and $\mathcal{E}_1 = (E_1, <_1, \sharp_1)$ be PESs. A morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$ is a partial function $f : E_0 \to E_1$ such that:*

1. *for all $e \in E_0$, if $f(e) \neq \bot$ then $\{e_1 \mid e_1 <_1 f(e)\} \subseteq \{f(e') \mid e' <_0 e\}$;*
2. *for all $e, e' \in E_0$, if $f(e) \neq \bot \neq f(e')$ and $f(e) \sharp_1 f(e')$, then $e \sharp_0 e'$;*
3. *for all $e, e' \in E_0$, if $f(e) \neq \bot \neq f(e')$, $f(e) = f(e')$, and $e \neq e'$, then $e \sharp_0 e'$.*

We can now describe functors from **PES** to **Dom** and from **Dom** to **PES**. These functors were used in [95] to establish an equivalence between domains and PESs. We will want to establish a similar equivalence between a category of reversible event structures and the formalism used to describe their configurations and transitions.

Definition 3.2.5 (From **PES** to **Dom** [95])**.** *The functor $D_p : \boldsymbol{PES} \to \boldsymbol{Dom}$ is defined as:*

1. *$D_p(\mathcal{E}) = (\mathsf{Conf}(\mathcal{E}), \subseteq)$;*
2. *$D_p(f) = f'$ where $f'(X) = \{f(e) \mid e \in X\}$.*

Definition 3.2.6 (From **Dom** to **PES** [95])**.** *The functor $P_d : \boldsymbol{Dom} \to \boldsymbol{PES}$ is defined as:*

1. *$P_d(\langle D, \sqsubseteq \rangle) = \langle \boldsymbol{Pr}(D), <, \sharp \rangle$ where $p < p'$ if $p \sqsubset p'$ and $p \sharp p'$ if not $p \uparrow p'$;*
2. *$P_d(f) = f'$ where*

$$f(p_0) = \begin{cases} p_1 & \text{if } p_0 \mapsto [x_0, x_0']_\sim,\ f(x_0) \leq f(x_0') \text{ and } [f(x_0), f(x_0')]_\sim \mapsto p_1 \\ \bot & \text{otherwise.} \end{cases}$$

This gives us a close correspondence between **Dom** and **PES**, which is worth keeping in mind as we use domains to model the configurations of other event structures in the rest of this chapter.

3.3 Asymmetric Event Structures

Asymmetric event structures resemble prime event structures, with the difference being that the conflict relation $e \rhd e'$[1] is asymmetric. This means that, rather than e and e' being unable to coexist in a configuration, e' cannot be added to a configuration that contains e. The inverse of this relation, written $e' \lhd e$, may also be seen as a relation of precedence or weak causality, where if $e' \lhd e$ (i.e. $e \rhd e'$) and both events are in a configuration then e' must have been added first. Definition 3.3.1 describes asymmetric event structures.

[1] [8] uses the notation $e \nearrow e'$ to denote the inverse of this.

Definition 3.3.1 (Asymmetric Event Structure [8]). *An asymmetric event structure (AES) is a triple $\mathcal{E} = (E, <, \lhd)$ where E is a set of events, $\lhd$ is the precedence relation, $<$ is the causality relation, which is an irreflexive partial order, and for all $e, e' \in E$:*

1. *$\{e'' \in E \mid e'' < e\}$ is finite and contains no $\lhd$ cycles;*
2. *if $e < e'$ then $e \lhd e'$;*
3. *if $e \sharp e'$ and $e < e''$ then $e'' \sharp e'$, where $\sharp = \lhd \cap \rhd$.*

The configurations of an AES are defined in Definition 3.3.2. It replaces the conflict-freeness condition in Definition 3.2.2 with requiring $\lhd$ to be well-founded, since the configuration would not otherwise be reachable in a finite number of steps, as $e \lhd e'$ means e must have been performed before e' if they are in the same configuration. This condition also prevents cycles of asymmetric conflict, which would make the configuration unreachable. Conditions 1 and 2 ensure that the partial order of an AES's configurations will be finitary, which is necessary for mapping AESs to domains.

Definition 3.3.2 (AES Configuration [8]). *Given an AES $\mathcal{E} = (E, <, \lhd)$, A configuration of $\mathcal{E}$ is a set $X \subseteq E$ such that*

1. *$\lhd$ is well-founded on X;*
2. *for all $e \in X$, $\{e' \in X \mid e' \lhd e\}$ is finite;*
3. *X is left-closed with respect to $<$.*

The set of all configurations of $\mathcal{E}$ is denoted by $\mathsf{Conf}(\mathcal{E})$.

We define a notion of morphism in Definition 3.3.3, giving us the category **AES**. This works very similarly to PES morphisms, again creating a sort of synchronisation between two AESs.

Definition 3.3.3 (AES morphism [8]). *Let $\mathcal{E}_0 = (E_0, <_0, \lhd_0)$ and $\mathcal{E}_1 = (E_1, <_1, \lhd_1)$ be AESs. A morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$ is a partial function $f : E_0 \to E_1$ such that*

1. *for all $e \in E_0$, if $f(e) \neq \bot$ then $\{e_1 \mid e_1 <_1 f(e)\} \subseteq \{f(e') \mid e' <_0 e\}$;*
2. *for all $e, e' \in E_0$, if $f(e) \neq \bot \neq f(e')$ and $f(e) \rhd_1 f(e')$, then $e \rhd_0 e'$;*
3. *for all $e, e' \in E_0$, if $f(e) \neq \bot \neq f(e')$, $f(e) = f(e')$, and $e \neq e'$, then $e \rhd_0 e'$.*

We define a functor from **AES** to **Dom** in Definition 3.3.5, but to do this we first need an ordering of configurations, as defined in Definition 3.3.4. Like the functor from **PES** to **Dom**, the functor from **AES** to **Dom** maps event structures to their set of configurations; however, instead of ordering configurations by set inclusion, it orders them by the extension ordering, which allows new events to be added to a configuration only if they are not prevented by any of the existing events.

Definition 3.3.4 ($\sqsubseteq$ on configurations [8]). *Given an AES $\mathcal{E} = (E, <, \lhd)$, and configurations $X, X' \in \mathsf{Conf}(\mathcal{E})$, we say X'* extends *X, written $X \sqsubseteq X'$ if:*

1. *$X \subseteq X'$;*
2. *no $e \in X$ and $e' \in X' \setminus X$ exist, such that $e' \lhd e$.*

Definition 3.3.5 (From **AES** to **Dom** [8]). *The functor $D_a : \boldsymbol{AES} \to \boldsymbol{Dom}$ is defined as:*

1. *$D_a(\mathcal{E}) = \langle \mathsf{Conf}(\mathcal{E}), \sqsubseteq \rangle$;*
2. *given an AES morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$, we get $D_a(f) = f^*$ where for all $X \in \mathsf{Conf}(\mathcal{E}_0)$, $f^*(X) = \{f(e) \mid e \in X\}$.*

Example 3.3.6 shows an AES modelled as a domain.



(a) AES $\mathcal{E}$ (b) Domain of AES $\mathcal{E}$

Figure 3.3: Example of an AES and the corresponding domain.

Example 3.3.6. *Let the AES $\mathcal{E} = (E, <, \lhd)$ be given by $E = \{a, b, c\}$ and $a < b$ and $b \lhd c$. See Figure 3.3a, where we use dotted arrows for weak causality $\lhd$. Then $\mathcal{E}$ has configurations $\emptyset$, $\{a\}$, $\{c\}$, $\{a, b\}$, $\{a, c\}$, and $\{a, b, c\}$, and therefore $\boldsymbol{D}_a(\mathcal{E})$ is the domain seen in Figure 3.3b.*

We define functors between **AES** and **PES**. Since PESs are less expressive than AESs, mapping from **PES** to **AES** is simple. We just need to turn a symmetric conflict into two asymmetric conflicts.

Definition 3.3.7 (From **PES** to **AES** [8])**.** *The functor $\boldsymbol{A} : \boldsymbol{PES} \to \boldsymbol{AES}$ is defined by:*

1. $\boldsymbol{A}((E, <, \sharp)) = (E, <, < \cup \sharp)$;
2. $\boldsymbol{A}(f) = f$.

One way to go from **AES** to **PES** is to use $\boldsymbol{D}_a$ and $\boldsymbol{P}_d$ to go through domains, but when defining reversible variants of PESs and AESs, [82] used a different mapping, which we can prove is also a functor.

Definition 3.3.8 (From **AES** to **PES**)**.** *The functor $\Sigma : \boldsymbol{AES} \to \boldsymbol{PES}$ is defined by:*

1. $\Sigma((E, <, \lhd)) = (E, <, \lhd \cap \rhd)$;
2. $\Sigma(f) = f$.

Proposition 3.3.9. *$\Sigma : \boldsymbol{AES} \to \boldsymbol{PES}$ is a functor.*

Proof. We prove that the mapping fulfils the conditions of a functor. The main challenge is proving that it preserves morphisms. See Appendix A.1 for full details. □

As our first result (Proposition 3.3.11) states, the two functors are not inverse, but form an adjunction. We shall rely on the following characterisation of adjunctions, based on Definition 9.1 of [5].

Definition 3.3.10. *Let $\boldsymbol{C}$ and $\boldsymbol{D}$ be categories, and let $F : \boldsymbol{D} \to \boldsymbol{C}$ and $G : \boldsymbol{C} \to \boldsymbol{D}$ be functors. Then F is a* left adjoint *of G if one of the following holds:*

1. *there exists a natural transformation $\eta : \boldsymbol{I_D} \to F \circ G$ (the* unit*) such that for any $c \in \boldsymbol{C}$, $d \in \boldsymbol{D}$ and morphism $g : d \to G(c)$ there is a unique morphism $\bar{g} : F(d) \to c$ such that $g = \eta_d \circ G(\bar{g})$, where η_d is the component of η at d;*
2. *there exists a natural transformation $\epsilon : G \circ F \to \boldsymbol{I_C}$ (the* counit*) such that for any $c \in \boldsymbol{C}$, $d \in \boldsymbol{D}$ and morphism $f : F(d) \to c$ there is a unique morphism $\bar{f} : d \to G(c)$ such that $f = F(\bar{f}) \circ \epsilon_c$, where ϵ_c is the component of ϵ at c.*

We shall write $F \dashv G$ to denote that F is left adjoint of G and G is right adjoint of F. Intuitively, if F can be seen as an inclusion, such as $\boldsymbol{A}$ where the behaviour of the resulting AES is always the same as the initial PES, then proving an adjunction helps us determine the relationship between the object in the more powerful category, the AES, and the object in the less powerful category we get by applying G. If G is a right adjoint of F then G creates an event structure in $\boldsymbol{B}$ which underestimates the capabilities of the original event structure, and which fewer other event structures can synchronise with. Converesely, if G a left adjoint of F then it overestimates.

Proposition 3.3.11. $\Sigma \dashv A$.

Proof. For any AES $\mathcal{E} = (E, <, \lhd)$, we have $A(\Sigma(\mathcal{E})) = (E, <, \lhd')$ where $\lhd' = < \cup (\lhd \cap \rhd)$.

Let the unit $\eta : \mathcal{E} \to A(\Sigma(\mathcal{E}))$ be defined by $\eta(e) = e$ for all $e \in E$. This is easily seen to be an AES morphism, noting that $\lhd' \subseteq \lhd$ since $\lhd' = < \cup (\lhd \cap \rhd)$ and $< \subseteq \lhd$. Since $\eta(e) = e$, it is a natural transformation. Let $\mathcal{E}$ be an AES and $\mathcal{E}'$ be a PES, and let $g : \mathcal{E} \to A(\mathcal{E}')$ be an AES morphism. Let $\bar{g} : \Sigma(\mathcal{E}) \to \mathcal{E}'$ use the same mapping on events. Then clearly $g(\mathcal{E}) = \eta(\mathcal{E}) \circ A(\bar{g})$ and $\bar{g}$ is the only morphism with this property.

We then show that for any AES morphism $g : \mathcal{E} \to A(\mathcal{E}')$, $\bar{g} : \Sigma(\mathcal{E}) \to \mathcal{E}'$ is a PES morphism, satisfying the conditions of Definition 3.2.4:

1. This is also a condition of an AES-morphism.
2. If $\bar{g}(e) \sharp' \bar{g}(e')$ then $g(e) \lhd' g(e')$ and $g(e) \rhd' g(e')$, implying that $e \lhd e'$ and $e \rhd e'$, and therefore $e \sharp e'$.
3. If $g(e) = g(e')$ and $e \neq e'$ then by definition of an AES morphism, $e \lhd e'$ and $e \rhd e'$, and therefore $e \sharp e'$.

□

Remark 3.3.12. *In [8] it is shown that going from **AES** to **PES** via the category of domains gives the opposite adjunction, that is, $A \dashv D_a \circ P_d$. Thus **A** has both left and right adjoints. This means when mapping to a PES, we can get either an over or under-estimation of our AESs depending on what we prefer. It also means **A** preserves both products and coproducts.*

We have a shown two examples of the kind of framework for categories of event structures we want, and it is therefore time to define a category ourselves.

3.4 Bundle Event Structures

Bundle event structures (BES) (Definition 3.4.1) extend prime event structures by allowing multiple possible causes for the same event. They do this by replacing the causal relation with a Bundle set, so that if $X \mapsto e$ then one of the events in X must have happened before e can happen. To ensure that we know the causes of each event in any given configuration, BESs also require that events in the same bundle must be in conflict. This gives us the configurations described in Definition 3.4.3.

Definition 3.4.1 (Bundle Event Structures [64])**.** *A bundle event structure (BES) is a triple $\mathcal{E} = (E, \mapsto, \sharp)$ where:*

1. *E is the set of events;*
2. *$\mapsto \subseteq 2^E \times E$ is the bundle set, satisfying $X \mapsto e \Rightarrow \forall e_1, e_2 \in X.(e_1 \neq e_2 \Rightarrow e_1 \sharp e_2)$;*
3. *$\sharp \subseteq E \times E$ is the irreflexive and symmetric conflict relation.*

We show an example of a process which we would use a BES rather than a PES or AES to model. Since we want to model reversible processes, a notion of reversible BESs will be useful.

Example 3.4.2 (Process easily representable by a bundle event structure)**.** *The CCS process $a.b \mid \overline{a}$ can be described by a labelled BES with four events labelled $a, \tau, \overline{a}, b$, the bundle $\{a, \tau\} \mapsto b$, and the conflicts $a \sharp \tau$ and $\overline{a} \sharp \tau$. The process cannot be represented by a labelled PES or AES without splitting the b-labelled event into two, one for each of its conflicting causes.*

Definition 3.4.3 (BES configuration [64])**.** *Given a BES $\mathcal{E} = (E, \mapsto, \sharp)$, a configuration of $\mathcal{E}$ is a set $X \subseteq E$ such that:*

1. *X if conflict-free that is, no events $e, e' \in X$ exist such that $e \sharp e'$;*
2. *there exists a sequence $e_1, \ldots, e_n$ $(n \geq 0)$, such that $X = \{e_1, \ldots, e_n\}$ and for all i, $1 \leq i \leq n$, if $Y \mapsto e_{i+1}$ then $\{e_1, \ldots, e_i\} \cap Y \neq \emptyset$.*

We use $\mathsf{Conf}(\mathcal{E})$ to denote the set of configurations of $\mathcal{E}$.

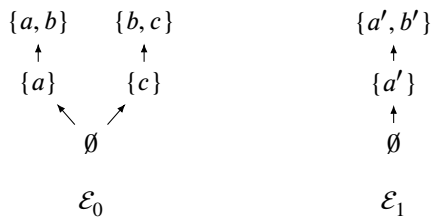


Figure 3.4: The BESs discussed in Example 3.4.5.

A category of BESs has not, to our knowledge, been defined, and so we define a BES morphism in Definition 3.4.4. We want to say that the events of E_0 can behave the same way as those they synchronise with in E_1, but the bundle sets mean this is somewhat harder to describe than in other event structures. If we said that $f(X) \mapsto f(e)^*$ implies $X \mapsto e^*$, we would be requiring $X' \mapsto e^*$ for every $X' = X \cup X''$ where $e \in X'' \Rightarrow f(e) = \bot$, and by extension $e \sharp e'$ if $f(e) = f(e') = \bot$. As this is not how synchronisation should work, we instead adopt Definition 3.4.4.

Definition 3.4.4 (BES-morphism)**.** *Given BESs $\mathcal{E}_0 = (E_0, \mapsto_0, \sharp_0)$ and $\mathcal{E}_1 = (E_1, \mapsto_1, \sharp_1)$, a BES-morphism from $\mathcal{E}_0$ to $\mathcal{E}_1$ is a partial function $f : E_0 \to E_1$ such that for all $e, e' \in E_0$:*

1. *if $f(e) \sharp_1 f(e')$ then $e \sharp_0 e'$;*
2. *if $f(e) = f(e') \neq \bot$ and $e \neq e'$ then $e \sharp_0 e'$;*
3. *for $X_1 \subseteq E_1$ if $X_1 \mapsto_1 f(e)$ then there exists $X_0 \subseteq E_0$ such that $X_0 \mapsto_0 e$, $f(X_0) \subseteq X_1$, and if $e' \in X_0$ then $f(e') \neq \bot$.*

Example 3.4.5 shows two BESs with similar behaviour and a morphism between them. Since a and c are conflicting events with similar behaviour, they can both synchronise with a'.

Example 3.4.5 (BES morphism)**.** *Consider the two BESs $\mathcal{E}_0 = (E_0, \mapsto_0, \sharp_0)$ and $\mathcal{E}_1 = (E_1, \mapsto_1, \sharp_1)$ where $E_0 = \{a, b, c\}$, $a \sharp c$ and $\{a, c\} \mapsto b$, and $E_1 = \{a', b'\}$ and $\{a'\} \mapsto b'$, with the configurations seen in Figure 3.4 where an arrow from X to X' indicates that X contains an event from every bundle associated with the events in $X' \setminus X$ and $X \cup X'$ is conflict-free.*

We can define a BES morphism f from $\mathcal{E}_0$ to $\mathcal{E}_1$ where

$$f(e) = \begin{cases} a' & \text{if } e \in \{a, c\} \\ b' & \text{if } e = b \end{cases}$$

We can also define a morphism f' from $\mathcal{E}_1$ to $\mathcal{E}_0$ where

$$f'(e) = \begin{cases} a & \text{if } e = a' \\ b & \text{if } e = b' \end{cases}$$

We show that BES-morphisms preserve configurations and that BESs and their morphisms actually forms a category.

Proposition 3.4.6. *Given BESs $\mathcal{E}_0 = (E_0, \mapsto_0, \sharp_0)$ and $\mathcal{E}_1 = (E_1, \mapsto_1, \sharp_1)$ and BES-morphism $f : E_0 \to E_1$, if $X \subseteq E_0$ is a configuration of $\mathcal{E}_0$, then $f(X)$ is a configuration of $\mathcal{E}_1$.*

Proof. We show that $f(X)$ fulfils Definition 3.4.3 in Appendix A.2. □

Proposition 3.4.7. ***BES*** *consisting of BESs and BES-morphisms is a category.*

Proof. We show this in Appendix A.3. The main challenge is showing that morphisms are composable. □

We also construct a product in this category in Definition 3.4.9. Products of event structures act as parallel composition. They are therefore very useful when using event structures to model processes. A product of two BESs creates pairs of synchronising events which are caused by the causes of both events and in conflict with other synchronisations involving one of the same events. Products of PESs [91] need to split each event into a copy per possible cause, but using bundles we can avoid that. We deal with bundles by saying that if $X \mapsto e_0$ then (e_0, e_1) is associated with a bundle of every event in which the first component is in X. We say that (e_0, e_1) and (e'_0, e'_1) are in conflict if either e_0 and e'_0 or e_1 and e'_1 are either in conflict or identical.

Definition 3.4.8 (Product)**.** *A product of two objects A and B in a category is an object $A \times B$ with two morphisms $\pi_0 : A \times B \to A$ and $\pi_1 : A \times B \to B$ called projections such that for any object C with morphisms $f_0 : C \to A$ and $f_1 : C \to B$, there exists a unique $f : C \to A \times B$ such that $f \circ \pi_j = f_j$ for $j \in \{0, 1\}$.*

Definition 3.4.9 (BES product)**.** *Given BESs $\mathcal{E}_0 = (E_0, \mapsto_0, \sharp_0)$ and $\mathcal{E}_1 = (E_1, \mapsto_1, \sharp_1)$, we construct $\mathcal{E}_0 \times \mathcal{E}_1 = (E, \mapsto, \sharp)$ with projections π_0, π_1 where:*

1. *$E = E_0 \times_* E_1 = \{(e, *) \mid e \in E_0\} \cup \{(*, e) \mid e \in E_1\} \cup \{(e, e') \mid e \in E_0$ and $e' \in E_1\}$;*
2. *for $(e_0, e_1) \in E$, $\pi_i((e_0, e_1)) = e_i$;*
3. *for any $e \in E$, $X \subseteq E$, $X \mapsto e$ iff there exists $i \in \{0, 1\}$ and $X_i \subseteq E_i$ such that $X_i \mapsto \pi_i(e)$ and $X = \{e' \in E \mid \pi_i(e') \in X_i\}$;*
4. *for any $e, e' \in E$, $e \sharp e'$ iff there exists $i \in \{0, 1\}$ such that $\pi_i(e) \sharp_i \pi_i(e')$, or $\pi_i(e) = \pi_i(e') \neq \bot$ and $\pi_{1-i}(e) \neq \pi_{1-i}(e')$.*

Proposition 3.4.10. *Given BESs $\mathcal{E}_0 = (E_0, \mapsto_0, \sharp_0)$ and $\mathcal{E}_1 = (E_1, \mapsto_1, \sharp_1)$, we get $\mathcal{E}_0 \times \mathcal{E}_1 = (E, \mapsto, \sharp)$ is their product in the category **BES**.*

Proof. We show that we have the required morphisms in Appendix A.4 □

Example 3.4.11 (Product)**.** *Consider the BESs $\mathcal{E}_0$ with events a, b and $\mathcal{E}_1$ with event c such that $\{a\} \mapsto b$. Then $\mathcal{E}_0 \times \mathcal{E}_1$ has the bundles $\{(a, *), (a, c)\} \mapsto (b, *)$ and $\{(a, *), (a, c)\} \mapsto (b, c)$ and conflict $(a, *) \sharp (a, c)$, $(b, *) \sharp (b, c)$, $(*, c) \sharp (a, c)$, $(*, c) \sharp (b, c)$, and $(a, c) \sharp (b, c)$.*

We construct a coproduct of BESs in Definition 3.4.13. This coproduct, similarly to the coproduct of BESs acts as a choice between which BES to behave as, since their events are in conflict with each other. As with the product, coproducts of event structures are useful when modelling processes.

Definition 3.4.12 (Coproduct)**.** *A coproduct of two objects A and B in a category is an object $A + B$ with two morphisms $i_0 : A \to A + B$ and $i_1 : B \to A + B$ called injections, such that for any object C with morphisms $f_0 : A \to C$ and $f_1 : B \to C$, there exists a unique $f : A + B \to C$ such that $i_j \circ f = f_j$ for $j \in \{0, 1\}$.*

Definition 3.4.13 (BES coproduct)**.** *Given BESs $\mathcal{E}_0 = (E_0, \mapsto_0, \sharp_0)$ and $\mathcal{E}_1 = (E_1, \mapsto_1, \sharp_1)$, we construct $\mathcal{E}_0 + \mathcal{E}_1 = (E, \mapsto, \sharp)$ with injections i_0, i_1 where:*

- *$E = \{(0, e) \mid e \in E_0\} \cup \{(1, e) \mid e \in E_1\}$;*
- *for $e \in E_j$, $i_j(e) = (j, e)$ for $j \in \{0, 1\}$;*
- *$X \mapsto (j, e)$ iff for all $(j', e') \in X$, $j = j'$ and $i_j(X) \mapsto_j e$;*
- *$(j, e) \sharp (j', e')$ iff $j \neq j'$ or $e \sharp_j e'$.*

Proposition 3.4.14. *If $\mathcal{E}_0$ and $\mathcal{E}_1$ are BESs, then $\mathcal{E}_0 + \mathcal{E}_1$ is their coproduct.*

Proof. We show that we have the required morphisms in Appendix A.5. □

We have mappings from **PES** to **BES** and from **BES** to **Dom**. B creates a BES with only one event in each bundle, replicating the causality relations of PESs and D_b like D_p orders the configurations of a BES by inclusion.

Definition 3.4.15 (**PES** to **BES**). *The mapping* $B : \boldsymbol{PES} \to \boldsymbol{BES}$ *is defined as:*

1. $B((E, <, \sharp)) = (E, \mapsto, \sharp)$, *where* $\{e'\} \mapsto e$ *if* $e' < e$;
2. $B(f) = f$.

Definition 3.4.16 (From **BES** to **Dom** [64]). *The functor* $D_b : \boldsymbol{BES} \to \boldsymbol{Dom}$ *is defined as:*

1. $D_b(\mathcal{E}) = (\mathsf{Conf}(\mathcal{E}), \subseteq)$;
2. $D_b(f) = f'$ *where* $f'(X) = \{f(e) \mid e \in X\}$.

We have now defined a useful category of BESs, similarly to what has previously been defined for other event structures, and have proved that our notion of morphisms is reasonable in that it preserves configurations and the existing mapping to domains is now a functor.

3.5 Extended Bundle Event Structures

Extended bundle event structures (EBES) (Definition 3.5.1) add asymmetric conflict, similar to that of AESs to bundle event structures.

Definition 3.5.1 (Extended Bundle Event Structure [64]). *An extended bundle event structure (EBES) is a triple* $\mathcal{E} = (E, \mapsto, \triangleright)$ *where:*

1. E *is the set of events;*
2. $\mapsto \subseteq 2^E \times E$ *is the bundle set, satisfying* $X \mapsto e \Rightarrow \forall e_1, e_2 \in X.(e_1 \neq e_2 \Rightarrow e_1 \lhd e_2)$;
3. $\lhd \subseteq E \times E$ *is the asymmetric conflict relation, which is irreflexive.*

EBES configurations are sets which have an order in which the events could have happened.

Definition 3.5.2 (EBES configuration [64]). *Given an EBES* $\mathcal{E} = (E, \mapsto, \triangleright)$, *a configuration of* $\mathcal{E}$ *is a set* $X \subseteq E$ *such that there exists a sequence* $e_0, \ldots, e_n$ *such that:*

1. $\{e_0, \ldots, e_n\} = X$;
2. *if* $e_i \lhd e_j$ *then* $i < j$;
3. *if* $X \mapsto e_i$ *then* $X \cap \{e_0, \ldots, e_{i-1}\} \neq \emptyset$.

A category of EBESs has not, to our knowledge, been defined, so we define an EBES-morphism in Definition 3.5.3. This morphism is similar to the BES morphism defined previously, and asymmetric conflict is treated much the same way as symmetric.

Definition 3.5.3 (EBES-morphism). *Given EBESs* $\mathcal{E}_0 = (E_0, \mapsto_0, \triangleright_0)$ *and* $\mathcal{E}_1 = (E_1, \mapsto_1, \triangleright_1)$, *an EBES-morphism from* $\mathcal{E}_0$ *to* $\mathcal{E}_1$ *is a partial function* $f : E_0 \to E_1$ *such that and for all* $e, e' \in E_0$:

1. *if* $f(e) \lhd_1 f(e')$ *then* $e \lhd_0 e'$;
2. *if* $f(e) = f(e') \neq \bot$ *and* $e \neq e'$ *then* $e \lhd_0 e'$;
3. *for* $X_1 \subseteq E_1$ *if* $X_1 \mapsto_1 f(e)$ *then there exists* $X_0 \subseteq E_0$ *such that* $X_0 \mapsto_0 e$, $f(X_0) \subseteq X_1$, *and if* $e' \in X_0$ *then* $f(e') \neq \bot$;
4. *for any* $X_0 \subseteq E_0$, *if* X_0 *is a configuration of* $\mathcal{E}_0$, *then* $f(X_0)$ *is a configuration of* $\mathcal{E}_1$.

Proposition 3.5.4. *Given EBESs* $\mathcal{E}_0 = (E_0, \mapsto_0, \triangleright_0)$ *and* $\mathcal{E}_1 = (E_1, \mapsto_1, \triangleright_1)$ *and EBES-morphism* $f : E_0 \to E_1$, *if* $X \subseteq E_0$ *is a configuration of* $\mathcal{E}_0$, *then* $f(X)$ *is a configuration of* $\mathcal{E}_1$.

Proposition 3.5.5. ***EBES*** *consisting of EBESs and EBES-morphisms is a category.*

Proof. We show this in Appendix A.6. The main challenge is showing that morphisms are composable. □

We also construct a product of EBESs in Definition 3.5.6. This definition is very similar to the product of BESs And treats asymmetric conflict in the same way as the BES product treats symmetric conflict.

Definition 3.5.6 (EBES product)**.** *Given EBESs* $\mathcal{E}_0 = (E_0, \mapsto_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, \mapsto_1, \rhd_1)$*, we construct* $\mathcal{E}_0 + \mathcal{E}_1 = (E, \mapsto, \rhd)$ *with projections* π_0*,* π_1 *where:*

1. $E = E_0 \times_* E_1 = \{(e, *) \mid e \in E_0\} \cup \{(*, e) \mid e \in E_1\} \cup \{(e, e') \mid e \in E_0 \text{ and } e' \in E_1\}$*;*
2. *for* $(e_0, e_1) \in E$*,* $\pi_i((e_0, e_1) = e_i$*;*
3. *for any* $e \in E$*,* $X \subseteq E$*,* $X \mapsto e$ *iff there exists* $i \in \{0, 1\}$ *and* $X_i \subseteq E_i$ *such that* $X_i \mapsto \pi_i(e)$ *and* $X = \{e' \in E \mid \pi_i(e') \in X_i\}$*;*
4. *for any* $e, e' \in E$*,* $e \lhd e'$ *iff there exists* $i \in \{0, 1\}$ *such that* $\pi_i(e) \lhd_i \pi_i(e')$*, or* $\pi_i(e) = \pi_i(e') \neq \bot$ *and* $\pi_{1-i}(e) \neq \pi_{1-i}(e')$*.*

Proposition 3.5.7. *Given EBESs* $\mathcal{E}_0 = (E_0, \mapsto_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, \mapsto_1, \rhd_1)$*, we have that* $\mathcal{E}_0 \times \mathcal{E}_1 = (E, \mapsto, \rhd)$ *is their product.*

Proof. We show that we have the required morphisms in Appendix A.7. □

We also construct a coproduct of EBESs in Definition 3.5.8.

Definition 3.5.8 (EBES coproduct)**.** *Given EBESs* $\mathcal{E}_0 = (E_0, \mapsto_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, \mapsto_1, \rhd_1)$*, we construct* $\mathcal{E}_0 + \mathcal{E}_1 = (E, \mapsto, \rhd)$ *with injections* i_0*,* i_1 *where:*

1. $E = \{(0, e) \mid e \in E_0\} \cup \{(1, e) \mid e \in E_1\}$*;*
2. *for* $e \in E_j$*,* $i_j(e) = (j, e)$ *for* $j \in \{0, 1\}$*;*
3. $X \mapsto (j, e)$ *iff for all* $(j', e') \in X$*,* $j = j'$ *and* $i_j(X) \mapsto_j e$*;*
4. $(j, e) \lhd (j', e')$ *iff* $j \neq j'$ *or* $e \lhd_j e'$*.*

Proposition 3.5.9. *If* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *are EBESs, then* $\mathcal{E}_0 \times \mathcal{E}_1$ *is their coproduct.*

Proof. Similar to the proof of BES coproduct (Proposition 3.4.14). □

We also describe functors mapping between the previously introduced categories and **EBES**.

Definition 3.5.10 (**AES** to **EBES**)**.** *The mapping* $B_a : \boldsymbol{AES} \to \boldsymbol{EBES}$ *is defined as:*

1. $B_a((E, <, \lhd)) = (E, \mapsto, \rhd)$*, where* $\{e'\} \mapsto e$ *if* $e' \prec e$*;*
2. $B_a(f) = f$*.*

Definition 3.5.11 (**BES** to **EBES**)**.** *The mapping* $B_e : \boldsymbol{BES} \to \boldsymbol{EBES}$ *is defined as:*

1. $B_e((E, \mapsto, \sharp)) = (E, \mapsto, \lhd)$*, where* $e \lhd' e'$ *if* $e \sharp e'$*;*
2. $B_e(f) = f$*.*

As we did for BESs in the previous section, we have now defined a category of EBESs and proved that our notion of morphisms is reasonable in that it preserves configurations and the existing mapping to domains is now a functor.



(a) FES $\mathcal{E}$ (b) Domain of FES $\mathcal{E}$

Figure 3.5: Example of a FES and corresponding Domain.

3.6 Flow Event Structures

A flow event structure (FES) (Definition 3.6.1) were introduced in [21] because they provide an event structure with both a graphical representation and an easy definition for process calculi operators. This has made them useful for defining semantics of process calculi [22, 24]. A FES resembles a PES, in that it contains a set of events and a symmetric conflict relation. However, a FES replaces the causation relation with a flow relation, which, while similar, allows events to happen without every cause having happened. In a flow event structure e can happen if whenever $e' \prec e$, either e' has happened, or there exists an e'' such that $e' \sharp e'' \prec e$, and e'' has happened. This gives us the configurations described in Definition 3.6.2. Flow event structures therefore more closely resemble BESs than PESs in that we can have conflicting causes. However, while bundle event structures explicitly state which causes we can choose from because they are in the same bundle, flow event structures leave their choice implied by the causal relation.

Definition 3.6.1 (Flow Event Structure [21])**.** *A flow event structure (FES) is a triple $\mathcal{E} = (E, \prec, \sharp)$ where:*

1. *E is the set of events;*
2. *$\sharp \subseteq E \times E$ is the symmetric conflict relation;*
3. *$\prec \subseteq E \times E$ is the irreflexive flow relation.*

Definition 3.6.2 (FES configurations [21])**.** *Given a FES $\mathcal{E} = (E, \prec, \sharp)$, a configuration of $\mathcal{E}$ is a subset $X \subseteq E$ such that:*

1. *X is conflict-free;*
2. *$\prec$ is acyclic on X;*
3. *for all $e \in X$, the set $\{e' \mid e' \in X \text{ and } e' \prec_X^* e\}$ is finite, where $\prec_X = \prec \cap X \times X$;*
4. *for all $e \in X$, if $e' \prec e$ and $e' \notin X$, then there exists $e'' \in X$ such that $e'' \prec e$ and $e' \sharp e''$.*

The set of configurations of $\mathcal{E}$ is denoted $\mathsf{Conf}(\mathcal{E})$.

Example 3.6.3 shows a FES modelled as a Domain.

Example 3.6.3. *The FES $\mathcal{E}$, shown in Figure 3.5a, with events a, b, c where $a \prec b$, $c \prec b$, and $a \sharp c$ has configurations $\emptyset$, $\{a\}$, $\{a, b\}$, $\{c\}$ and $\{c, b\}$, and therefore corresponds to the Domain seen in Figure 3.5b, the same as the BES seen in Example 3.4.5.*

A categorical definition of FESs was presented in [30], using the notion of morphisms described in Definition 3.6.4. They used this to define the product of the subcategory of *well-behaved* FESs (Definition 3.6.5), as seen in Definition 3.6.6. Unlike previous event structure morphisms, this one makes preserving configurations an explicit condition, rather than a property arising from the definition.

Definition 3.6.4 (FES-morphism [30])**.** *Given FESs $\mathcal{E}_0 = (E_0, \prec_0, \sharp_0)$ and $\mathcal{E}_1 = (E_1, \prec_1, \sharp_1)$, a FES-morphism from $\mathcal{E}_0$ to $\mathcal{E}_1$ is a partial function $f : E_0 \to E_1$ such that for $e, e' \in E_0$:*

1. *If $f(e) \sharp_1 f(e')$ then $e \sharp_0 e'$;*

2. *If* $f(e) = f(e') \neq \bot$ *and* $e \neq e'$ *then* $e \sharp_0 e'$;

3. *If* $f(e) \prec_1 f(e')$ *then* $e \prec_0 e'$;

4. *If* X *is a finite configuration of* $\mathcal{E}_0$ *then* $f(X)$ *is a configuration of* $\mathcal{E}_1$.

Definition 3.6.5 (Well-behaved FES [30]). *A FES,* $(E, \prec, \sharp)$*, is well behaved if for all events* $a, b, c \in E$*, if* $a \sharp b \prec c$ *and neither* $a \sharp c$*,* $a \prec c$ *or* $c \prec a$*, then there exists an event* $d \in E$*, such that* $b \sharp d \prec c$*, and for all events* $e \in E$ *such that* $e \sharp d$ *and* $e \neq b$*, we get* $e \sharp b$ *and either* $e \sharp c$*,* $e \prec c$*, or* $c \prec e$.

Definition 3.6.6 (Product of FESs [30]). *Given FESs* $\mathcal{E}_0 = (E_0, \prec_0, \sharp_0)$ *and* $\mathcal{E}_1 = (E_1, \prec_1, \sharp_1)$*, we can construct* $\mathcal{E}_0 \times \mathcal{E}_1 = (E, \prec, \sharp)$ *with projections* π_0 *and* π_1 *where:*

1. $E = E_0 \times_* E_1 = \{(e, *) \mid e \in E_0\} \cup \{(*, e) \mid e \in E_1\} \cup \{(e, e') \mid e \in E_0$ *and* $e' \in E_1\}$;

2. *for* $(e_0, e_1) \in E$*,* $\pi_i((e_0, e_1) = e_i$;

3. *for any* $e, e' \in E$*,* $e \prec e'$ *iff there exists* $i \in \{0, 1\}$ *such that* $\pi_i(e) \prec_i \pi_i(e')$;

4. *for any* $e, e' \in E$*,* $e \sharp e'$ *iff there exists* $i \in \{0, 1\}$ *such that* $\pi_i(e) \sharp_i \pi_i(e')$*, or* $\pi_i(e) = \pi_i(e') \neq \bot$ *and* $\pi_{1-i}(e) \neq \pi_{1-i}(e')$.

As with previous forward-only event structures, we have functors from **PES** to **FES** and **FES** to **Dom**.

Definition 3.6.7 (From **FES** to **Dom** [19]). *The functor* $D_f : \boldsymbol{FES} \rightarrow \boldsymbol{Dom}$ *is defined as:*

1. $D_f((E, \prec, \sharp)) = ((\mathsf{Conf}((E, \prec, \sharp)), \subseteq)$;

2. $D_f(f) = f'$ *where* $f'(X) = \{f(e) \mid e \in X\}$.

Definition 3.6.8 (**PES** to **FES**). *The functor* $F_p : \boldsymbol{PES} \rightarrow \boldsymbol{FES}$ *is defined as:*

1. $F_p((E, <, \sharp)) = (E, <, \sharp)$;

2. $F_p(f) = f$.

Proposition 3.6.9. F_p *is a functor from* ***PES*** *to* ***FES***.

Proof. From the definitions of PESs and FESs it is clear that, if $\mathcal{E}$ is a PES then $F_p(\mathcal{E})$ is a FES, and if $f : \mathcal{E}_0 \rightarrow \mathcal{E}_1$ is a PES morphism then $f : F_p(\mathcal{E}_0) \rightarrow F_p(\mathcal{E}_1)$ is a FES morphism.

Having proved that F_p is a mapping from **PES** to **FES**, the rest of the proof is similar to previous functors. □

We define a mapping from BESs to FESs, which is not a functor due to not preserving morphisms for the reasons shown in Example 3.6.12. Definition 3.6.10 has 3 conditions for transforming bundles into a flow relation. The need for conditions 1 and 3 is obvious. As for condition 2, consider a BES with bundles $\{a\} \mapsto b$ and $\{a, c\} \mapsto b$ and $a \sharp c$. Without condition 2, the corresponding FES would have $a \prec b$, $c \prec b$, and $a \sharp c$, meaning c on its own could cause b.

Definition 3.6.10 (BES to FES). *We define* $F((E, \mapsto, \sharp)) = (E \cup E', \prec, \sharp \cup \sharp'')$*, where:*

1. *if* $e' \in X$ *and* $X \mapsto e$ *then* $e' \prec e$;

2. *to each bundle* $X \mapsto e$ *and* $X \neq \emptyset$*, corresponds a unique* $e' \in E'$ *such that* $e' \prec e$*,* $e' \sharp' e'$*, and for all* $e'' \in X$*,* $e'' \sharp' e'$;

3. $e \sharp' e'$ *if* $\emptyset \mapsto e$ *and* $e = e'$.

Example 3.6.11 shows why we cannot map from a FES to a BES.

Example 3.6.11 (FES not representable by BES [19]). *The FES* $\mathcal{E} = (E, \prec, \sharp)$ *where* $E = \{a, b, c, d, e, f\}$*,* $a \prec f$*,* $b \prec f$*,* $c \prec f$*,* $d \prec f$*,* $e \prec f$*, and* $a \sharp b \sharp c \sharp d \sharp e$ *cannot be represented by a BES. The only possible bundles for* f*, without changing the conflicts, would be* $\{a, b\} \mapsto f$*,* $\{b, c\} \mapsto f$*,* $\{c, d\} \mapsto f$*, and* $\{d, e\} \mapsto f$*. No combination of these bundles would allow* a, d *and* b, e *to enable* f *without also allowing* a, e *to enable* f.

Example 3.6.12 (BES to FES does not preserve morphisms)**.** *Consider the two BESs* $(E_0, \mapsto_0, \sharp_0)$ *and* $(E_1, \mapsto_1, \sharp_1)$ *with* $E_0 = \{a, b, c\}$, $\{a\} \mapsto_0 c$, $a \sharp_0 b$, $E_1 = \{a', b', c'\}$, $\{a', b'\} \mapsto_1 c'$, *and* $a' \sharp_1 b'$. *Here* $f(e) = e'$ *is a BES morphism.*

However, if we map these to FESs, both get new events d *and* d', *with flow* $a \prec_0 c$ *and* $d \prec_0 c$ *and new conflict* $d \sharp'_0 d$ *and* $a \sharp'_0 d$, *and* $a' \prec_1 c'$, $b' \prec_1 c'$, $d' \prec_1 c'$, $b' \sharp'_1 d'$, $a' \sharp'_1 d'$, *and* $d' \sharp'_1 d'$.

Since $b' \prec_1 c'$, *we cannot have* $f(b) = b'$ *and* $f(c) = c'$ *without* $b \prec c$.

As Example 3.6.12 shows, the need for adding an additional event for each bundle when converting from a BES to a FES causes us problems with morphisms. However, we still have one more category of more powerful event structures to go and we shall see that these can model both BESs and FESs no problem.

3.7 General Event Structures

General event structures, or simply *event structures* have an enabling relation, $\vdash$, so that if a set of events X enables e, $X \vdash e$, then e can happen in the configuration X and any configurations X is a subset of. General event structures also replace conflict with a set of consistent sets of events, so that configurations cannot have any inconsistent finite subsets.

Definition 3.7.1 (Event structure [95])**.** *An* event structure *(ES) is a triple* $\mathcal{E} = (E, \mathsf{Con}, \vdash)$, *where* E *is a set of events,* $\mathsf{Con} \subseteq_{\text{fin}} 2^E$ *is the consistency relation,* $\vdash \subseteq \mathsf{Con} \times E$ *is the enabling relation, and*

1. *if* $X \in \mathsf{Con}$ *and* $Y \subseteq X$ *then* $Y \in \mathsf{Con}$;
2. *if* $X \vdash e$ *and* $X \subseteq Y \in \mathsf{Con}$ *then* $Y \vdash e$.

ES configurations are defined in Definition 3.7.2. Since conflict has been replaced with consistent sets, we now require all finite subsets of a configuration to be consistent. In order to ensure that the configuration is reachable, we need there to exist a sequence in which the events could have happened.

Definition 3.7.2 (Configuration [95])**.** *Given an ES* $\mathcal{E} = (E, \mathsf{Con}, \vdash)$, *a* configuration *of* $\mathcal{E}$ *is a set* $X \subseteq E$ *such that*

1. *for all* $X' \subseteq_{\text{fin}} X$, $X' \in \mathsf{Con}$;
2. *for all* $e \in X$, *there exists a sequence* $e_0, e_1, \ldots, e_n$ *such that* $e_n = e$ *and for all* $0 \leq i \leq n$, $\{e_0, \ldots, e_{i-1}\} \vdash e_i$.

Again, the set of all configurations of $\mathcal{E}$ *is denoted* $\mathsf{Conf}(\mathcal{E})$.

We define an ES morphism in Definition 3.7.3, giving us the category **ES**.

Definition 3.7.3 (ES morphism [95])**.** *Let* $\mathcal{E}_0 = (E_0, \mathsf{Con}_0, \vdash_0)$ *and* $\mathcal{E}_1 = (E_1, \mathsf{Con}_1, \vdash_1)$ *be event structures. An ES morphism is a partial function* $f : E_0 \to E_1$ *such that*

1. *for all* $e_0 \in E_0$, *if* $f(e_0) \neq \bot$ *and* $X \vdash_0 e_0$, *then* $f(X) \vdash_1 f(e_0)$;
2. *if* $X \in \mathsf{Con}_0$ *then* $f(X) \in \mathsf{Con}_1$;
3. *for all* $e_0, e'_0 \in E_0$, *if* $\{e_0, e'_0\} \in \mathsf{Con}_0$ *and* $f(e_0) = f(e'_0) \neq \bot$ *then* $e_0 = e'_0$.

Stable event structures, defined in Definition 3.7.4, are a subset of event structures, which along with ES morphisms between stable event structures make up the category **SES**. An ES is stable if in any given configuration, each event has a unique enabling set.

Definition 3.7.4 (Stable event structure [95])**.** *A* stable *event structure (SES) is an event structure* $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ *such that if* $X \vdash e$, $X' \vdash e$ *and* $X \cup X' \cup \{e\} \in \mathsf{Con}$ *then* $X \cap X' \vdash e$.

We then define functors from **PES** and **FES** to **SES** and from **SES** to **Dom**. The mapping E_f was defined though not shown to be a functor by [21].

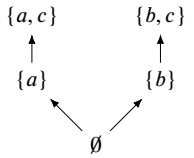


Figure 3.6: Domain corresponding to SES $\mathcal{E}$ of Example 3.7.10.

Definition 3.7.5 (From **PES** to **SES** [95]). *The functor* $\boldsymbol{P_{ps}}$: $\boldsymbol{PES} \to \boldsymbol{SES}$ *is defined as:*

1. $P_{ps}((E, <, \sharp)) = (E, Con, \vdash)$ *where* $\mathsf{Con} = \{X \subseteq_{\text{fin}} E \mid e, e' \in X \Rightarrow \neg(e \sharp e')\}$ *and* $X \vdash e$ *if* $\{e' \mid e' < e\} \subseteq X \in \mathsf{Con}$*;*

2. $P_{ps}(f) = f'$ *where* $f'(X) = \{f(e) \mid e \in X\}$.

Remark 3.7.6. *AESs are not as easily mapped into ESs. Consider the AES* $\mathcal{E} = (E, <, \lhd)$ *where* $E = \{a, b\}$ *and* $a \lhd b$. *An attempt to describe* $\mathcal{E}$ *as a ES* $(E, \mathsf{Con}, \vdash)$ *where* $E = \{a, b\}$ *would require* $\emptyset \vdash a$ *and* $\{b\} \in \mathsf{Con}$, *but not* $\{b\} \vdash a$, *which is not possible. We shall see that matters are different in the reversible setting, thanks to SRES enablings having a preventing set.*

Definition 3.7.7 (From **FES** to **ES** [21]). *The functor* $\boldsymbol{E_f}$: $\boldsymbol{FES} \to \boldsymbol{ES}$ *is defined as:*

1. $E_f((E, \prec, \sharp)) = (E, Con, \vdash)$ *where* $\mathsf{Con} = \{X \subseteq_{\text{fin}} E \mid e, e' \in X \Rightarrow \neg(e \sharp e')\}$ *and* $X \vdash e$ *if* $X \in \mathsf{Con}$ *and for any* $e' \prec e$, *either* $e' \in X$ *or there exists* $e'' \in X$ *such that* $e'' \prec e$ *and* $e'' \sharp e'$*;*

2. $E_f(f) = f$.

Proposition 3.7.8. E_f *is a functor from* $\boldsymbol{FES}$ *to* $\boldsymbol{ES}$.

Proof. We show this in Appendix A.8. The main challenge is showing that E_f preserves morphisms. □

Definition 3.7.9 (From **SES** to **Dom** [95]). *The functor* $\boldsymbol{D_s}$: $\boldsymbol{SES} \to \boldsymbol{Dom}$ *is defined as:*

1. $D_s((E, Con, \vdash)) = \langle \mathsf{Conf}(\mathcal{E}), \subseteq \rangle$*;*

2. $D_s(f) = f$.

Example 3.7.10 shows an SES modelled as a domain.

Example 3.7.10. $\mathcal{E} = (E, Con, \vdash)$ *where* $E = \{a, b, c\}$, $\mathsf{Con} = \{\emptyset, \{a\}, \{b\}, \{a, c\}, \{b, c\}\}$, *and* $\emptyset \vdash a$, $\emptyset \vdash b$, $\{a\} \vdash c$, *and* $\{b\} \vdash c$ *can be represented by the domain* $\boldsymbol{D_s}(\mathcal{E})$ *seen in Figure 3.6.*

We now see that while in many ways SESs are more powerful than the event structures introduced earlier in this chapter, they lack asymmetric conflict as we have in AESs and EBESs. This is something we can consider adding in the reversible setting.

3.8 Conclusion

Event structures are a well-known model of true concurrency first introduced in [77]. Since then, more varieties of event structures with different levels of expressiveness have been introduced. In this chapter we have recalled prime event structures [77], general event structures [95], often called simply event structures, which replace conflict with a set of consistent sets of events and causation with sets of enabling events, asymmetric event structures [8], which make the conflict asymmetric, flow event structures [21] which allow an event to have multiple possible conflicting causes, bundle event structures [64] which replace causes with bundles, and extended bundle event structures [64] with bundles and asymmetric conflict. These, along with relations between them are shown in Figure 3.7, which extends Figure 3.1 with categories of bundle and extended bundle event structures.

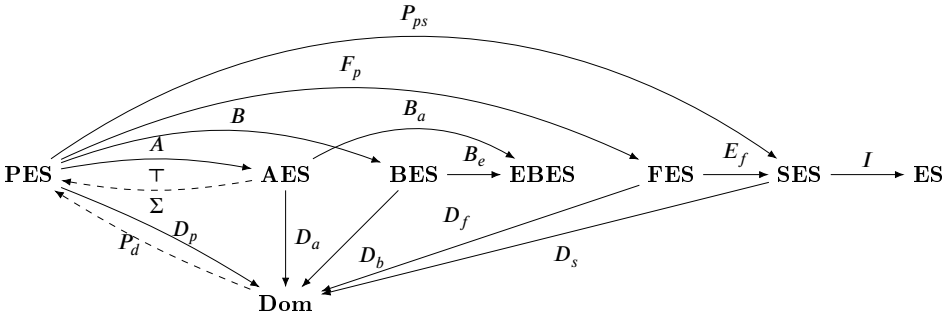


Figure 3.7: The categories and functors described in this chapter.

The categories of bundle and extended bundle event structures are the only ones in this chapter new to this work. In addition to defining the categories, we have constructed products and coproducts in both of these categories. The fact that we have functors between **BES** and the other categories helps show that our chosen notion of BES morphism is sensible. In addition to the functors, we have a mapping from BESs to FESs which is not a functor.

Having functors from **PES** and to **dom** for all event structure categories allows us to use P_d (Definition 3.2.5 [95]) to convert between all the categories we have defined. We are aiming to get a similar diagram of categories and functors between reversible event structures.

Since many of these categories of event structures have been used to define semantics of process calculi [21, 22, 31, 32, 64, 66, 94] and we want true concurrency semantics of reversible process calculi, it is natural to want a similar diagram of categories of reversible event structures to choose from. These functors help us determine which event structures to use depending on how expressive we need them to be. The products and coproducts defined in this section have also been used for describing the semantics of choice and parallel composition, which makes those a useful feature for our categories of reversible event structures to have. The figure shows how domains have been used to describe the configurations of many variants of event structures, and it will be necessary to have a reversible formalism which can serve the same purpose for reversible event structures.

The key result of this chapter is the categorical definitions of **BES** and **EBES** and that they fit well into the existing framework.

Chapter 4

Reversible Event Structures and Configuration systems

Reversible versions of various kinds of event structures were introduced in [82, 84]. Our aim here is to interpret these as objects in appropriate categories and study functors between them (as a foundation for using reversible event structures to give true concurrency semantics of reversible process calculi). Hitherto few reversible frameworks have been defined categorically, though [38] used category theory to describe the relationship between RCCS processes and their histories, and [25] used dagger categories to define a reversible process calculus called Π.

We formulate categories for the reversible prime event structures (RPESs) and reversible asymmetric event structures (RAESs) of [82] and for the reversible (general) event structures (RESs) of [84]. Additionally we define reversible variants of Flow, Bundle, and Extended Bundle Event Structures (RFES, RBES, and REBES) and categories of these. We define morphisms for each category and functors between them, which in some cases form adjunctions. We construct coproducts, and, in the case of RESs, RBESs, EREBESs, and a subcategory of RPESs (see Remark 4.2.22), products.

Although no functor is known from asymmetric event structures to general event structures such that the general event structure maps to the same domain as the asymmetric, we define a functor from reversible asymmetric event structures to reversible general event structures and show a correspondence between them (Theorem 4.7.23).

RPESs, RAESs, RFESs, RBESs, and REBESs differ from their forward-only variants by not only by adding the ability for events to reverse, but also by having additional causality and prevention relations describing when events are allowed to reverse. RESs on the other hand modify the enablings used to describe causation in general event structures, adding asymmetric conflict specific to each enabling in the form of a preventing set. Since the preventing set is specific to each enabling, RESs can have multiple different asymmetric conflicts associated with an event depending on the enabling being used. This is similar to how forward-only general event structures can have multiple different sets of causes, or enabling sets, associated with an event, unlike prime and asymmetric event structures, which can only have one.

Having functors between the different variants of event structures improves our ability to compare systems modelled by different kinds of event structures. Examples 4.0.1 and 4.0.3 show reversible prime and general event structures being used to model CCS and π-calculus processes respectively. Having a functor which maps the RPES created by the CCS process to an RES allows us to compare the processes (e.g. by defining morphisms between their corresponding RESs). We can also use the product or coproduct of the RESs to get a parallel composition or choice between the processes if they are intended to be part of a larger system.

Example 4.0.1 (Using an RPES to model reversible CCS)**.** *The simple reversible CCS process $a \mid \overline{a}.b$ can be modelled by an RPES which splits the b into two events, one caused by $\overline{a}$ and one caused by τ, wherein a and $\overline{a}$ are both in conflict with τ. We can also say that both b's prevent τ and $\overline{a}$ from reversing before they do. We return to this in Example 4.2.6.*

An RBES can model the process seen in Example 4.0.1 without creating multiple b events, as it allows a single b event to have multiple conflicting causes.

Example 4.0.2 (Using an RAES to model an error [82]). *Suppose we wish to model a computation wherein a series of events, e_0,e_1,e_2,... happen in a sequential fashion unless interrupted by an* error *event, which causes the computation to undo all events in reverse order and start over. We would need the* error *event to prevent the computation from continuing. However, we would not want any events from the computation to prevent the error from happening as we would have if we say* error *is in conflict with e_i in an RPES. To model this we need asymmetric conflict:* error *prevents e_i. We return to this in Example 4.3.4.*

The REBES adds asymmetric conflict to the RBES.

Example 4.0.3 (Using an RES to model π-calculus [33]). *Consider the π-calculus process $(\nu n)(\overline{a}\langle n\rangle \mid \overline{b}\langle n\rangle \mid c(x))$, wherein a bound name n can be extruded on either channel a or b after which it can be received by c. This creates an action with two possible causes, which are not mutually exclusive. If c receives n after both outputs have been performed then either output could have caused the input, since either output could have been what caused n to become free and therefore available for c to receive. Hence one can argue that to describe this process with a reversible event structure, we would need $\overline{a}\langle n\rangle$ or $\overline{b}\langle n\rangle$ to be able to cause $c(n)$. Also we would want either output to be able to reverse after $c(n)$ has been performed as long as the other output has also been performed and not reversed. We return to this in Example 4.7.2.*

Unlike their forward-only counterparts, reversible event structures do not necessarily have a notion of a configuration, since it can be difficult to determine which sets of events are reachable once we have non-causal reversibility. Instead, reversible event structures are mapped to *configuration systems* (CSs) [82]. CSs are a model of concurrency intended to serve a similar purpose as domains do for the forward-only event structures, letting the various kinds of reversible event structures be translated into one formalism. A CS consists of a set of events, a set of configurations of events, and transitions between them. We therefore also give a categorical definition of CSs and define functors from the described varieties of event structures to configuration systems.

We show that, just as domains can be modelled as event structures, so a subclass of CSs which we call *finitely enabled* can be modelled as RESs. We prove that our functors between RESs and finitely enabled CSs form an adjunction (Theorem 4.7.17). Furthermore they are inverses for finitely enabled CSs under certain conditions (Theorem 4.7.16). Having these mappings in both directions is very useful, as modelling a system as a CS can often be more intuitive than an event structure, especially an RES with many enablings.

We further define a subcategory of the RESs which are *stable* (RSESs), meaning that the causes of an event cannot be ambiguous. While Example 4.0.3 required the use of a non-stable RES, Example 4.0.4 below describes a case where we would want to use an RSES to describe a reversible computation. We also define stable CSs, and functors between them and stable RESs.

Example 4.0.4 (Using an RSES to model multiple errors). *Consider the scenario in Example 4.0.2, wherein a computation can be interrupted and forced to restart by an* error *event. We now modify it so that we have multiple* error *events, only one of which can happen at a time. Then we need to be able to say that an event e_i in the computation can be reversed if one of the mutually exclusive* error *events has happened. We cannot achieve this with RAESs, as having multiple* error *events cause the same event to reverse would imply all the* error *events need to have occurred before we can reverse. However with RSESs we can realise the desired behaviour by creating a separate enabling for each* error *event. Since only one* error *event can exist in any given configuration, this will be stable. We return to this in Example 4.8.2.*

We present a definition of normal form for RESs, which ensures that we only have the minimum enablings necessary to describe the corresponding CS. This in particular gives us mappings which are full inverses between stable finitely enabled CSs and stable RESs in normal form (Proposition 4.8.15). We also define a normaliser functor (Definition 4.8.19), which maps RESs to RESs in normal form. Some non-stable RESs become stable when normalised. Since there only exists one stable RES in normal form which describes any given stable CS, normalising two stable RESs which map to the same CS also gives us the same result. This makes normalising a useful tool for comparing stable RESs and to a lesser degree non-stable RESs.

With a few exceptions [83, 90], reversible process calculi have always adopted *causal* reversibility, where an action can be reversed if and only if all the actions caused by it have been reversed first. By contrast, the reversible event structures of [82,84] that we consider here allow non-causal reversibility, inspired by bonding in biochemical processes. Hence causal subcategories are of interest.

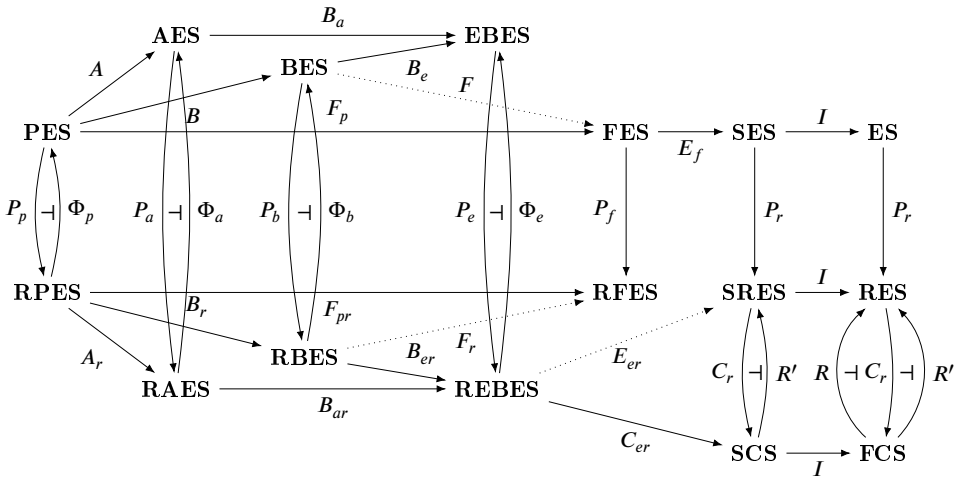


Figure 4.1: Event structure and configuration system categories and the functors between them.

We define subcategories of RESs which are (1) *cause-respecting*, meaning that no action can be reversed unless all the actions caused by it have been reversed first [82], which can be seen as a safety property for causal reversibility, and (2) *causal*, which means both cause-respecting and also that events whose consequences have all been reversed can themselves be reversed (a liveness property). Consider again Examples 4.0.1 and 4.0.2. The RPES in Example 4.0.1 is causal, but the error event in Example 4.0.2 makes the RAES non-causal, as the most recent e_i event cannot reverse, despite not having caused any other events, unless the error event is also present in the configuration.

Other models of concurrency have also been used to define semantics of reversible process calculi. The reversible π-calculus has been modelled using rigid families [34], and RCCS has been modelled using configuration structures [4]. Both of these took advantage of the fact that categorical definitions of rigid families and configuration structures exist, as they used the products and coproducts to model parallel composition and choice. However, while both rigid families and configuration structures can model causality, since neither was designed to be reversible, they cannot model situations such as the ones seen in Examples 4.0.2 and 4.0.4, where the reverse transitions are not the exact inverse of the forwards transitions.

Section 4.2 describes the category of the reversible variant of prime event structures, while Sections 4.3 to 4.7 cover reversible asymmetric, bundle, extended bundle, flow, and general event structures. Section 4.8 investigates stable variants of general reversible event structures and configuration systems. Finally Section 4.9 describes cause-respecting and causal variants of general reversible event structures and configuration systems. Figure 4.1 shows an overview of the categories and functors introduced in this paper.

4.1 Configuration Systems

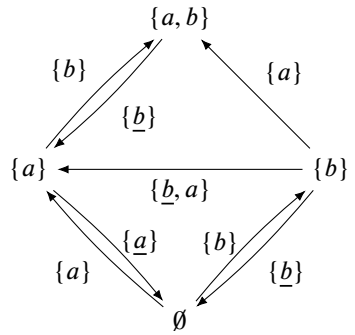
Before we start describing the various reversible event structures, we describe a model of concurrency, which we can translate the reversible event structures into. Domains cannot model reversibility, and so we need a different model. We adopt *configuration systems* [82] (Definition 4.1.1). These consist of a set of events, E, some of which, F, are reversible, a set C of configurations on these, and an optionally labelled transition relation $\rightarrow$ such that if $X \xrightarrow{A \cup \underline{B}} Y$ then the events of A can happen and the events of B can be undone in any order starting from configuration X, resulting in Y. When discussing reversible events we will use $\underline{e}$ to denote reversing e and e^* to denote that e may be performed or reversed and $X + e^*$ to denote $X \cup \{e\}$ if $e^* = e$ and $X \setminus \{e\}$ if $e^* = \underline{e}$. We may also leave out Y when describing such a transition, since it is implied that $Y = (X \setminus B) \cup A$. A $\overline{\text{CS}}$ is shown in Example 4.1.2.

We define the category of CSs and construct coproducts (Definition 4.1.5) and products (Definition 4.1.7). We also define reachability in CSs and a subcategory of CSs where every configuration is reachable (Definition 4.1.19) and a functor mapping CSs to their reachable subsystems.

Definition 4.1.1 (Configuration system [82]). *A* configuration system *(CS) is a quadruple $\mathcal{C} = (E, F, \mathsf{C}, \rightarrow)$ where E is a set of events, $F \subseteq E$ is a set of reversible events, $\mathsf{C} \subseteq 2^E$ is the set of configurations, and $\rightarrow \subseteq \mathsf{C} \times 2^{E \cup \underline{F}} \times \mathsf{C}$ is a labelled transition relation such that if $X \xrightarrow{A \cup \underline{B}} Y$ then:*

1. $A \cap X = \emptyset$;
2. $B \subseteq X \cap F$;
3. $Y = (X \setminus B) \cup A$;
4. *for all $A' \subseteq A$ and $B' \subseteq B$, we have $X \xrightarrow{A' \cup \underline{B'}} Z \xrightarrow{(A \setminus A') \cup \underline{(B \setminus B')}} Y$ (where $Z = (X \setminus B') \cup A' \in \mathsf{C}$).*

Example 4.1.2 (Configuration System). *The figure shows a CS with the events a and b. Notice that there exist reverse transitions, which remove events from the configuration, and that because we have a transition going directly from $\{b\}$ to $\{a\}$, we must be able to go from $\{b\}$ to $\{a\}$ both via $\emptyset$ and via $\{a, b\}$. However, despite there existing paths from $\emptyset$ to $\{a, b\}$ via $\{a\}$ and $\{b\}$, we do not require a direct transition $\emptyset \xrightarrow{\{a,b\}} \{a, b\}$. In future examples we will leave out the labels of the transitions to avoid clutter, as they are easily derived from the configurations.*



Even forwards CS transitions have a lot more freedom than domains, as their transitions are not derived from an ordering. The only requirement of their transitions is that when events can happen concurrently, they must also be able to happen in any order.

We introduce a notion of morphism in Definition 4.1.3, creating the category **CS**. The two first conditions in this definition ensure that morphisms preserve configurations and transitions, while the third condition ensures that two events which map to the same event cannot appear in the same configuration.

Definition 4.1.3 (CS morphism). *Let $\mathcal{C}_0 = (E_0, F_0, \mathsf{C}_0, \rightarrow_0)$ and $\mathcal{C}_1 = (E_1, F_1, \mathsf{C}_1, \rightarrow_1)$ be configuration systems. A CS morphism is a partial function $f : E_0 \to E_1$ such that*

1. *for any $X \in \mathsf{C}_0$, $f(X) \in \mathsf{C}_1$;*
2. *for any $X, Y \in \mathsf{C}_0$, $A \subseteq E_0$, and $B \subseteq F_0$, if $X \xrightarrow{A \cup \underline{B}}_0 Y$ and $f(A \cup B) \neq \emptyset$ then $f(X) \xrightarrow{f(A) \cup f(\underline{B})}_1 f(Y)$;*
3. *for all $e_0, e_0' \in E_0$, if $f(e_0) = f(e_0') \neq \perp$ and $e_0 \neq e_0'$ then there exists no $X \in \mathsf{C}_0$ such that $e_0, e_0' \in X$.*

Proposition 4.1.4. ***CS*** *where CSs are objects and CS morphisms are arrows, is a category.*

Proof. We prove this in Appendix B.1. The main challenge is showing that morphisms are composable. □

We construct a coproduct of two CSs in Definition 4.1.5. A coproduct acts as a 'choice' between which of the two CSs to behave like, as illustrated by Figure 4.2. We do this by making a CS which includes both sets of events, with their original configurations and transitions, including the ones going to and from the empty configuration, but not creating any configurations with events from both.

Definition 4.1.5 (CS coproduct). *Given CSs $\mathcal{C}_0 = (E_0, F_0, \mathsf{C}_0, \rightarrow_0)$ and $\mathcal{C}_1 = (E_1, F_1, \mathsf{C}_1, \rightarrow_1)$, we can construct $\mathcal{C}_0 + \mathcal{C}_1 = (E, F, \mathsf{C}, \rightarrow)$ with injections i_0 and i_1 where:*

1. $E = \{(0, e) \mid e \in E_0\} \cup \{(1, e) \mid e \in E_1\}$;
2. $F = \{(0, e) \mid e \in F_0\} \cup \{(1, e) \mid e \in F_1\}$;
3. *for $j \in \{0, 1\}$ and $e \in E_j$ we have $i_j(e) = (j, e)$;*
4. *$X \in \mathsf{C}$ iff $\exists X_0 \in \mathsf{C}_0 . i_0(X_0) = X$ or $\exists X_1 \in \mathsf{C}_1 . i_1(X_1) = X$;*

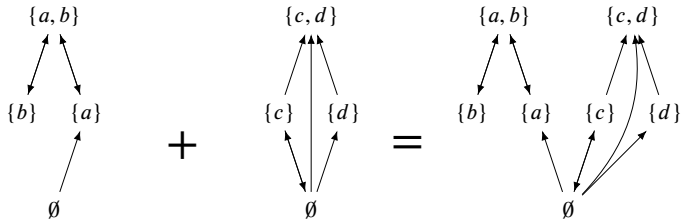


Figure 4.2: Example of a coproduct

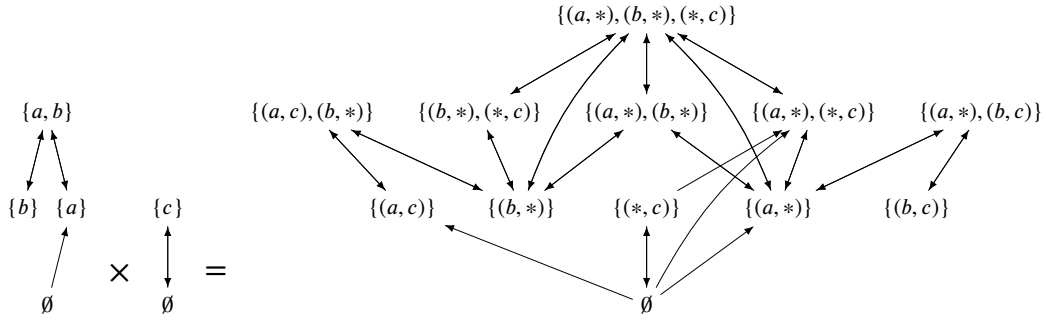


Figure 4.3: Example of a Product

5. $X \xrightarrow{A \cup \underline{B}} Y$ *iff there exist* $j \in \{0,1\}$, *and* $X_j, Y_j, A_j, B_j \subseteq E_j$ *such that* $i_j(X_j) = X$, $i_j(Y_j) = Y$, $i_j(A_j) = A$, $i_j(B_j) = B$, *and* $X_j \xrightarrow{A_j \cup \underline{B_j}}_j Y_j$.

Proposition 4.1.6. *Given CSs* $\mathcal{C}_0$ *and* $\mathcal{C}_1$, *then* $\mathcal{C}_0 + \mathcal{C}_1$ *is their coproduct in* $\boldsymbol{CS}$.

Proof. We show that we have the required morphisms in Appendix B.2. □

We also construct a product of CSs in Definition 4.1.7. A product can be described as the parallel composition of two CSs, as illustrated by Figure 4.3.

We construct the product by creating a CS where the events from one set of events can either happen on their own or synchronise with an event from the other CS. When an action happens in one CS without synchronising with an action from the other, we use $*$ instead of the other action. Configurations in this new CS must consist of events such that if one looks only at the events that came from one of the CSs, without considering what they synchronised with, they must be distinct and form a configuration in the original CS. Similarly, projecting a transition onto one of the original CSs, must yield a transition unless all the events being added or removed are events from the other CS that have not synchronised with anything.

Definition 4.1.7 (partially synchronous CS product)**.** *Given two CSs* $\mathcal{C}_0 = (E_0, F_0, \mathsf{C}_0, \to_0)$ *and* $\mathcal{C}_1 = (E_1, F_1, \mathsf{C}_1, \to_1)$, *we can construct* $\mathcal{C}_0 \times \mathcal{C}_1 = (E, F, \mathsf{C}, \to)$ *with projections* π_0 *and* π_1 *where:*

1. $E = E_0 \times_* E_1 = \{(e, *) \mid e \in E_0\} \cup \{(*, e) \mid e \in E_1\} \cup \{(e, e') \mid e \in E_0 \text{ and } e' \in E_1\}$;
2. $F = F_0 \times_* F_1 = \{(e, *) \mid e \in F_0\} \cup \{(*, e) \mid e \in F_1\} \cup \{(e, e') \mid e \in F_0 \text{ and } e' \in F_1\}$;
3. *for* $j \in \{0,1\}$ *and* $(e_0, e_1) \in E$ *we have* $\pi_j((e_0, e_1)) = e_j$.
4. $X \in \mathsf{C}$ *if:*
   (a) $\pi_0(X) \in \mathsf{C}_0$;
   (b) $\pi_1(X) \in \mathsf{C}_1$;
   (c) *for all* $e, e' \in X$, *if* $\pi_0(e) = \pi_0(e') \neq *$ *or* $\pi_1(e) = \pi_1(e') \neq *$ *then* $e = e'$.

5. $X \xrightarrow{A \cup B} Y$ *if:*

   (a) *if* $\pi_0(A \cup B) \neq \emptyset$ *then* $\pi_0(X) \xrightarrow{\pi_0(A) \cup \pi_0(B)}_0 \pi_0(Y)$;

   (b) *if* $\pi_1(A \cup B) \neq \emptyset$ *then* $\pi_1(X) \xrightarrow{\pi_1(A) \cup \pi_1(B)}_1 \pi_1(Y)$;

   (c) $B \subseteq X$.

Proposition 4.1.8. *If* C_0 *and* C_1 *are CSs then* $C_0 \times C_1$ *is their product in* $\boldsymbol{CS}$.

Proof. We show that we have the required morphisms in Appendix B.3. □

Functors exist from **PES**, **AES**, **BES**, **FES**, and **ES** to **CS** as described in Definitions 4.1.9 to 4.1.17. The first two are based on mappings from [82]. Of the conditions for transitions in Definition 4.1.9, (a) to (c) are part of Definition 4.1.1, and (d) simply ensures that e can only happen in transitions going from configurations that contain all of e's causes.

Definition 4.1.9 (From **PES** to **CS**). *The functor* $C_p : \boldsymbol{PES} \to \boldsymbol{CS}$ *is defined as*

1. $C_p((E, <, \sharp)) = (E, \emptyset, \mathsf{C}, \to)$ *where* $\mathsf{C} = \{X \subseteq E \mid X \text{ is conflict-free}\}$ *and* $X \xrightarrow{A} Y$ *if:*

   (a) $X, Y \in \mathsf{C}$;

   (b) $Y = X \cup A$;

   (c) $A \cap X = \emptyset$;

   (d) *for every* $e \in A$, $\{e' \in E \mid e' < e\} \subseteq X$.

2. $C_p(f) = f$.

Proposition 4.1.10. C_p *is a functor from* $\boldsymbol{PES}$ *to* $\boldsymbol{CS}$.

Proof. We show this in Appendix B.4. The main challenge is proving that C_p preserves morphisms. □

Definition 4.1.11 differs from Definition 4.1.9 in that it also requires that if e' prevents e, then e cannot be part of a transition which contains e' either as part of the start configuration or as one of the events being performed simultaneously.

Definition 4.1.11 (From **AES** to **CS**). *The functor* $C_a : \boldsymbol{AES} \to \boldsymbol{CS}$ *is defined as:*

1. $C_a((E, <, \lhd)) = (E, \emptyset, \mathsf{C}, \to)$ *where* $\mathsf{C} = \{X \subseteq E \mid \lhd \text{ is well-founded on } X\}$ *and* $X \xrightarrow{A} Y$ *if:*

   (a) $X, Y \in \mathsf{C}$;

   (b) $Y = X \cup A$;

   (c) $A \cap X = \emptyset$;

   (d) *for every* $e \in A$, $\{e' \in E \mid e' < e\} \subseteq X$ *and* $\{e' \in E \mid e' \rhd a\} \cap (X \cup A) = \emptyset$.

2. $C_a(f) = f$.

Proposition 4.1.12. C_a *is a functor from* $\boldsymbol{AES}$ *to* $\boldsymbol{CS}$.

Proof. We show this in Appendix B.5. The proof is similar to previous functors. □

Example 4.1.13 shows how the AES from Example 3.3.6 can be mapped to a CS.

Example 4.1.13. *The AES* $\mathcal{E} = (E, <, \lhd)$ *where* $E = \{a, b, c\}$ *and* $a < b$ *and* $b \lhd c$, *maps to the CS* $C_a(\mathcal{E})$, *shown in Figure 4.4. Notice that, unlike when we used the domain* $D_a(\mathcal{E})$ *to model* $\mathcal{E}$ *in Example 3.3.6, we sometimes get transitions with multiple events, such as* $\emptyset \xrightarrow{\{a,c\}} \{a, c\}$, *but at other times transitions such as* $\emptyset \to \{a\} \to \{a, b\}$ *exist without* $\emptyset \to \{a, b\}$, *allowing us to distinguish between concurrent and sequential events. We also get the unreachable configurations* $\{b\}$ *and* $\{b, c\}$, *which were not present in the domain.*

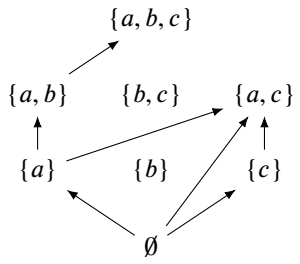


Figure 4.4: The CS generated by the AES described in Example 4.1.13.

Definition 4.1.14 (From **BES** to **CS**). *The functor* $C_b : \boldsymbol{BES} \to \boldsymbol{CS}$ *is defined as:*

1. $C_b((E, \mapsto, \sharp)) = (E, F, \mathsf{C}, \to)$ *where:*
   (a) $X \in \mathsf{C}$ *if* X *is conflict-free;*
   (b) *For* $X, Y \in \mathsf{C}$, *and* $A \subseteq E$, *there exists a transition* $X \xrightarrow{A} Y$ *if:*
      i. $Y = X \cup A$;
      ii. $X \cap A = \emptyset$;
      iii. $X \cup A$ *conflict-free;*
      iv. *for all* $e \in A$ *and* $X' \subseteq E$, *if* $X' \mapsto e$ *then* $X' \cap (X \setminus B) \neq \emptyset$.
2. $C_b(f) = f$.

Defining a functor from **FES** to **CS** is somewhat more complex than for previous event structure categories, since the condition that morphisms preserve configurations is explicit in Definition 3.6.4, rather than being implied, as in other event structure morphisms. In the generated CS, we want to allow configurations which do not fulfil all the conditions of Definition 3.6.2, so that we will have the same CS generated by an FES and a corresponding reversible FES in which no events can reverse. This makes it harder to reason about what Condition 4 of Definition 3.6.4 means for morphisms preserving unreachable transitions in CS representations of FESs.

Definition 4.1.15 (From **FES** to **CS**). *The functor* $C_f : \boldsymbol{FES} \to \boldsymbol{CS}$ *is defined as:*

1. $C_f((E, \prec, \sharp)) = (E, F, \mathsf{C}, \to)$ *where:*
   (a) $X \in \mathsf{C}$ *if* X *is conflict-free;*
   (b) *For* $X, Y \in \mathsf{C}$ *and* $A \subseteq E$, *there exists a transition* $X \xrightarrow{A} Y$ *if:*
      i. $Y = X \cup A$;
      ii. $X \cap A = \emptyset$;
      iii. *for all* $e \in A$, *if* $e' \prec e$ *and* $e' \notin X$, *then there exists* $e'' \in X$ *such that* $e' \sharp e''$ *and* $e'' \prec e$.
2. $C_f(f) = f$.

Proposition 4.1.16. C_f *is a functor from* $\boldsymbol{FES}$ *to* $\boldsymbol{CS}$.

Proof. We show this in Appendix B.6. The main challenge is proving that C_f preserves morphisms. □

We define a functor from **ES** to **CS** in Definition 4.1.17. The first three conditions on transitions are similar to the previous definitions of CSs generated from event structures, but condition (d) uses the enablings by stating that e can only be part of a transition if e is associated with an enabling set which is a subset of the starting configuration.

Definition 4.1.17 (From **ES** to **CS**). *The functor* $C : \boldsymbol{ES} \to \boldsymbol{CS}$ *is defined as:*

1. $\boldsymbol{C}((E, \mathsf{Con}, \vdash)) = (E, \emptyset, \mathsf{C}, \rightarrow)$ *where* $X \in \mathsf{C}$ *if for all* $X' \subseteq_{\text{fin}} X$, $X' \in \mathsf{Con}$, *and* $X \xrightarrow{A} Y$ *if:*
   (a) $X, Y \in \mathsf{C}$;
   (b) $Y = X \cup A$;
   (c) $A \cap X = \emptyset$;
   (d) *for every* $e \in A$, *there exists an* $X' \subseteq_{\text{fin}} X$ *such that* $X' \vdash e$.
2. $\boldsymbol{C}(f) = f$.

Proposition 4.1.18. $\boldsymbol{C}$ *is a functor from* $\boldsymbol{ES}$ *to* $\boldsymbol{CS}$.

Proof. We show this in Appendix B.7. The proof is similar to previous functors. □

The definitions of configurations are not the same here as they were when mapping to domains in Chapter 3, discarding the conditions which ensured the configurations would be reachable, and keeping only those concerned with conflict. This is because (1) CSs do not have the same requirements of the relationships between their transitions and configurations as domains, and (2) it will become much more difficult to tell which configurations are reachable (in a finite number of steps) once we start discussing reversible event structures. Therefore not restricting ourselves to reachable configurations now will mean that when we map our forward-only event structures into their reversible variants their CS representation will be preserved.

However, we will usually not be interested in any part of a CS which is not reachable from the empty configuration $\emptyset$, and so we will define a functor to remove those parts. For this we first define reachability in Definition 4.1.19.

Definition 4.1.19 (Reachable CS)**.** *Given a CS* $\mathcal{C} = (E, F, \mathsf{C}, \rightarrow)$, *a configuration* $X \in \mathsf{C}$ *is reachable if there exists some sequence* $\emptyset \xrightarrow{A_0 \cup \underline{B_0}} \ldots \xrightarrow{A_n \cup \underline{B_n}} X$. *A configuration* $X \in \mathsf{C}$ *is forwards-reachable if there exists some sequence* $\emptyset \xrightarrow{A_0} \ldots \xrightarrow{A_n} X$. *We say that* $\mathcal{C}$ *is reachable if all configurations in* C *are reachable. We let* $\boldsymbol{RCS}$ *be the category consisting of reachable CSs and the CS morphisms between them.*

We then define the reachable subsystem of a CS and a functor Reach restricting the CS to said subsystem in Definition 4.1.20.

Definition 4.1.20 (Restricting to Reachable)**.** *Given a CS* $\mathcal{C} = (E, F, \mathsf{C}, \rightarrow)$, *the* reachable part *of* $\mathcal{C}$, $(E, F, \mathsf{C}', \rightarrow')$, *is defined as:*

1. $\mathsf{C}' = \{X \mid X \in \mathsf{C},\ X \text{ is reachable}\}$;
2. $X \xrightarrow{A \cup \underline{B}}{}' X'$ *if* $X \xrightarrow{A \cup \underline{B}} X'$ *and* $X, X' \in \mathsf{C}'$.

We define a functor $\mathsf{Reach} : \boldsymbol{CS} \to \boldsymbol{RCS}$ *such that:*

1. $\mathsf{Reach}(\mathcal{C})$ *is the reachable part of* $\mathcal{C}$;
2. $\mathsf{Reach}(f) = f$.

Proposition 4.1.21. Reach *is a functor from* $\boldsymbol{CS}$ *to* $\boldsymbol{RCS}$.

Proof. First we show that Reach is a mapping, meaning:

1. If $\mathcal{C}$ is a CS then $\mathsf{Reach}(\mathcal{C})$ is an RCS.
2. If f is a CS morphism then $\mathsf{Reach}(f)$ is an RCS morphism.

This holds because:

1. Follows from Definition 4.1.1.

2. Here we need to show that for any CSs $\mathcal{C}_0 = (E_0, F_0, \mathsf{C}_0, \to_0)$ and $\mathcal{C}_1 = (E_1, F_1, \mathsf{C}_1, \to_1)$ and CS morphism $f : \mathcal{C}_0 \to \mathcal{C}_1$, if $X \in \mathsf{C}_0$ is reachable then $f(X)$ is reachable in C_1. But clearly, if $\emptyset \xrightarrow{A_0 \cup \underline{B_0}} \ldots \xrightarrow{A_n \cup \underline{B_n}} X$, then $\emptyset \xrightarrow{f(A_0) \cup \underline{f(B_0)}} \ldots \xrightarrow{f(A_n) \cup \underline{f(B_n)}} f(X)$.

   In addition, since all reachable configurations were configurations of the original CS, condition 3 of Definition 4.1.3 is clearly satisfied.

The rest of the proof is similar to previous functors. □

It should come to no surprise that Reach, being a restriction, is a right adjoint of the inclusion.

Proposition 4.1.22. *Let I be the inclusion functor $I : \boldsymbol{RCS} \to \boldsymbol{CS}$. Then $I \dashv \mathsf{Reach}$.*

Proof. For any CS $\mathcal{C} = (E, F, \mathsf{C}, \to)$ we define counit $\epsilon : I(\mathsf{Reach}(\mathcal{C})) \to \mathcal{C}$ such that for all $e \in E$, $\epsilon(e) = e$. Clearly ϵ is a CS morphism.

We then show that given an RCS $\mathcal{C}$, a CS $\mathcal{C}'$, and a morphism $f : I(\mathcal{C}) \to \mathcal{C}'$, then $f : \mathcal{C} \to \mathsf{Reach}(\mathcal{C}')$ is an RCS morphism. As established in the proof of Proposition 4.1.21, if X is a reachable configuration, then $f(X)$ is a reachable configuration, and the result follows. □

RCS is a full subcategory of **CS**, i.e. includes all the functors between reachable CSs. Moreover, Proposition 4.1.22 establishes that **RCS** is coreflective in **CS**, meaning its inclusion functor has a right adjoint.

4.2 Reversible Prime Event Structures

In this section we add reversibility to our PESs and define the categories and functors seen in Figure 4.5. *Reversible prime event structures* (Definition 4.2.1) consist of events, some of which may be reversible, causality and conflict similar to a PES (though conflict is not necessarily hereditary), reverse causality, which works similarly to causality, in that $e \prec \underline{e'}$ means e' can only be reversed in configurations containing e, and finally prevention, where if $e \rhd \underline{e'}$ then e prevents e' from reversing, meaning e' can only be reversed in configurations not containing e. Example 4.2.6 shows how we can use these to describe a reversible CCS process with causal reversibility. We define the category of RPESs, along with coproducts (Definition 4.2.8), and extend the mappings from PESs to RPESs, and from RPESs to PESs and CSs defined by [82] into functors. We show that the functors between RPESs and PESs form an adjunction (Proposition 4.2.16).

Definition 4.2.1 (RPES [82])**.** *A* reversible prime event structure *(RPES) is a sextuple $\mathcal{E} = (E, F, <, \sharp, \prec, \rhd)$ where E is the set of events and*

1. *$<$ is an irreflexive partial order such that for every $e \in E$, $\{e' \in E \mid e' < e\}$ is finite and conflict-free ;*
2. *$\sharp$ is irreflexive and symmetric;*
3. *if $e < e'$ then not $e \sharp e'$;*
4. *$F \subseteq E$ is the set of reversible events;*

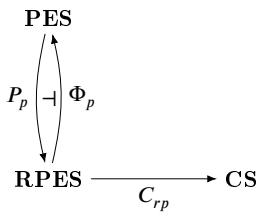


Figure 4.5: Functors introduced in this section.

5. $\rhd \subseteq E \times \underline{F}$ *is the prevention relation;*

6. $\prec \subseteq E \times \underline{F}$ *is the reverse causality relation where for each* $e \in F$, $e \prec \underline{e}$ *and* $\{e' \mid e' \prec \underline{e}\}$ *is finite and conflict-free;*

7. *if* $e \prec \underline{e'}$ *then not* $e \rhd \underline{e'}$*;*

8. $\sharp$ *is hereditary with respect to* sustained causation $\lessdot\!\!\lessdot$*, where* $e \lessdot\!\!\lessdot e'$ *means that* $e < e'$ *and if* $e \in F$ *then* $e' \rhd \underline{e}$*;*

9. $\lessdot\!\!\lessdot$ *is transitive.*

As previously, in order to define the category **RPES**, we need a notion of morphism, which we describe in Definition 4.2.2. An RPES morphism can be seen as a combination of a PES morphism for the forwards part and an AES morphism for the reverse part.

Definition 4.2.2 (RPES morphism)**.** *Let* $\mathcal{E}_0 = (E_0, F_0, <_0, \sharp_0, \prec_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, <_1, \sharp_1, \prec_1, \rhd_1)$ *be RPESs. An RPES morphism* $f : \mathcal{E}_0 \to \mathcal{E}_1$ *is a partial function* $f : E_0 \to E_1$ *such that*

1. *for all* $e \in E_0$*, if* $f(e) \neq \bot$ *then* $\{e_1 \mid e_1 <_1 f(e)\} \subseteq \{f(e') \mid e' <_0 e\}$*;*

2. *for all* $e, e' \in E_0$*, if* $f(e) \neq \bot \neq f(e')$ *and* $f(e) \sharp_1 f(e')$ *then* $e \sharp_0 e'$*;*

3. *for all* $e \in F_0$*, if* $f(e) \neq \bot$ *then* $\{e_1 \mid e_1 \prec_1 \underline{f(e)}\} \subseteq \{f(e') \mid e' \prec_0 \underline{e}\}$*;*

4. *for all* $e \in E_0$ *and* $e' \in F_0$*, if* $f(e) \neq \bot \neq f(e')$ *and* $f(e) \rhd_1 \underline{f(e')}$ *then* $e \rhd_0 \underline{e'}$ *;*

5. *for all* $e, e' \in E_0$*, if* $f(e) = f(e') \neq \bot$ *and* $e \neq e'$ *then* $e \sharp_0 e'$*;*

6. $f(F_0) \subseteq F_1$*.*

Proposition 4.2.3. ***RPES****, where RPESs are objects and RPES morphisms are arrows, is a category.*

Proof. We prove this in Appendix B.8. The main challenge is showing that morphisms are composable. □

In order to use CSs as a model for RPESs, [82] defined a mapping, which we extend to the functor $\boldsymbol{C_{rp}}$, from RPESs to CSs, illustrated in Figures 4.6 and 4.7. It uses conflict-free sets of events as configurations, and the transition relation is similar to the one generated by PESs and AESs, except that we add reverse transitions, which remove events instead of adding them, and treat reverse causality and prevention similarly to forward causality and asymmetric conflict.

Definition 4.2.4 (From **RPES** to **CS**)**.** *The functor* $\boldsymbol{C_{rp}} : \boldsymbol{RPES} \to \boldsymbol{CS}$ *is defined as*

1. $\boldsymbol{C_{rp}}((E, F, <, \sharp, \prec, \rhd)) = (E, F, \mathsf{C}, \to)$ *where* $\mathsf{C} = \{X \subseteq E \mid X \text{ is conflict-free}\}$*, and* $X \xrightarrow{A \cup \underline{B}} Y$ *if:*
   (a) $X, Y \in \mathsf{C}$*,* $A \subseteq E$*, and* $B \subseteq F$*;*
   (b) $Y = (X \setminus B) \cup A$*;*
   (c) $A \cap X = \emptyset$*;*
   (d) $B \subseteq X$*;*
   (e) $X \cup A$ *is conflict-free;*
   (f) *for every* $e \in A$*, if* $e' < e$ *then* $e' \in X \setminus B$*;*
   (g) *for every* $e \in B$*, if* $e' \prec \underline{e}$ *then* $e' \in X \setminus (B \setminus \{e\})$*;*
   (h) *for every* $e \in B$*, if* $e' \rhd \underline{e}$ *then* $e' \notin X \cup A$*.*

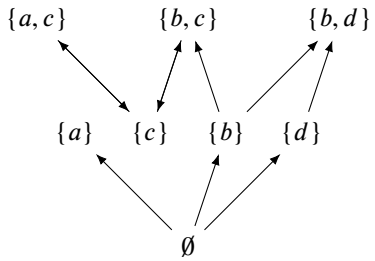
2. $\boldsymbol{C_{rp}}(f) = f$*.*

Proposition 4.2.5. $\boldsymbol{C_{rp}}$ *is a functor from* ***RPES*** *to* ***CS****.*

Proof. We show this in Appendix B.9. The main challenge is proving that $\boldsymbol{Crp}$ preserves morphisms. □

$\mathcal{E} = (E, F, <, \sharp, \prec, \rhd)$ where:
$E = \{a, b, c, d\}$
$F = \{a, b\}$
$b < c$
$a \sharp b$, $a \sharp d$, $c \sharp d$
$c \prec \underline{a}$, $c \prec \underline{b}$, $a \prec \underline{a}$, $b \prec \underline{b}$
$d \rhd \underline{b}$

(a) RPES $\mathcal{E}$

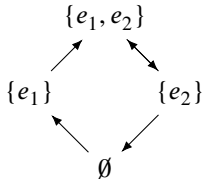


(b) The CS derived from $\mathcal{E}$

Figure 4.6: Example of an RPES and corresponding CS.

$\mathcal{E}' = (E', F', <', \sharp', \prec', \rhd')$ where:
$E' = \{e_1, e_2\}$
$F' = \{e_1, e_2\}$
$e_1 <' e_2$
$e_2 \prec' \underline{e_1}$, $e_1 \prec' \underline{e_1}$, $e_2 \prec' \underline{e_2}$
$e_1 \rhd' \underline{e_2}$

(a) RPES $\mathcal{E}'$



(b) The CS derived from $\mathcal{E}'$

Figure 4.7: Another RPES and corresponding CS.

We will now show how to model a process with an RPES and generate a CS from this RPES.

Example 4.2.6. *Here we want to model the reversible CCS process $a \mid \overline{a}.b$. This can be done by an RPES wherein we have two b events b_τ and $b_{\overline{a}}$ such that $\tau < b_\tau$ and $\overline{a} < b_{\overline{a}}$, $b_\tau \sharp b_{\overline{a}}$, $a \sharp \tau$, and $\overline{a} \sharp \tau$, and $b_\tau \rhd \underline{\tau}$ and $b_{\overline{a}} \rhd \underline{\overline{a}}$.*

Example 4.2.7 (RPES morphism)**.** *Consider the RPESs $\mathcal{E}$ from Figure 4.6 and $\mathcal{E}'$ from Figure 4.7. Then*

$$f(e) = \begin{cases} e_1 & \text{if } e \in \{a, b\} \\ e_2 & \text{if } e = c \end{cases}$$

is an RPES morphism between them, and a CS morphism between their associated CSs.

We also construct a coproduct of two RPESs in Definition 4.2.8. A coproduct of two RPESs, which can be represented by the CSs from Figure 4.2, is shown in Example 4.2.10. Similarly to Definition 4.1.5, we keep the two sets of events separate, and with the same causality, conflict, reverse causality, and prevention as they had in their respective original RPESs, but add conflict and prevention between the events that come from different RPESs.

Definition 4.2.8 (RPES coproduct)**.** *Given RPESs $\mathcal{E}_0 = (E_0, F_0, <_0, \sharp_0, \prec_0, \rhd_0)$ and $\mathcal{E}_1 = (E_1, F_0, <_1, \sharp_1, \prec_1, \rhd_1)$, we can construct $\mathcal{E}_0 + \mathcal{E}_1 = (E, F, <, \sharp, \prec, \rhd)$ with injections i_0 and i_1 where:*

1. $E = \{(0, e) \mid e \in E_0\} \cup \{(1, e) \mid e \in E_1\}$;
2. $F = \{(0, e) \mid e \in F_0\} \cup \{(1, e) \mid e \in F_1\}$;
3. $(j, e) < (j', e')$ *iff* $j = j'$ *and* $e <_j e'$;
4. $(j, e) \sharp (j', e')$ *iff* $j \neq j'$ *or* $e \sharp_j e'$;
5. $(j, e) \prec \underline{(j', e')}$ *iff* $j = j'$ *and* $e \prec_j \underline{e'}$;
6. $(j, e) \rhd \underline{(j', e')}$ *iff* $j \neq j'$ *or* $e \rhd_j \underline{e'}$;

7. *for* $e \in E_j$, $i_j(e) = (j, e)$ *for* $j \in \{0, 1\}$.

Proposition 4.2.9. *Given RPESs* $\mathcal{E}_0$ *and* $\mathcal{E}_1$, *we have* $\mathcal{E}_0 + \mathcal{E}_1$ *is their coproduct in* ***RPES***.

Proof. We prove that we have the required morphisms in Appendix B.10. □

Example 4.2.10 (RPES coproduct)**.** *Given RPESs corresponding to the CSs in Figure 4.2,* $\mathcal{E}_0 = (E_0, F_0, <_0, \sharp_0, \prec_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, <_1, \sharp_1, \prec_1, \rhd_1)$ *where* $E_0 = \{a, b\}$, $F_0 = \{a, b\}$, $a <_0 b$, $a \prec_0 \underline{b}$ *and* $E_1 = \{c, d\}$, $F_1 = \{c\}$, *and* $d \rhd_1 \underline{c}$, *the coproduct* $\mathcal{E}_0 + \mathcal{E}_1$ *is* $(E, F, <, \sharp, \prec, \rhd)$, *where*

$E = \{(0, a), (0, b), (1, c), (1, d)\}$,
$F = \{(0, a), (0, b), (1, c)\}$,
$(0, a) < (0, b)$,
$(0, a) \prec \underline{(0, b)}$,
$(0, a) \sharp (1, c)$, $(0, a) \sharp (1, d)$, $(0, b) \sharp (1, c)$, $(0, b) \sharp (1, d)$,
$(0, a) \rhd \underline{(1, c)}$, $(0, b) \rhd \underline{(1, c)}$, $(1, c) \rhd \underline{(0, a)}$, $(1, d) \rhd \underline{(0, a)}$, $(1, c) \rhd \underline{(0, b)}$, $(1, d) \rhd \underline{(0, b)}$, $(1, d) \rhd \underline{(1, c)}$

We do not give a way to construct a product of RPESs, as we have not found a notion of PES product which translates to a reversible setting. One way to construct a PES product can be seen in [91], where the definition is similar to mapping both event structures to domains, and mapping the product of those domains to a prime event structure. Since we do not have mappings from CSs to RPESs or RAESs, this is not a solution we can use.

Another way of looking at the PES product is that we can split an event, similar to in Example 4.2.6, based on which of the events of the other PES the event's causes synchronise with. However, splitting an event to allow it to have multiple causes does not work when dealing with reverse causation, as shown by Example 4.2.11. We shall see later in Remark 4.2.22 that splitting is not an issue in all RPESs, and so we can construct products in a subcategory.

Example 4.2.11. *Consider an RPES where we want* $\underline{a}$ *to be caused by either* b *or* c. *We attempt to accomplish this by splitting* a *into* a_b *and* a_c *such that* $b \prec \underline{a_b}$ *and* $c \prec \underline{a_c}$. *However, we would like to have a computation* $\emptyset \xrightarrow{a} \{a\}$ *after which we have both* $\{a\} \xrightarrow{b} \{a, b\} \xrightarrow{\underline{a}} \{b\}$ *and* $\{a\} \xrightarrow{c} \{a, c\} \xrightarrow{\underline{a}} \{c\}$ *are possible computations. We do not have this. By contrast, when splitting because of forwards computation, we know when we first perform the split event which of its causes is being used, and so we do not have this issue.*

We leave it as an open question whether products exist in general in the category **RPES**.

We also have functors between PES and RPES in Definitions 4.2.12 and 4.2.13, likewise based on mappings from [82]. Since $\sharp$ is only closed under conflict heredity with respect to sustained causation in an RPES, turning an RPES into a PES requires expanding conflict to be closed under all causality.

Definition 4.2.12 (From **PES** to **RPES**)**.** *The functor* $P_p : \boldsymbol{PES} \to \boldsymbol{RPES}$ *is defined as*

1. $P_p((E, <, \sharp)) = (E, \emptyset, <, \sharp, \emptyset, \emptyset)$;
2. $P_p(f) = f$.

Definition 4.2.13 (From **RPES** to **PES**)**.** *The functor* $\Phi_p : \boldsymbol{RPES} \to \boldsymbol{PES}$ *is defined as*

1. $\Phi_p((E, F, <, \sharp, \prec, \rhd)) = (E, <, \sharp')$ *where* $\sharp'$ *is* $\sharp$ *closed under conflict heredity and symmetry using the rules:*

$$\frac{a \sharp b}{a \sharp' b} \qquad \frac{a \sharp' b}{b \sharp' a} \qquad \frac{a \sharp' b < c}{a \sharp' c}$$

2. $\Phi_p(f) = f$.

For an example of an RPES being mapped to a PES, see Example 4.2.17.

Proposition 4.2.14. P_p *is a functor from* ***PES*** *to* ***RPES***.



(a) $C_{rp}(\mathcal{E})$ (b) $C_p(\Phi_p(\mathcal{E}))$

Figure 4.8: The CSs corresponding to the RPES and PES of Example 4.2.17.

Proof. Firstly, [82] showed that if $\mathcal{E}$ is a PES, then $P_p(\mathcal{E})$ is an RPES; so it remains to show that given a PES morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$, we get $f : P_p(\mathcal{E}_0) \to P_p(\mathcal{E}_1)$ is an RPES morphism. This is because $F_0 = F_1 = \emptyset$, and items 1, 2 and 6 of Definition 4.2.2 are implied by the definition of a PES morphism.

Having shown that P_p maps from **PES** to **RPES**, the rest of the proof is similar to previous functors. □

Proposition 4.2.15. *Φ_p is a functor from **RPES** to **PES**.*

Proof. We show this in Appendix B.11. The main challenge is proving that Φ_p preserves morphisms. □

While $\Phi_p(P_p(\mathcal{E})) = \mathcal{E}$ if $\mathcal{E}$ is a PES, $P_p(\Phi_p(\mathcal{E})) = \mathcal{E}$ only if $\mathcal{E}$ is an RPES with no reversible events [82]. However, Φ_p and P_p do form an adjunction. Hence we get Proposition 4.2.16. Applying first Φ_p and then P_p to an RPES always yields an under-estimation of the original, in that it is the original with all actions made irreversible.

Proposition 4.2.16. $P_p \dashv \Phi_p$.

Example 4.2.17. *For an example of how the functors C_{rp} and Φ_p work, we look at the RPES $\mathcal{E} = (E, F, <, \sharp, \prec, \rhd)$ where $E = \{a, b, c\}$, $F = \{a, b\}$, $a < c$, $a \sharp b$, $a \prec \underline{a}$, $b \prec \underline{b}$, $c \prec \underline{a}$, and $c \rhd \underline{b}$. This means that $\Phi_p(\mathcal{E}) = (E, <, \sharp')$ where $c \sharp' b$. The corresponding CSs $C_{rp}(\mathcal{E})$ and $C_p(\Phi_p(\mathcal{E}))$ can be seen in Figure 4.8.*

Notice that not only do the transitions involving reversal disappear, but because of hereditary conflict, so does the configuration $\{b, c\}$. Note also that $\{b, c\}$ was not forwards-reachable.

Since most reversible models restrict themselves to causal reversibility where actions cannot reverse if they have caused other actions, unlike RPESs which may allow other forms of reversibility as well, we define subcategories **crRPES** and **CRPES** of **RPES**, consisting of *cause-respecting* and *causal* RPESs as described in Definition 4.2.18 and the RPES morphisms between them.

Definition 4.2.18 (CRPES and crRPES [82])**.**

1. *A* cause-respecting *RPES (crRPES) $\mathcal{E} = (E, F, <, \sharp, \prec, \rhd)$ is an RPES such that for all $e \in E$ and $e' \in F$, if $e' < e$ then $e \rhd \underline{e'}$. In other words, all causation is sustained causation as described in Definition 4.2.1.*

2. *A* causal *RPES (CRPES) $\mathcal{E} = (E, F, <, \sharp, \prec, \rhd)$ is an RPES such that for all $e \in E$ and $e' \in F$, $e' < e$ if and only if $e \rhd \underline{e'}$, and $e \prec \underline{e'}$ if and only if $e = e'$.*

We can regard P_p as mapping to **CRPES** with empty set F. Consider the alternative mapping P'_p from [82], described as a functor below. This instead maps a PES to a CRPES where $F = E$, and the prevention is determined by the causality relation.

Definition 4.2.19 (From **PES** to **CRPES**)**.** *The functor $P'_p : \mathbf{PES} \to \mathbf{CRPES}$ is defined as*

1. *$P'_p((E, <, \sharp)) = (E, F, <, \sharp, \prec, \rhd)$ where $F = E$, $e' \prec \underline{e'}$ for all $e' \in F$, and $e \rhd \underline{e'}$ if and only if $e' < e$;*

2. *$P'_p(f) = f$.*

Proposition 4.2.20. *P'_p is a functor.*

Proof. According to [82], for any PES $\mathcal{E}$, $P'_p(\mathcal{E})$ is a CRPES.

We show that given a PES morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$, $f : P'_p(\mathcal{E}_0) \to P'_p(\mathcal{E}_1)$ is a CRPES morphism, satisfying the conditions of Definition 4.2.2: conditions 1, 2 and 5 are also conditions of a PES morphism and conditions 3, 4 and 6 are straightforward to prove.

Once established that P'_p is a mapping from **PES** to **CRPES**, the rest of the proof is similar to previous functors. □

We now show that P'_p forms a right adjoint of Φ_p between the categories **CRPES** and **PES**.

Proposition 4.2.21. $\Phi_p \dashv P'_p$.

Proof. For any CRPES $\mathcal{E} = (E, F, <, \sharp, \prec, \rhd)$, we have $P'_p(\Phi_p(\mathcal{E})) = (E, E, <, \sharp', \prec', \rhd')$. We define the unit $\eta : \mathcal{E} \to P'_p(\Phi_p(\mathcal{E}))$ as $\eta(e) = e$ for all $e \in E$.

We show that η is a CRPES morphism, satisfying the conditions of Definition 4.2.2:

1. Clearly $\{e \mid e < \eta(e')\} = \{\eta(e) \mid e < e'\}$.
2. Since all causation is sustained in $\mathcal{E}$, we have $\sharp = \sharp'$.
3. Suppose $e' \in F$ and $e \prec' \underline{\eta(e')}$. Then $e = e'$ and therefore $e \prec \underline{e'}$.
4. Suppose $e \in E$, $e' \in F$, and $e \rhd' \underline{e'}$. Then $e' < e$, and therefore $e \rhd \underline{e'}$.
5. No $e, e' \in E$ exist such that $\eta(e) = \eta(e')$ but $e \neq e'$.
6. Clearly $F \subseteq E$.

And since $\eta(e) = e$, clearly η is a natural transformation.

Let $\mathcal{E}$ be a CRPES, $\mathcal{E}'$ be a PES, and $g : \mathcal{E} \to P'_p(\mathcal{E}')$ be a CRPES morphism. If $\bar{g} : \Phi_p(\mathcal{E}) \to \mathcal{E}'$ uses the same mapping on events, then clearly $g(\mathcal{E}) = \Phi_p(\bar{g}) \circ \eta(\mathcal{E})$ and it is straightforward to show that $\bar{g}$ is a PES morphism. □

Remark 4.2.22. *Since it is a right adjunction, P'_p preserves products. We therefore get a product in **CRPES** by mapping to PESs using Φ_p, finding their product [91], and mapping back using P'_p.*

*Note that this adjunction only exists between **PES** and **CRPES**, not **RPES**, and therefore this method for constructing products only works for CRPESs.*

Since part of our motivation for defining RPESs was modelling reversible processes, we need to be able to label our events with a corresponding action from a process. For this we use a *labelled RPES* (LRPES). We will not be describing labelled variants of every reversible event structure, but only those we actually use for defining semantics. However, other labelled variants can be defined similarly.

Definition 4.2.23 (Labelled Reversible Prime Event Structure)**.** *A labelled RPES (LRPES) $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ consist of an RPES $(E, F, <, \sharp, \prec, \rhd)$, a set of labels Act, and a labelling function $\lambda : E \to \mathsf{Act}$.*

Definition 4.2.24 (LRPES morphism)**.** *Let $\mathcal{E}_0 = (E_0, F_0, <_0, \sharp_0, \prec_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$ be LRPESs. A LRPES morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$ is a partial function $f : E_0 \to E_1$ such that $f : (E_0, F_0, <_0, \sharp_0, \prec_0, \rhd_0) \to (E_1, F_1, <_1, \sharp_1, \prec_1, \rhd_1)$ is an RPES morphism and for all $e \in E_0$, either $f(e) = \bot$ or $\lambda_0(e) = \lambda_1(f(e))$.*

We now have our first category of reversible event structures. We have found that, since we have no way of distinguishing between a reversed event and an event that was never performed, we need asymmetric conflict to prevent events from reversing. We have also discovered that when dealing with non-causal reversibility it is not as easy to find the (reachable) configurations of an event structure as in the forward-only setting, since we can have a configurations which we can only reach by doing and later reversing an event as in Figure 4.7. This means we only define configurations as part of our functor from **RPES** to **CS**.

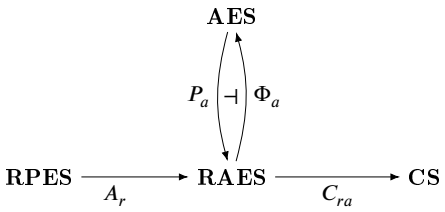


Figure 4.9: Functors introduced in this section.

4.3 Reversible Asymmetric Event Structures

As we did with PESs, we will now add reversibility to AESs and define the functors seen in Figure 4.9. *Reversible asymmetric event structures* (Definition 4.3.1) consist of events, some of which may be reversible, as well as causation $\prec$ and precedence $\lhd$, similar to an AES, except that $\prec$ is no longer a partial order, and instead just well-founded. In addition, both work on the reversible events, similarly to the RPES. One possible use of RAESs could be to model a scenario where an error causes an entire computation to reverse and start over, as illustrated by Example 4.3.4. We define the category of RAESs, along with coproducts (Definition 4.3.7), and extend the mappings from AESs and RPESs to RAESs, and from RAESs to AESs and CSs defined by [82] into functors. We show that the functors between RAESs and AESs form an adjunction (Proposition 4.3.14) similar to the one seen in Proposition 4.2.16.

Definition 4.3.1 (RAES [82])**.** *A* reversible asymmetric event structure *(RAES) is a quadruple* $\mathcal{E} = (E, F, \prec, \lhd)$ *where* E *is the set of events and*

1. $F \subseteq E$ *is the set of reversible events;*
2. $\lhd \subseteq (E \cup \underline{F}) \times E$ *is the irreflexive precedence relation;*
3. $\prec \subseteq E \times (E \cup \underline{F})$ *is the causation relation, which is irreflexive and well-founded, such that for all* $\alpha \in E \cup \underline{F}$, $\{e \in E \mid e \prec \alpha\}$ *is finite and has no* $\lhd$*-cycles;*
4. *for all* $e \in F$, $e \prec \underline{e}$*;*
5. *for all* $e \in E$ *and* $\alpha \in E \cup \underline{F}$ *if* $e \prec \alpha$ *then not* $e \rhd \alpha$*;*
6. $e \ll e'$ *implies* $e \lhd e'$*, where* $e \ll e'$ *means that* $e \prec e'$ *and if* $e \in F$ *then* $e' \rhd \underline{e}$*;*
7. $\ll$ *is transitive;*
8. *if* $e \sharp e'$ *and* $e \ll e''$ *then* $e'' \sharp e'$*, where* $\sharp = \lhd \cap \rhd$*.*

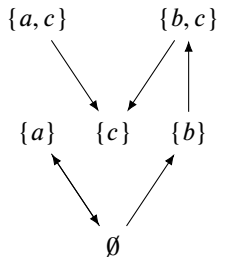
In order to model RAESs as CSs, [82] defined C_{ra}. The mapping from RAES to CS, C_{ra}, resembles C_{rp}, though as with AES configurations, the requirement of conflict-freedom is replaced with $\lhd$ being well-founded. Like C_{rp}, the mapping C_{ra} preserves morphisms, and is a functor. An example of applying C_{ra} to an RAES can be seen in Figure 4.10.

Definition 4.3.2 (From **RAES** to **CS**)**.** *The functor* $C_{ra} : \boldsymbol{RAES} \to \boldsymbol{CS}$ *is defined as :*

1. $C_{ra}((E, F, \prec, \lhd)) = (E, F, \mathsf{C}, \to)$ *where* $\mathsf{C} = \{X \subseteq E \mid \lhd$ *is well-founded on* $X\}$ *and* $X \xrightarrow{A \cup \underline{B}} Y$ *if*
   (a) $X, Y \in \mathsf{C}$, $A \subseteq E$, *and* $B \subseteq F$*;*
   (b) $Y = (X \setminus B) \cup A$*;*
   (c) $A \cap X = \emptyset$*;*
   (d) $B \subseteq X$*;*
   (e) *for every* $e \in A$*, if* $e' \prec e$ *then* $e' \in X \setminus B$*;*
   (f) *for every* $e \in A$*, if* $e' \rhd e$ *then* $e' \notin X \cup A$*;*

$\mathcal{E} = (E, F, \prec, \lhd)$ where:
$E = \{a, b, c\}$
$F = \{a, b\}$
$b \prec c$, $c \prec \underline{b}$
$a \rhd b$, $b \rhd a$, $c \rhd a$, $c \rhd b$

(a) RAES $\mathcal{E}$



(b) CS derived from $\mathcal{E}$

Figure 4.10: Example of an RAES and the corresponding CS.

(g) for every $e \in B$, if $e' \prec \underline{e}$ then $e' \in X \setminus (B \setminus \{e\})$;

(h) for every $e \in B$, if $e' \rhd \underline{e}$ then $e' \notin X \cup A$.

2. $C_{ra}(f) = f$.

Proposition 4.3.3. *C_{ra} is a functor from **RAES** to **CS**.*

Proof. We show this in Appendix B.13. The main challenge is proving that C_{ra} preserves morphisms. □

We give an example of a use of an RAES to model a computation and generate a CS.

Example 4.3.4. *We wish to describe a computation, $e_0, e_1, e_2, \ldots$, which can be stopped and forced to undo itself causally by an* error *event. To model this as an RAES we need for $i \geq 0$: $e_i \prec e_{i+1}$, $e_{i+1} \rhd \underline{e_i}$, $\mathsf{error} \prec \underline{e_i}$, $\mathsf{error} \rhd e_i$ and $e_i \rhd \underline{\mathsf{error}}$.*

We create a category **RAES** by defining RAES morphisms in Definition 4.3.5. This definition is nearly identical to that of an AES morphism, with the added condition that, like in the RPES morphism, reversible events can only synchronise with other reversible events.

Definition 4.3.5 (RAES morphism)**.** *Let $\mathcal{E}_0 = (E_0, F_0, \prec_0, \lhd_0)$ and $\mathcal{E}_1 = (E_1, F_1, \prec_1, \lhd_1)$ be RAESs. An RAES morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$ is a partial function $f : E_0 \to E_1$ such that*

1. *for all $e^* \in E_0 \cup \underline{F_0}$, if $f(e) \neq \bot$ then $\{e_1 \mid e_1 \prec_1 f(e^*)\} \subseteq \{f(e') \mid e' \prec_0 e^*\}$;*
2. *for all $e \in E_0$ and $e'^* \in E_0 \cup \underline{F_0}$, if $f(e) \neq \bot \neq f(e'^*)$ and $f(e'^*) \lhd_1 f(e)$ then $e'^* \lhd_0 e$;*
3. *for all $e, e' \in E_0$, if $f(e) = f(e') \neq \bot$ and $e \neq e'$ then $e \sharp_0 e'$;*
4. *$f(F_0) \subseteq F_1$.*

Proposition 4.3.6. ***RAES**, where RAESs are objects and RAES morphisms are arrows, is a category.*

Proof. Similar to Proposition 4.2.3. □

We construct a coproduct of RAESs in Definition 4.3.7, in much the same way as we did for RPESs.

Definition 4.3.7 (RAES coproduct)**.** *Given RAESs $\mathcal{E}_0 = (E_0, F_0, \prec_0, \lhd_0)$ and $\mathcal{E}_1 = (E_1, F_1, \prec_1, \lhd_1)$, we can construct $\mathcal{E}_0 + \mathcal{E}_1 = (E, F, \prec, \lhd)$ with injections i_0 and i_1 where:*

1. *$E = \{(0, e) \mid e \in E_0\} \cup \{(1, e) \mid e \in E_1\}$;*
2. *$F = \{(0, e) \mid e \in F_0\} \cup \{(1, e) \mid e \in F_1\}$;*
3. *$(j, e) \prec (j', e')^*$ iff $j = j'$ and $e \prec_j e'^*$;*
4. *$(j, e) \rhd (j', e')^*$ iff $j \neq j'$ or $e \rhd_j e'^*$;*

5. *for* $e \in E_j$, $i_j(e) = (j, e)$ *for* $j \in \{0, 1\}$.

Proposition 4.3.8. *Given RAESs* $\mathcal{E}_0$ *and* $\mathcal{E}_1$, *we have* $\mathcal{E}_0 + \mathcal{E}_1$ *is their coproduct in* ***RAES***.

Proof. We show that we have the required morphisms in Appendix B.12. □

In order to create functors from **RAES** to **AES** (Definition 4.3.10), based on a mapping from [82], we need to be able to split the causality and precedence relations into forwards and reverse parts.

Definition 4.3.9. *Given an RAES* $\mathcal{E} = (E, F, \prec, \lhd)$, *let*

1. $\prec_E = \prec \cap (E \times E)$ *and* $\prec_F = \prec \cap (E \times \underline{F})$;
2. $\lhd_E = \lhd \cap (E \times E)$ *and* $\lhd_F = \lhd \cap (E \times \underline{F})$.

We also define functors from **RPES** and **AES** to **RAES**, also based on mappings from [82]. To turn an RAES into an AES, we need to remove events with $< \cup \lhd$-cycles in their causes, as these are not allowed in the definition of an AES. Conditions 6-8 of Definition 4.3.1 of an RAES also only apply to sustained causation, while in an AES, all causation is sustained. We therefore expand the asymmetric conflict and causation relations to fulfil those conditions for all causation.

Definition 4.3.10 (From **RAES** to **AES**). *The functor* Φ_a : ***RAES*** $\to$ ***AES*** *is defined as:*

1. $\Phi_a((E, F, \prec, \lhd)) = (E', <', \lhd')$ *where* E' *is* E *where any event* e *that has a* $\lhd_E \cup \prec_E$*-cycle in* $\{e' \in E \mid e' = e \text{ or } e' \prec_E^+ e\}$ *has been removed,* $<' = \prec_E^+ \cap (E' \times E')$, *and* $\lhd'$ *is* $\lhd_E \cup <_E$ *closed under conflict heredity, using the rules*

$$\frac{a \lhd b}{a \lhd' b} \qquad \frac{a <' b}{a \lhd b} \qquad \frac{a \sharp' b <' c}{a \sharp' c}$$

2. $\Phi_a(f) = f \upharpoonright E'$.

Proposition 4.3.11. Φ_a *is a functor from* ***RAES*** *to* ***AES***.

Proof. We show this in Appendix B.14. The main challenge is proving that Φ_a preserves morphisms. □

Definition 4.3.12 (From **AES** to **RAES**). *The functor* P_a : ***AES*** $\to$ ***RAES*** *is defined as:*

1. $P_a((E, <, \lhd)) = (E, \emptyset, <, \lhd)$;
2. $P_a(f) = f$.

Proposition 4.3.13. P_a *is a functor from* ***AES*** *to* ***RAES***.

Proof. According to [82] if $\mathcal{E}$ is an AES, then $P_a(\mathcal{E})$ is an RAES.

If $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an AES morphism then $f : P_a(\mathcal{E}_0) \to \Phi_a(\mathcal{E}_1)$ is an RAES morphism because it satisfies the conditions of Definition 4.3.5 because $F_0 = F_1 = \emptyset$ and conditions 1, 2, and 3 are implied by Definition 3.3.3.

Having established that P_a is a mapping from **AES** to **RAES**, the rest of the proof is similar to previous functors.

□

The relationship between Φ_a and P_a mimics that of Φ_p and P_p in two ways: (1) $\Phi_a(P_a(\mathcal{E})) = \mathcal{E}$ if $\mathcal{E}$ is a AES, but $P_a(\Phi_a(\mathcal{E})) = \mathcal{E}$ only if $\mathcal{E}$ is an RAES with no reversible events [82], and (2) they form an adjunction, as described in Proposition 4.3.14.

Proposition 4.3.14. $P_a \dashv \Phi_a$.

Proof. The proof is similar to Proposition 4.2.16, also using counit $\epsilon(e) = e$ and $\bar{f} = f$. □

To map an RPES to an RAES, we must combine the forwards and reverse causation into one relation. We also need to define an asymmetric conflict relation. In the functor from PES to AES, this was done by adding the causation to the conflict, but here we only add the sustained causation, in order to fulfil Condition 6 of Definition 4.3.1. We need to add the prevention as well.

Definition 4.3.15 (From **RPES** to **RAES**). *The functor* A_r : ***RPES*** $\rightarrow$ ***RAES*** *is defined as:*

1. $A_r((E, F, <, \sharp, \prec, \rhd)) = (E, F, < \cup \prec, \ll \cup \sharp \cup \lhd)$;
2. $A_r(f) = f$.

Proposition 4.3.16. *A_r is a functor from* ***RPES*** *to* ***RAES****.*

Proof. We show this in Appendix B.15. The main challenge is proving that A_r preserves morphisms. □

We define subcategories **crRAES** and **CRAES** consisting of the cause-respecting and causal RAESs defined in Definition 4.3.17 and the RAES morphisms between them.

Definition 4.3.17 (CRAES and crRAES [82])**.**

1. *A* cause-respecting *RAES (crRAES) $\mathcal{E} = (E, F, \prec, \lhd)$ is an RAES such that for all $e \in E$ and $e' \in F$, if $e' \prec e$ or $e \rhd e'$ then $e \rhd \underline{e'}$.*
2. *A* causal *RAES (CRAES) $\mathcal{E} = (E, F, \prec, \lhd)$ is an RAES such that for all $e \in E$ and $e' \in F$, $e \rhd \underline{e'}$ if and only if $e' \prec e$ or $e \rhd e'$, and $e \prec \underline{e'}$ if and only if $e = e'$.*

The functor P_a maps AESs to CRAESs with no reversible events, and we define an alternative functor, P'_a, based on a mapping from [82], which maps AESs to CRAESs where all events are reversible.

Definition 4.3.18 (From **AES** to **CRAES**). *The functor P'_a is defined as*

1. $P'_a((E, <, \lhd)) = (E, F, \prec, \lhd')$ *where* $F = E$, $e \prec e'^*$ *if* $e < e'^*$ *or* $e'^* = \underline{e}$, *and* $e^* \lhd' e'$ *if* $e \lhd e'$;
2. $P'_a(f) = f$.

Proposition 4.3.19. *P'_a is a functor from* ***AES*** *to* ***CRAES****.*

Proof. According to [82], if $\mathcal{E}$ is an AES, then $P'_a(\mathcal{E})$ is a CRAES.

Given an AES morphism, $f : \mathcal{E}_0 \rightarrow \mathcal{E}_1$, it is straightforward to show that $f : P'_a(\mathcal{E}_0) \rightarrow P'_a(\mathcal{E}_1)$ is a CRAES morphism, satisfying the conditions of Definition 4.3.5.

Having established that P'_a is a mapping from **AES** to **CRAES**, the rest of the proof is similar to previous functors. □

We show that P'_a forms a right adjoint of Φ_a between the categories **CRAES** and **AES**.

Proposition 4.3.20. $\Phi_a \dashv P'_a$.

Proof. The proof is similar to Proposition 3.3.11. See Appendix B.16 for full details. □

We now have a second category of reversible event structures, **RAES**. RAESs are somewhat simpler to define than RPESs because we do not split the causation and conflict into forwards and reverse, but we have found that this is not an issue in the categorical definition since our morphisms can also treat them both the same way. We have found that the category **RAES** relates to its corresponding forward-only category, **AES**, in much the same way as **RPES** and **PES**.

4.4 Reversible Bundle Event Structures

We define reversible bundle structures (RBES) by extending the bundle relation to map to reverse events and adding a prevention relation, similar to RPESs. Unlike RPESs, we do not split our bundles into separate relations for causing forwards and reverse events. We also introduce the functors seen in Figure 4.11.

Definition 4.4.1 (Reversible Bundle Event Structure)**.** *A reversible bundle event structure (RBES) is a 5-tuple $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ where:*

1. *E is the set of events;*

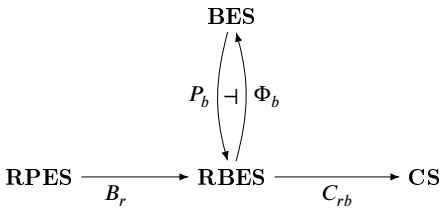


Figure 4.11: Functors introduced in this section.

2. *$F \subseteq E$ is the set of reversible events;*
3. *the bundle set, $\mapsto \subseteq 2^E \times (E \cup \underline{F})$, satisfies $X \mapsto e^* \Rightarrow \forall e_1, e_2 \in X. e_1 \neq e_2 \Rightarrow e_1 \sharp e_2$ and for all $e \in F$, $\{e\} \mapsto \underline{e}$;*
4. *the conflict relation, $\sharp \subseteq E \times E$, is symmetric and irreflexive;*
5. *$\triangleright \subseteq E \times \underline{F}$ is the prevention relation.*

Once again, in order to get a categorical definition of RBESs, we define a morphism in Definition 4.4.2.It is very similar to Definition 3.4.4, and treats prevention in the same way as conflict.

Definition 4.4.2 (RBES-morphism)**.** *Given RBESs $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \triangleright_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \triangleright_1)$, an RBES-morphism from $\mathcal{E}_0$ to $\mathcal{E}_1$ is a partial function $f : E_0 \to E_1$ such that $f(F_0) \subseteq F_1$ and for all $e, e' \in E_0$:*

1. *if $f(e) \sharp_1 f(e')$ then $e \sharp_0 e'$;*
2. *if $f(e) = f(e')$ and $e \neq e'$ then $e \sharp_0 e'$;*
3. *for $X_1 \subseteq E_1$ if $X_1 \mapsto_1 f(e)^*$ then there exists $X_0 \subseteq E_0$ such that $X_0 \mapsto_0 e^*$, $f(X_0) \subseteq X_1$, and if $e' \in X_0$ then $f(e') \neq \bot$;*
4. *if $f(e) \triangleright_1 \underline{f(e')}$ then $e \triangleright_0 \underline{e'}$.*

Proposition 4.4.3. ***RBES*** *consisting of RBESs and RBES-morphisms is a category.*

Proof. Partial functions are associative, and $f(e) = e$ functions as a an identity arrow, and the morphisms are composable. □

As we did for the reversible prime and asymmetric event structures, we want to model RBESs as Configuration structures (CSs), and therefore define a functor from one category to the other in Definition 4.4.4 and prove it to be a functor in Proposition 4.4.5.

Definition 4.4.4 (From **RBES** to **CS**)**.** *The functor $C_{rb} : \boldsymbol{RBES} \to \boldsymbol{CS}$ is defined as:*

1. *$C_{rb}((E, F, \mapsto, \sharp, \triangleright)) = (E, F, \mathsf{C}, \to)$ where*
   (a) *$X \in \mathsf{C}$ if X is conflict-free;*
   (b) *For $X, Y \in \mathsf{C}$, $A \subseteq E$, and $B \subseteq F$, there exists a transition $X \xrightarrow{A \cup \underline{B}} Y$ if:*
      i. *$Y = (X \setminus B) \cup A$;*
      ii. *$X \cap A = \emptyset$;*
      iii. *$B \subseteq X$;*
      iv. *$X \cup A$ is conflict-free;*
      v. *for all $e \in B$, if $e' \triangleright \underline{e}$ then $e' \notin X \cup A$;*
      vi. *for all $e \in A$ and $X' \subseteq E$, if $X' \mapsto e$ then $X' \cap (X \setminus B) \neq \emptyset$;*
      vii. *for all $e \in B$ and $X' \subseteq E$, if $X' \mapsto \underline{e}$ then $X' \cap (X \setminus (B \setminus \{e\})) \neq \emptyset$.*

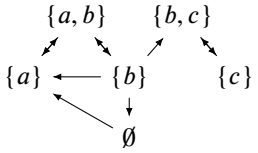


Figure 4.12: The configurations of the RBES in Example 4.4.6.

2. $\boldsymbol{C}_{rb}(f) = f$.

Proposition 4.4.5. *$\boldsymbol{C}_{br}$ is a functor from **RBES** to **CS**.*

Proof. We show this in Appendix B.17. The main challenge is proving that $\boldsymbol{C}_{br}$ preserves morphisms. □

Example 4.4.6 shows an RBES mapped to a CS. The configuration $\{b, c\}$ is reachable despite b being required for c to happen, and c being a possible cause of b. In future examples we will leave out bundles on the form $\{e\} \mapsto \underline{e}$, since they can be assumed to exist for any $e \in F$.

Example 4.4.6 (RBES)**.** *An RBES $\mathcal{E} = (E, F, \mapsto, \sharp, \triangleright)$ where $E = \{a, b, c\}$, $F = \{a, b\}$, $a \sharp c$, $\{a, c\} \mapsto b$, $\{b\} \mapsto c$, $\{a\} \mapsto \underline{a}$, $\{b\} \mapsto \underline{a}$, and $\{b\} \mapsto \underline{b}$, gives the CS $\boldsymbol{C}_{rb}(\mathcal{E})$ seen in Figure 4.12.*

We again construct a product of RBESs in Definition 4.4.7. This is similar to the product of BESs with the addition that events prevent each other is either of their components prevent each other.

Definition 4.4.7 (RBES product)**.** *Given RBESs $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \triangleright_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \triangleright_1)$, we construct $\mathcal{E}_0 \times \mathcal{E}_1 = (E, F, \mapsto, \sharp, \triangleright)$ with projections π_0, π_1 where:*

1. *$E = E_0 \times_* E_1 = \{(e, *) \mid e \in E_0\} \cup \{(*, e) \mid e \in E_1\} \cup \{(e, e') \mid e \in E_0$ and $e' \in E_1\}$;*
2. *$F = F_0 \times_* F_1 = \{(e, *) \mid e \in F_0\} \cup \{(*, e) \mid e \in F_1\} \cup \{(e, e') \mid e \in F_0$ and $e' \in F_1\}$;*
3. *for $(e_0, e_1) \in E$, $\pi_i((e_0, e_1) = e_i$;*
4. *for any $e^* \in E \cup \underline{F}$, $X \subseteq E$, $X \mapsto e^*$ iff there exists $i \in \{0, 1\}$ and $X_i \subseteq E_i$ such that $X_i \mapsto \pi_i(e)^*$ and $X = \{e' \in E \mid \pi_i(e') \in X_i\}$;*
5. *for any $e, e' \in E$, $e \sharp e'$ iff there exists $i \in \{0, 1\}$ such that $\pi_i(e) \sharp_i \pi_i(e')$, or $\pi_i(e) = \pi_i(e') \neq \bot$ and $\pi_{1-i}(e) \neq \pi_{1-i}(e')$;*
6. *for any $e \in E$, $e' \in F$, $e \triangleright \underline{e'}$ iff there exists $i \in \{0, 1\}$ such that $\pi_i(e) \triangleleft_i \underline{\pi_i(e')}$.*

Proposition 4.4.8. *Given RBESs $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \triangleright_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \triangleright_1)$, we have $\mathcal{E}_0 \times \mathcal{E}_1$ is their product.*

Proof. Similar to the proof of Proposition 3.4.10. □

Example 4.4.9. *Consider the RBESs $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \triangleright_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \triangleright_1)$ where $E_0 = F_0 = \{a_0\}$ and $E_1 = \{a_1, b_1, c_1\}$, $F_1 = \{a_1, b_1\}$, $a_1 \sharp_1 c_1$, $\{a_1, c_1\} \mapsto_1 b_1$, and $\{c_1\} \mapsto_1 \underline{b_1}$, with the configurations seen in Figure 4.13.*

Then $\mathcal{E}_0 \times \mathcal{E}_1 = (E, F, \mapsto, \sharp, \triangleright)$ where

$E = \{(a_0, *), (a_0, a_1), (a_0, b_1), (a_0, c_1), (*, a_1), (*, b_1), (*, c_1)\}$

$F = \{(a_0, *), (a_0, a_1), (a_0, b_1), (*, a_1), (*, b_1)\}$

$(a_0, a_1) \sharp (a_0, b_1)$	$(a_0, b_1) \sharp (a_0, c_1)$	$(a_0, a_1) \sharp (a_0, c_1)$
$(a_0, a_1) \sharp (a_0, *)$	$(a_0, b_1) \sharp (a_0, *)$	$(a_0, c_1) \sharp (a_0, *)$
$(a_0, a_1) \sharp (*, a_1)$	$(a_0, b_1) \sharp (*, b_1)$	$(a_0, c_1) \sharp (*, c_1)$
$(a_0, a_1) \sharp (a_0, c_1)$	$(*, a_1) \sharp (*, c_1)$	$(a_0, a_1) \sharp (*, c_1)$
$(*, a_1) \sharp (a_0, c_1)$		
$\{(a_0, a_1), (a_0, c_1), (*, a_1), (*, c_1)\} \mapsto (a_0, b_1)$	$\{(a_0, c_1), (*, c_1)\} \mapsto \underline{(*, b_1)}$	$\{(a_0, c_1), (*, c_1)\} \mapsto \underline{(a_0, b_1)}$
$\{(a_0, a_1), (a_0, c_1), (*, a_1), (*, c_1)\} \mapsto (*, b_1)$		

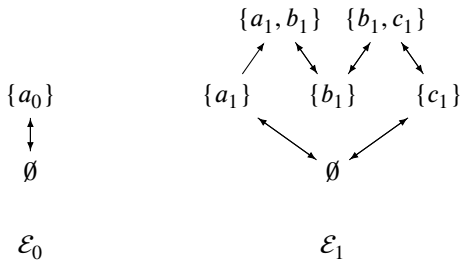


Figure 4.13: Configurations of RBESs in Example 4.4.9.

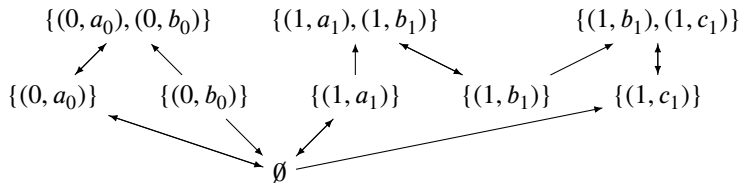


Figure 4.14: Configurations of RBES in Example 4.4.12.

We also construct a coproduct of RBESs in Definition 4.4.10. This again works similarly to the coproduct of BESs and treats prevention the same as RPESs.

Definition 4.4.10 (RBES coproduct)**.** *Given RBESs* $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1)$*, we construct* $\mathcal{E}_0 + \mathcal{E}_1 = (E, F, \mapsto, \sharp, \rhd)$ *with injections* i_0*,* i_1 *where:*

- $E = \{(0, e) \mid e \in E_0\} \cup \{(1, e) \mid e \in E_1\}$*;*
- $F = \{(0, e) \mid e \in F_0\} \cup \{(1, e) \mid e \in F_1\}$*;*
- *for* $e \in E_j$*,* $i_j(e) = (j, e)$ *for* $j \in \{0, 1\}$*;*
- $X \mapsto (j, e)^*$ *iff for all* $(j', e') \in X$*,* $j = j'$ *and* $i_j(X) \mapsto_j e^*$*;*
- $(j, e) \sharp (j', e')$ *iff* $j \neq j'$ *or* $e \sharp_j e'$*;*
- $(j, e) \rhd \underline{(j', e')}$ *iff* $j \neq j'$ *or* $e \rhd_j \underline{e'}$*.*

Proposition 4.4.11. *If* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *are RBESs, then* $\mathcal{E}_0 + \mathcal{E}_1$ *is their coproduct.*

Proof. Similar to the BES coproduct (Proposition 3.4.14). □

We now give an example of constructing a coproduct between two RBESs.

Example 4.4.12. *Consider again the RBESs* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *from Example 4.4.9. We can also find* $\mathcal{E}_0 + \mathcal{E}_1 = (E, F, \mapsto, \sharp, \rhd)$ *where:*

$E = \{(0, a_0), (0, b_0), (1, a_1), (1, b_1), (1, c_1)\}$
$F = \{(0, a_0), (0, b_0), (1, a_1), (1, b_1)\}$
$(1, a_1) \sharp (1, c_1)$
$\{(0, a_0)\} \mapsto (0, b_0)$
$\{(1, a_1), (1, c_1)\} \mapsto (1, b_1)$
$\{(1, c_1)\} \mapsto \underline{(1, b_1)}$
$(0, b_0) \rhd \underline{(0, a_0)}$

with the configurations seen in Figure 4.14.

We define cause-respecting and causal variants of RBES in Definition 4.4.13. The CRBES is slightly different from other causal event structures, in that the reversal of an event $\underline{e}$ can have other events in its causes, as long as each bundle contains e.

Definition 4.4.13 (cause-respecting and causal RBES)**.** *We say that $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ is a cause-respecting RBES (crRBES) if whenever $X \mapsto e$ and $e' \in X \cap F$, then $e \rhd \underline{e'}$.*

We say that $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ is a causal RBES (CRBES) if (1) if $e \rhd \underline{e'}$ then either $e \sharp e'$ or there exists an $X \subseteq E$ such that $X \mapsto e$ and $e' \in X$, (2) if $X \mapsto e$ and $e' \in X \cap F$, then $e \rhd \underline{e'}$, and (3) if $X \mapsto \underline{e}$ then $e \in X$.

We then need to prove the standard results from causal and cause-respecting systems.

Proposition 4.4.14. *Given a crRBES, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ and corresponding CS $\mathbf{C}_{rb}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, any reachable configuration $C \in \mathsf{C}$ is forwards-reachable.*

Proof. There exists a trace $\emptyset \xrightarrow{\{e_0^*\}} C_0 \xrightarrow{\{e_1^*\}} C_1 \ldots \xrightarrow{\{e_n^*\}} C_n$ where $C_n = C$. Clearly $e_0^* = e_0$, and C_0 is forwards-reachable, and we will show that if C_j is forwards-reachable for $0 \leq j \leq i$, then C_{i+1} is forwards-reachable.

If $e_{i+1}^* = e_{i+1}$, then the trace is forward and C_{i+1} is forwards-reachable.

If $e_{i+1}^* = \underline{e_i}$, then $C_{i+1} = C_{i-1}$, which is by definition forwards-reachable.

If $e_{i+1}^* = \underline{e_j}$ for some $0 \leq j < i$, then for all $0 \leq j' \leq i$, since $e_{j'} \not\rhd \underline{e_j}$, there does not exist $X \subseteq E$ such that $e_j \in X$ and $X \mapsto e_{j'}$. This means we get $\emptyset \xrightarrow{\{e_0^*\}} C_0 \xrightarrow{\{e_1^*\}} C_1 \ldots \xrightarrow{\{e_{j-1}^*\}} C_{j-1} \xrightarrow{\{e_{j+1}^*\}} C_{j+1} \setminus \{e_j\} \ldots \xrightarrow{\{e_i^*\}} C_{i+1}$. □

Proposition 4.4.15.

1. *If $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ is a crRBES and $\mathbf{C}_{rb}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$ then whenever $X \in \mathsf{C}$ is a reachable configuration and $X \xrightarrow{B} Y$, there exists a transition $Y \xrightarrow{\underline{B}} X$.*

2. *If $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ is a CRBES and $\mathbf{C}_{rb}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$ then whenever $X \in \mathsf{C}$, $X \xrightarrow{A \cup \underline{B}} Y$ and $A \cup B \subseteq F$, there exists a transition $Y \xrightarrow{B \cup \underline{A}} X$.*

Proof. We prove this in Appendix B.18 by showing the existence of these transitions according to Definition 4.4.4. □

We define functors between **BES** and **RBES**, and show that they form an adjunction, meaning that, as in the previous reversible categories, applying first Φ_b going from **RBES** to **BES** and then P_b going from **BES** to **RBES** to an RBES always yields an under-estimation of the original, in that it is the original with all actions made irreversible.

Definition 4.4.16 (**BES** to **RBES**)**.** *The functor $P_b : \boldsymbol{BES} \rightarrow \boldsymbol{RBES}$ is defined as:*

1. $P_b((E, \mapsto, \sharp)) = (E, \emptyset, \mapsto, \sharp, \emptyset \times \emptyset)$;
2. $P_b(f) = f$.

Definition 4.4.17 (**RBES** to **BES**)**.** *The functor $\Phi_b : \boldsymbol{RBES} \rightarrow \boldsymbol{BES}$ is defined as:*

1. $\Phi_b((E, F, \mapsto, \sharp, \rhd)) = (E, \mapsto \cap (2^E \times E), \sharp)$;
2. $\Phi_b(f) = f$.

Proposition 4.4.18. $P_b \dashv \Phi_b$.

Proof. We prove this similarly to previous adjunctions in Appendix B.19. □

We also wish to relate RBESs to RPESs, so we define a functor from **RPES** to **RBES**. Like $\boldsymbol{B}$ (Definition 3.4.15), $\boldsymbol{B}_r$ creates bundles with single events. This shows that RBESs are more expressive than RPESs, just as in the forward-only setting.

Definition 4.4.19 (**RPES** to **RBES**). *The mapping* $B_r : \boldsymbol{RPES} \to \boldsymbol{RBES}$ *is defined as:*

1. $B_r((E, F, <, \sharp, \prec, \rhd)) = (E, F, \mapsto, \sharp, \rhd)$, *where* $\{e'\} \mapsto e$ *if* $e' < e$ *and* $\{e'\} \mapsto \underline{e}$ *if* $e' \prec \underline{e}$;
2. $B_r(f) = f$.

Since part of our motivation for defining RBESs was modelling reversible processes, we need to be able to label our events with a corresponding action from a process. For this we use a *labelled RBES* (LRBES).

Definition 4.4.20 (Labelled Reversible Bundle Event Structure). *A labelled RBES (LRBES)* $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ *consist of an RBES* $(E, F, \mapsto, \sharp, \rhd)$, *a set of labels* Act, *and a labelling function* $\lambda : E \to \mathsf{Act}$.

Definition 4.4.21 (LRBES morphism). *Let* $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$ *be LRBESs. A LRBES morphism* $f : \mathcal{E}_0 \to \mathcal{E}_1$ *is a partial function* $f : E_0 \to E_1$ *such that* $f : (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0) \to (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1)$ *is an RBES morphism and for all* $e \in E_0$, *either* $f(e) = \bot$ *or* $\lambda_0(e) = \lambda_1(f(e))$.

This gives us a good picture of the category of bundle event structures. Like RAESs, we decided not to split the bundle relation into separate forwards and reverse bundle, but since we do not want asymmetric conflict between the forwards events, we have split the conflict similarly to RPESs into symmetric conflict and reverse prevention. Our definition of BES-morphism from Definition 3.4.4 turned out to translate well to the reverse setting, and even allowed this to be the first of our reversible event structure categories where the forward-only notion of product translated to the reversible setting.

4.5 Reversible Extended Bundle Event Structures

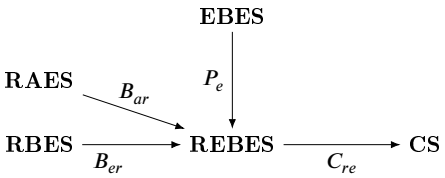


Figure 4.15: Functors introduced in this section.

We define a reversible version of EBESs in Definition 4.5.2, extending the asymmetric conflict and bundles to act on reversible events similarly to RBESs. We also introduce the functors seen in Figure 4.15. Adding asymmetric conflict to RBESs lets us to model scenarios such as Example 4.5.1.

Example 4.5.1 (The necessity of REBESs for modelling rollback). *Consider* $a.b \mid \overline{a}_\gamma.\mathsf{roll}\ \gamma$, *where* $\mathsf{roll}\ \gamma$ *means undo the action labelled* γ, *that is* $\overline{a}$, *and everything caused by it before continuing. To model this we would need an event* $\mathsf{roll}\ \gamma$, *and to split* b *into two different events depending on whether it needs to be reversed during the rollback or not. This would give us an RBES* $(\{a, \tau, \overline{a}, b_a, b_\tau, \mathsf{roll}\ \gamma\}, \{a, \tau, \overline{a}, b_a, b_\tau, \mathsf{roll}\ \gamma\}, \mapsto, \sharp, \rhd)$ *where* $\{a\} \mapsto b_a$, $\{\tau\} \mapsto b_\tau$, $\{\overline{a}, \tau\} \mapsto \mathsf{roll}\ \gamma$, $\{\mathsf{roll}\ \gamma\} \mapsto \underline{\tau}$, $\{\mathsf{roll}\ \gamma\} \mapsto \underline{\overline{a}}$, $\{\mathsf{roll}\ \gamma\} \mapsto \underline{b_\tau}$, $a \sharp \tau$, $\overline{a} \sharp \tau$, $b_a \rhd \underline{a}$, $b_\tau \rhd \underline{\tau}$, $\overline{a} \rhd \mathsf{roll}\ \gamma$, *and* $\tau \rhd \mathsf{roll}\ \gamma$. *This would indeed ensure that* $\overline{a}$ *and the events caused by it could only reverse if one of the* roll *events had occurred, but it would not force them to do so before doing anything else. For this we use asymmetric conflict:* $\mathsf{roll}\ \gamma \rhd \overline{a}$, $\mathsf{roll}\ \gamma \rhd \tau$, $\mathsf{roll}\ \gamma \rhd b_\tau$, *giving us a CS with the* reachable configurations *shown in Figure 4.16.*

Definition 4.5.2 (Reversible Extended Bundle Event Structure). *A reversible extended bundle event structure (REBES) is a triple* $\mathcal{E} = (E, F, \mapsto, \rhd)$ *where:*

1. E *is the set of events;*
2. $F \subseteq E$ *is the set of reversible events;*
3. $\mapsto \subseteq 2^E \times (E \cup \underline{F})$ *is the bundle set, satisfying* $X \mapsto e \Rightarrow \forall e_1, e_2 \in X.(e_1 \neq e_2 \Rightarrow e_1 \lhd e_2)$, *and for all* $e \in F$, $\{e\} \mapsto \underline{e}$;

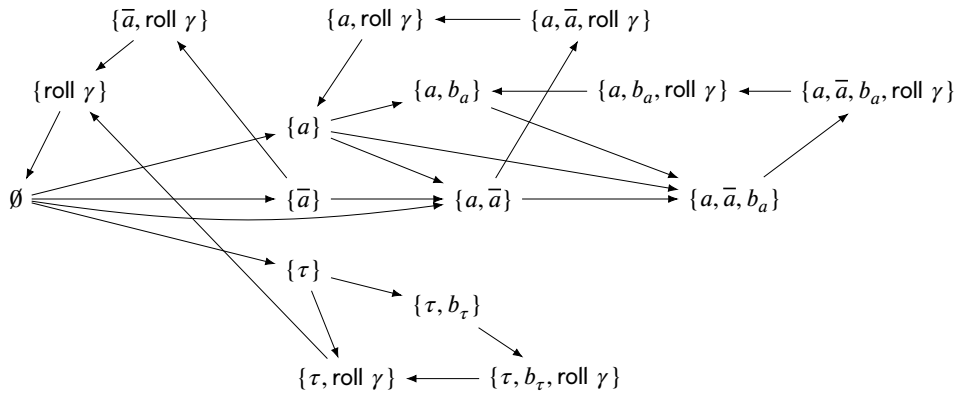


Figure 4.16: The reachable configurations of the REBES described in Example 4.5.1.

4. $\lhd \subseteq (E \cup \underline{F}) \times E$ *is the asymmetric conflict relation, which is irreflexive.*

In order to define REBES-morphisms, we extend the RBES-morphisms definition in the expected way, as seen in Definition 4.5.3. Like in the RAES-morphism we do not need to distinguish between preventing a reversal and forwards asymmetric conflict.

Definition 4.5.3 (REBES-morphism)**.** *Given REBESs* $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \rhd_1)$*, an REBES-morphism from* $\mathcal{E}_0$ *to* $\mathcal{E}_1$ *is a partial function* $f : E_0 \to E_1$ *such that* $f(F_0) \subseteq F_1$ *and for all* $e, e' \in E_0$*:*

1. *if* $f(e) = f(e')$ *and* $e \neq e'$ *then* $e \sharp_0 e'$*;*
2. *for* $X_1 \subseteq E_1$ *if* $X_1 \mapsto_1 f(e)^*$ *then there exists* $X_0 \subseteq E_0$ *such that* $f(X_0) \subseteq X_1$*, if* $e' \in X_0$ *then* $f(e') \neq \bot$*, and* $X \mapsto_0 e^*$*;*
3. *if* $f(e) \lhd_1 f(e')^*$ *then* $e \lhd_0 \underline{e'}^*$*.*

We again model REBESs as CSs, defining a functor in Definition 4.5.4.

Definition 4.5.4 (From **REBES** to **CS**)**.** *The functor* $C_{re} : \boldsymbol{REBES} \to \boldsymbol{CS}$ *is defined as:*

1. $C_{re}((E, F, \mapsto, \rhd)) = (E, F, \mathsf{C}, \to)$ *where:*
   (a) $X \in \mathsf{C}$ *if* $\lhd$ *is well-founded on* X*;*
   (b) *For* $X, Y \in \mathsf{C}$*,* $A \subseteq E$*, and* $B \subseteq F$*, there exists a transition* $X \xrightarrow{A \cup \underline{B}} Y$ *if:*
      i. $Y = (X \setminus B) \cup A$*;*
      ii. $X \cap A = \emptyset$*;*
      iii. $B \subseteq X$*;*
      iv. *for all* $e^* \in A \cup \underline{B}$*, if* $e' \rhd e^*$ *then* $e' \notin X \cup A$*;*
      v. *for all* $e \in A$ *and* $X' \subseteq E$*, if* $X' \mapsto e$ *then* $X' \cap (X \setminus B) \neq \emptyset$*;*
      vi. *for all* $e \in B$ *and* $X' \subseteq E$*, if* $X' \mapsto \underline{e}$ *then* $X' \cap (X \setminus (B \setminus \{e\})) \neq \emptyset$*.*
2. $C_{re}(f) = f$*.*

Example 4.5.5 shows an REBES which cannot be represented by an RBES, since you get a transition $\emptyset \to \{a\}$, but no $\{b\} \to \{a, b\}$, despite $\{a, b\}$ being a configuration.

Example 4.5.5 (REBES)**.** *An REBES* $\mathcal{E} = (E, F, \mapsto, \rhd)$ *where* $E = \{a, b, c\}$*,* $F = \{a, b\}$*,* $\{a, c\} \mapsto b$*,* $\{b\} \mapsto c$ $\{a\} \mapsto \underline{a}$*,* $\{b\} \mapsto \underline{a}$*,* $\{b\} \mapsto \underline{b}$*,* $a \rhd c$*,* $c \rhd a$*, and* $b \rhd a$ *gives the CS in Figure 4.17.*

We again construct a product of REBESs in Definition 4.5.6 and a coproduct in Definition 4.5.8 and prove they work as intended.

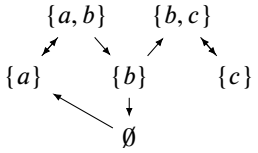


Figure 4.17: The configurations of the REBES in Example 4.5.5.

Definition 4.5.6 (REBES product)**.** *Given BESs* $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \rhd_1)$*, we construct* $\mathcal{E}_0 \times \mathcal{E}_1 = (E, F, \mapsto, \rhd)$ *with projections* π_0*,* π_1 *where:*

1. $E = E_0 \times_* E_1 = \{(e, *) \mid e \in E_0\} \cup \{(*, e) \mid e \in E_1\} \cup \{(e, e') \mid e \in E_0 \text{ and } e' \in E_1\}$;
2. $F = F_0 \times_* F_1 = \{(e, *) \mid e \in F_0\} \cup \{(*, e) \mid e \in F_1\} \cup \{(e, e') \mid e \in F_0 \text{ and } e' \in F_1\}$;
3. *for* $(e_0, e_1) \in E$, $\pi_i((e_0, e_1) = e_i$;
4. *for any* $e^* \in E \cup \underline{F}$, $X \subseteq E$, $X \mapsto e^*$ *iff there exists* $i \in \{0, 1\}$ *and* $X_i \subseteq E_i$ *such that* $X_i \mapsto \pi_i(e)^*$ *and* $X = \{e' \in E \mid \pi_i(e') \in X_i\}$;
5. *for any* $e \in E$, $e'^* \in E \cup \underline{F}$, $e \rhd e'^*$ *iff there exists* $i \in \{0, 1\}$ *such that* $\pi_i(e) \lhd_i \pi_i(e')^*$.

Proposition 4.5.7. *Given REBESs* $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \rhd_1)$*, then* $\mathcal{E}_0 \times \mathcal{E}_1 = (E, F, \mapsto, \rhd)$ *is their product.*

Proof. Similar to the proof of Proposition 3.5.7. □

Definition 4.5.8 (REBES coproduct)**.** *Given REBESs* $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \rhd_1)$*, we construct* $\mathcal{E}_0 + \mathcal{E}_1 = (E, F, \mapsto, \rhd)$ *with injections* i_0*,* i_1 *where:*

- $E = \{(0, e) \mid e \in E_0\} \cup \{(1, e) \mid e \in E_1\}$;
- $F = \{(0, e) \mid e \in F_0\} \cup \{(1, e) \mid e \in F_1\}$;
- *for* $e \in E_j$, $i_j(e) = (j, e)$ *for* $j \in \{0, 1\}$;
- $X \mapsto (j, e)^*$ *iff for all* $(j', e') \in X$, $j = j'$ *and* $i_j(X) \mapsto_j e^*$;
- $(j, e)^* \lhd (j', e')$ *iff* $j \neq j'$ *or* $e^* \lhd_j e'$.

Proposition 4.5.9. *If* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *are REBESs, then* $\mathcal{E}_0 \times \mathcal{E}_1$ *is their coproduct.*

Proof. Similar to the proof of BES coproduct (Proposition 3.4.14). □

The definition of a cause-respecting and causal REBES (Definition 4.5.10) is practically identical to that of a crRBES and CRBES.

Definition 4.5.10 (cause-respecting and causal REBES)**.** $\mathcal{E} = (E, F, \mapsto, \rhd)$ *is a crREBES if whenever* $X \mapsto e$ *and* $e' \in X$ *or* $e \rhd e'$*, then* $e \rhd \underline{e'}$.

$\mathcal{E} = (E, F, \mapsto, \rhd)$ *is a causal REBES (CREBES) if (1) if* $e \rhd \underline{e'}$ *then either* $e \sharp e'$ *or there exists an* $X \subseteq E$ *such that* $X \mapsto e$ *and* $e' \in X$*, (2) if* $e \rhd e'$ *or* $X \mapsto e$ *and* $e' \in X \cap F$*, then* $e \rhd \underline{e'}$*, and (3) if* $X \mapsto \underline{e}$ *then* $e \in X$.

We then prove some standard results for cause-respecting and causal systems.

Proposition 4.5.11. *Given a CREBES,* $\mathcal{E} = (E, F, \mapsto, \rhd)$ *and corresponding CS* $\mathbf{C}_{re}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$*, any reachable* $C \in \mathsf{C}$ *is forwards-reachable.*

Proof. Similar to the corresponding proof for RBESs (Proposition 4.4.14). □

Proposition 4.5.12.

1. *If* $\mathcal{E} = (E, F, \mapsto, \rhd)$ *is a crREBES and* $\boldsymbol{C}_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$ *then whenever* $X \in \mathsf{C}$ *is a reachable configuration,* $X \xrightarrow{\underline{B}} Y$, $Y \xrightarrow{B} X$.

2. *If* $\mathcal{E} = (E, F, \mapsto, \rhd)$ *is a CREBES and* $\boldsymbol{C}_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$ *then whenever* $X \in \mathsf{C}$, $X \xrightarrow{A \cup \underline{B}} Y$ *and* $A \cup B \subseteq F$, *we get a transition* $Y \xrightarrow{B \cup \underline{A}} X$.

Proof. Similar to the corresponding proofs for RAESs [82] and RBESs (Proposition 4.4.15). □

We define a simple functor from **RBES** to **REBES**. We can create asymmetric conflict from symmetric conflict by making it go both ways, similarly to the functor $\boldsymbol{A}$ from PESs to AESs (Definition 3.3.7).

Definition 4.5.13 (**RBES** to **REBES**). *The mapping* $\boldsymbol{B_{er}} : \boldsymbol{RBES} \to \boldsymbol{REBES}$ *is defined as:*

1. $\boldsymbol{B_{er}}((E, F, \mapsto, \sharp, \rhd)) = (E, F, \mapsto, \lhd')$, *where* $e \lhd' e'$ *if* $e \sharp e'$ *and* $\underline{e} \lhd' e'$ *if* $e' \rhd \underline{e}$;

2. $\boldsymbol{B_{er}}(f) = f$.

We define functors between **EBES** and **REBES**, which form an adjunction. These are similar to the previous functors between reversible and forward-only event structures.

Definition 4.5.14 (**EBES** to **REBES**). *The functor* $\boldsymbol{P_e} : \boldsymbol{EBES} \to \boldsymbol{REBES}$ *is defined as:*

1. $\boldsymbol{P_e}((E, \mapsto, \rhd)) = (E, \emptyset, \mapsto, \lhd)$;

2. $\boldsymbol{P_e}(f) = f$.

Definition 4.5.15 (**REBES** to **EBES**). *The functor* $\boldsymbol{\Phi_e} : \boldsymbol{REBES} \to \boldsymbol{EBES}$ *is defined as:*

1. $\boldsymbol{\Phi_e}((E, F, \mapsto, \rhd)) = (E, \mapsto_E, \lhd_E)$;

2. $\boldsymbol{\Phi_e}(f) = f$.

Proposition 4.5.16. $\boldsymbol{P_e} \dashv \boldsymbol{\Phi_e}$.

Proof. Similar to the proof of Proposition 4.4.18. □

As REBESs use asymmetric conflict, we can create a functor from **RAES** to **REBES** in Definition 4.5.17.

Definition 4.5.17 (**RAES** to **REBES**). *The mapping* $\boldsymbol{B_{ar}} : \boldsymbol{RAES} \to \boldsymbol{REBES}$ *is defined as:*

1. $\boldsymbol{B_{ar}}((E, F, \prec, \lhd)) = (E, F, \mapsto, \rhd)$, *where* $\{e'\} \mapsto e^*$ *if* $e' \prec e^*$;

2. $\boldsymbol{B_{ar}}(f) = f$.

Since we intend to use our REBESs for modelling the semantics of Rollback, we need a labelled variant, much as we have of other event structures.

Definition 4.5.18 (Labelled Reversible Extended Bundle Event Structure (LREBES)). *A labelled REBES (LREBES)* $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$ *consist of an REBES* $(E, F, \mapsto, \rhd)$, *a set of labels* Act, *and a labelling function* $\lambda : E \to \mathsf{Act}$.

Definition 4.5.19 (LREBES-morphism). *Given two LREBESs* $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$, *a LREBES morphism* $f : \mathcal{E}_0 \to \mathcal{E}_1$ *is a partial function* $f : E_0 \to E_1$ *such that* $f : (E_0, F_0, \mapsto_0, \rhd_0) \to (E_1, F_1, \mapsto_1, \rhd_1)$ *is an REBES-morphism and for all* $e \in E_0$, *either* $f(e) = \bot$ *or* $\lambda_0(e) = \lambda_1(f(e))$.

We now have a category of REBESs, which we can use to model rollback and other reversible systems with disjunctive causes and asymmetric conflict. Like in the previous section, we found that the EBES-morphism (Definition 3.5.3) translated well to the reversible setting and allowed us to define a product. We will see that not every category of event structures translates as well to the reversible setting.

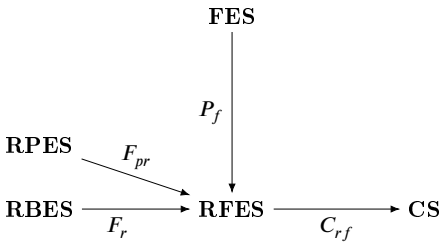


Figure 4.18: Functors introduced in this section.

4.6 Reversible Flow Event Structures

Since FESs resemble PESs, we use a similar approach to defining reversible versions in Definition 4.6.1: We extend the flow relation to also allow $e \prec \underline{e'}$, and add a prevention relation similar to that of RPES and RAES. We also introduce the functors seen in Figure 4.18.

Definition 4.6.1 (Reversible Flow Event Structure)**.** *A RFES is a 5-tuple $\mathcal{E} = (E, F, \prec, \sharp, \rhd)$ where:*

1. *E is the set of events;*
2. *$F \subseteq E$ is the set of reversible events;*
3. *the flow relation, $\prec \subseteq E \times (E \cup \underline{F})$, is irreflexive and $e \prec \underline{e}$ for all $e \in F$;*
4. *the conflict relation, $\sharp \subseteq E \times E$, is symmetric;*
5. *$\rhd \subseteq E \times \underline{F}$ is the prevention relation.*

Unlike other reversible variants of event structures, this definition does not guarantee that any configuration from which e can be undone contains e, as long as it contains an e' such that $e \sharp e'$ and $e' \prec e$. We instead rely on the mapping to CSs to ensure that e can only be reversed if it has actually happened.

As before, we need a way to map RFESs to CSs, which we define in Definition 4.6.2.

Definition 4.6.2 (From RFES to CS)**.** *The mapping C_{rf} is defined as $C_{rf}((E, F, \prec, \sharp, \rhd)) = (E, F, \mathsf{C}, \rightarrow)$ where:*

1. *$X \in \mathsf{C}$ if X is conflict-free;*
2. *For $X, Y \in \mathsf{C}$, $A \subseteq E$, and $B \subseteq F$, there exists a transition $X \xrightarrow{A \cup \underline{B}} Y$ if:*
   (a) *$Y = (X \setminus B) \cup A$;*
   (b) *$X \cap A = \emptyset$;*
   (c) *$B \subseteq X$;*
   (d) *$X \cup A$ conflict-free;*
   (e) *for all $e \in B$, if $e' \rhd \underline{e}$ then $e' \notin X \cup A$;*
   (f) *for all $e \in A$, if $e' \prec e$ and $e' \notin X \setminus B$, then there exists $e'' \in X \setminus B$ such that $e' \sharp e''$ and $e'' \prec e$;*
   (g) *for all $e \in B$, if $e' \prec \underline{e}$ and $e' \notin X \setminus (B \setminus \{e\})$, then there exists $e'' \in X \setminus (B \setminus \{e\})$ such that $e' \sharp e''$ and $e'' \prec \underline{e}$.*

This is not a functor, as it does not preserve morphisms in Example 4.6.3. This issue comes from not being able to find an equivalent of Condition 4 in Definition 3.6.4 for the reversible setting.

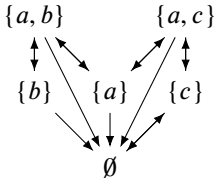


Figure 4.19: CS generated by an RFES.

Example 4.6.3. *Consider the RFESs* $\mathcal{E}_0 = (E_0, F_0, \prec_0, \sharp_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \prec_1, \sharp_1, \rhd_1)$ *where* $E_0 = F_0 = \{a, b, c\}$, $a \prec_0 \underline{b}$, $c \prec_0 \underline{b}$, *and* $a \sharp_0 c$, *and* $e_1 = \{a', b'\}$ *and* $a' \prec_1 b'$. *Between these RFESs we have the morphism*

$$f(e) = \begin{cases} a' & \text{if } e = a \\ b' & \text{if } e = b \end{cases}$$

However, while the CS $\mathbf{C}_{rf}(\mathcal{E}_0)$ *has a transition* $\{c, b\} \xrightarrow{\{b\}} \{b\}$, *the CS* $\mathbf{C}_{rf}(\mathcal{E}_1)$ *does not have a transition* $\{b\} \xrightarrow{\{b\}} \emptyset$. *Therefore* f *is not a morphism on the CSs.*

Like RBESs (see Example 4.4.6), RFESs allow causal cycles, which when combined with non-causal reversibility can lead to configurations where two events cause each other. Example 4.6.4 shows such an RFES, where the event c can be used to cause a, and then reverse, leaving room for a conflicting cause of a, b to be added to the configuration.

Example 4.6.4 (RFES)**.** *The RFES* $\mathcal{E} = (E, F, \prec, \sharp, \rhd)$ *where* $E = F = \{a, b, c\}$, $a \prec b$, $b \prec a$, $c \prec a$, *and* $b \sharp c$ *creates the CS seen in Figure 4.19. Note that* $\{a, b\}$ *is a reachable configuration, despite* a *and* b *causing each other.*

As we did for the RPESs, RAESs, and RBESs, we define RFES-morphisms in Definition 4.6.5. These are very similar to RPES-morphisms, with the causation relation replaced with the flow relation. Condition 4 of Definition 3.6.4 does not translate well to the reversible setting, since we do not have a notion of configurations. By removing it, we get products on all RFESs, not just well-behaved ones, but our morphisms no longer preserve transitions.

Definition 4.6.5 (RFES-morphism)**.** *Given RFESs* $\mathcal{E}_0 = (E_0, F_0, \prec_0, \sharp_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \prec_1, \sharp_1, \rhd_1)$, *an RFES-morphism from* $\mathcal{E}_0$ *to* $\mathcal{E}_1$ *is a partial function* $f : E_0 \to E_1$ *such that* $f(F_0) \subseteq F_1$ *and for* $e, e' \in E_0$*:*

1. *If* $f(e) \sharp_1 f(e')$ *then* $e \sharp_0 e'$;
2. *If* $f(e) = f(e')$ *and* $e \neq e'$ *then* $e \sharp_0 e'$;
3. *If* $f(e) \prec_1 f(e')^*$ *then* $e \prec_0 e'^*$;
4. *If* $f(e) \rhd_1 \underline{f(e')}$ *then* $e \rhd_0 \underline{e'}$.

Proposition 4.6.6. ***RFES*** *consisting of RFESs and RFES-morphisms is a category.*

Proof. Partial functions are obviously associative, and $f(e) = e$ functions as a an identity arrow, and the morphisms are composable. □

We also define a product of RFESs in Definition 4.6.7. Thanks to not having a condition on morphisms preserving configurations, the product is not restricted to well-behaved RFESs.

Definition 4.6.7 (Product of RFESs)**.** *Given RFESs* $\mathcal{E}_0 = (E_0, F_0, \prec_0, \sharp_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \prec_1, \sharp_1, \rhd_1)$, *we can construct* $\mathcal{E}_0 \times \mathcal{E}_1 = (E, F, \prec, \sharp, \rhd)$ *with projections* π_0 *and* π_1 *where:*

1. $E = E_0 \times_* E_1 = \{(e, *) \mid e \in E_0\} \cup \{(*, e) \mid e \in E_1\} \cup \{(e, e') \mid e \in E_0 \text{ and } e' \in E_1\}$;
2. $F = F_0 \times_* F_1 = \{(e, *) \mid e \in F_0\} \cup \{(*, e) \mid e \in F_1\} \cup \{(e, e') \mid e \in F_0 \text{ and } e' \in F_1\}$;

3. *for* $(e_0, e_1) \in E$, $\pi_i((e_0, e_1)) = e_i$;

4. *for any* $e \in E$, $e' \in E \cup \underline{F}$, $e \prec e'^*$ *iff there exists* $i \in \{0,1\}$ *such that* $\pi_i(e) \prec_i \pi_i(e')^*$;

5. *for any* $e, e' \in E$, $e \sharp e'$ *iff there exists* $i \in \{0,1\}$ *such that* $\pi_i(e) \sharp_i \pi_i(e')$, *or* $\pi_i(e) = \pi_i(e') \neq \perp$ *and* $\pi_{1-i}(e) \neq \pi_{1-i}(e')$;

6. *for any* $e \in E$, $e' \in F$, $e \rhd \underline{e'}$ *iff there exists* $i \in \{0,1\}$ *such that* $\pi_i(e) \lhd_i \underline{\pi_i(e')}$.

Proposition 4.6.8. *If* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *are RFESs, then* $\mathcal{E}_0 \times \mathcal{E}_1$ *is their product.*

Proof. Similar to the product of FESs in [30] □

We also construct a coproduct of RFESs.

Definition 4.6.9 (RFES coproduct)**.** *Given RFESs* $\mathcal{E}_0 = (E_0, F_0, \prec_0, \sharp_0, \rhd_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \prec_1, \sharp_1, \rhd_1)$, *we can construct* $\mathcal{E}_0 + \mathcal{E}_1 = (E, F, \prec, \sharp, \rhd)$ *with projections* i_0 *and* i_1 *where:*

- $E = \{(0, e) \mid e \in E_0\} \cup \{(1, e) \mid e \in E_1\}$;
- $F = \{(0, e) \mid e \in F_0\} \cup \{(1, e) \mid e \in F_1\}$;
- *for* $e \in E_j$, $i_j(e) = (j, e)$ *for* $j \in \{0,1\}$;
- $(j, e) \prec (j', e')^*$ *iff* $j = j'$ *and* $e \prec_j e'^*$;
- $(j, e) \sharp (j', e')$ *iff* $j \neq j'$ *or* $e \sharp_j e'$;
- $(j, e) \rhd \underline{(j', e')}$ *iff* $j \neq j'$ *or* $e \rhd_j \underline{e'}$.

Proposition 4.6.10. *If* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *are RFESs, then* $\mathcal{E}_0 \times \mathcal{E}_1$ *is their coproduct.*

Proof. Similar to that of RPES coproduct (Proposition 4.2.9). □

We define a functor from **FES** to **RFES**.

Definition 4.6.11 (**FES** to **RFES**)**.** *The functor* P_f : ***FES*** → ***RFES*** *is defined as:*

1. $P_f((E, \prec, \sharp)) = (E, \emptyset, \prec, \sharp, \emptyset \times \emptyset)$;
2. $P_f(f) = f$.

Proposition 4.6.12. P_f *is a functor from* ***FES*** *to* ***RFES****.*

Proof. If $\mathcal{E}$ is an FES then it is straightforward to check that $P_f(\mathcal{E})$ is an RFES, and if $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an FES morphism then $f : P_f(\mathcal{E}_0) \to P_f(\mathcal{E}_1)$ is an RFES morphism.

Having proved that P_f is a mapping from **FES** to **RFES**, the rest of the proof is similar to previous functors. □

Because of condition 4 of Definition 3.6.4, we cannot easily define a mapping from RFES morphisms to FES morphisms. If we want such a mapping, we will need to add a condition 5 to Definition 4.6.5 stating that: "If $X \subseteq E_0$ is a finite configuration in the FES $(E_0, \prec_0, \sharp_0)$ then $f(X)$ is a configuration in the FES $(E_1, \prec_1, \sharp_1)$." However, not only would this restrict our product to well-behaved RFESs, but while the condition in Definition 3.6.4 applies to all configurations of the FES, this condition only considers the forwards-reachable configurations of the RFES. We therefore do not define a functor from **RFES** to **FES**.

We define a functor from **RPES** to **RFES** in Definition 4.6.13.

Definition 4.6.13 (**RPES** to **RFES**)**.** *The functor* F_{pr} : ***RPES*** → ***RFES*** *is defined as:*

1. $F_{pr}((E, F, <, \sharp, \prec, \rhd)) = (E, F, \prec', \sharp, \rhd)$, *where* $\prec' = < \cup \prec$;
2. $F_{pr}(f) = f$.

Proposition 4.6.14. *F_{pr} is a functor from **RPES** to **RFES**.*

Proof. If $\mathcal{E}$ is an RPES then it is straightforward to check that $F_{pr}(\mathcal{E})$ is an RFES, and if $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an RPES morphism then $f : F_{pr}(\mathcal{E}_0) \to F_{pr}(\mathcal{E}_1)$ is an RFES morphism.

Having proved that F_{pr} is a mapping from **RPES** to **RFES**, the rest of the proof is similar to previous functors. □

We also define a mapping from RBES to RFES in Definition 4.6.15. As with the forward-only flow and bundle event structures, mapping the other way is not as simple. The problem outlined in Example 3.6.12 is also still an issue in the reversible setting, hence we do not get a functor.

Definition 4.6.15 (RBES to RFES)**.** *We define $F_r((E, F, \mapsto, \sharp, \rhd)) = (E \cup E', F, \prec, \sharp \cup \sharp', \rhd \cup \rhd')$, where:*

1. *if $e' \in X$ and $X \mapsto e^*$ then $e' \prec e^*$;*
2. *to each bundle $X \mapsto e^*$ and $X \neq \emptyset$, corresponds a unique $e' \in E'$ such that $e' \prec e^*$, $e' \sharp' e'$, and for all $e'' \in X$, $e'' \sharp' e'$;*
3. *$e \sharp' e'$ if $\emptyset \mapsto e$ and $e = e'$;*
4. *$e \rhd' \underline{e}$ if $\emptyset \mapsto \underline{e}$.*

As with previous reversible event structures, we define a cause-respecting subcategory of **RFES**, **cr-RFES**, consisting of the cause-respecting RFESs defined in Definition 4.6.16, and the RFES-morphisms between them.

Definition 4.6.16 (cause-respecting RFES)**.** *A RFES $\mathcal{E} = (E, F, \prec, \sharp, \rhd)$ is a* cause-respecting RFES *(cr-RFES) if for any $e, e' \in E$ if $e \prec e'$ and $e \in F$, then $e' \rhd \underline{e}$.*

Defining a causal RFES comes with complications. In some cases, such as Example 4.6.17, an event may flow to another, but never actually wind up causing it. This will cause confusion about whether that event should be prevented from being undone when the event it flows to or not. Of course if the RFES is causal, this is only a problem in unreachable transitions, as no reachable transition can contain both events.

Example 4.6.17. *Consider the FES $\mathcal{E} = (E, \prec, \sharp)$ where $E = F = \{a, b, c, d\}$, $a \prec d$, $b \prec d$, $c \prec d$, and $a \sharp a \sharp b \sharp c$. While c is formally a possible cause of d, because a will never happen, d cannot happen without b, which excludes c. This means that if we map $\mathcal{E}$ to a CS, these will no longer consider c a cause of d, and will therefore not want d to prevent the undoing of c, as one would expect.*

Definition 4.6.18 (Causal RFES)**.** *A RFES $\mathcal{E} = (E, F, \prec, \sharp, \rhd)$ is a* causal RFES *(CRFES) if for any $e, e' \in E$, (1) if $e' \rhd \underline{e}$ then either $e' \sharp e$ or $e \prec e'$, (2) if $e \prec e'$ and $e \in F$ then $e' \rhd \underline{e}$, and (3) if $e \prec \underline{e'}$ then $e = e'$ or $e \sharp e'$.*

Proposition 4.6.19. *Given a CRFES, $\mathcal{E} = (E, F, \prec, \sharp, \rhd)$ and corresponding CS $\boldsymbol{C}_{rf}(\mathcal{E}) = (E, F, \mathsf{C}, \to)$, any reachable $C \in \mathsf{C}$ is forwards-reachable.*

Proof. There exists a trace $\emptyset \xrightarrow{\{e_0^*\}} C_0 \xrightarrow{\{e_1^*\}} C_1 \ldots \xrightarrow{\{e_n^*\}} C_n$ where $C_n = C$. Clearly $e_0^* = e_0$, and C_0 is forwards-reachable, and we will show that if C_j is forwards-reachable for $0 \leq j \leq i$, then C_{i+1} is forwards-reachable.

If $e_{i+1}^* = e_{i+1}$, then clearly C_{i+1} is forwards-reachable.

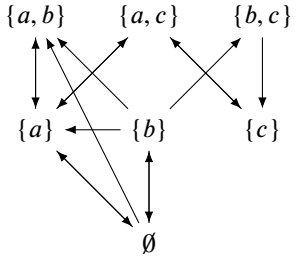
If $e_{i+1}^* = \underline{e_i}$, then $C_{i+1} = C_{i-1}$, which is forwards-reachable.

If $e_{i+1}^* = \underline{e_j}$ for some $0 \leq j < i$, then for all $0 \leq j' \leq i$, since $e_{j'} \not\rhd \underline{e_j}$, we know $e_j \not\prec e'_j$. This means obviously $\emptyset \xrightarrow{\{e_0^*\}} C_0 \xrightarrow{\{e_1^*\}} C_1 \ldots \xrightarrow{\{e_{j-1}^*\}} C_{j-1} \xrightarrow{\{e_{j+1}^*\}} C_{j+1} \setminus \{e_j\} \ldots \xrightarrow{\{e_i^*\}} C_{i+1}$. □

We have now finished defining our category of flow event structures. Due to not having an easy way to define (reachable) configurations in the reversible setting, we have been unable to translate the definition of FES-morphisms (Definiton 3.6.4) to the reversible setting, and so we have wound up with a definition that, while it gives us products for the whole category, is not preserved by any of our mappings.

$\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$ where
$E = F = \{a, b, c\}$
$\mathsf{Con} = \{\emptyset, \{a\}, \{b\}, \{a, b\}, \{a, c\}, \{b, c\}\}$
$\emptyset \otimes \emptyset \vdash a$, $\emptyset \otimes \{c\} \vdash b$,
$\{a\} \otimes \{b\} \vdash c$, $\{b\} \otimes \{a\} \vdash c$
$\{b\} \otimes \emptyset \vdash \underline{b}$, $\{a\} \otimes \{b\} \vdash \underline{a}$,
$\{a, c\} \otimes \emptyset \vdash \underline{c}$

(a) RES $\mathcal{E}$



(b) The CS derived from $\mathcal{E}$

Figure 4.20: Example of an RES and the corresponding CS.

4.7 Reversible General Event Structures

The last kind of event structure to which we add reversibility is the general event structure. The *reversible (general) event structure* (Definition 4.7.1) differs from the general event structure, not only by allowing the reversal of events, but also by including a preventing set in the enabling relation, so that $X \otimes Y \vdash e$ means e is enabled in configurations that include all the events of X but none of the events of Y. Example 4.7.2 shows how RESs can be used to model aspects of the reversible π-calculus that RPESs and RAESs cannot. We show that, unlike in the forward-only setting, most RAESs can be modelled by RESs by using prevention (Theorem 4.7.23), though the fact that we can only describe consistency of finite sets of events is still an issue. We also formulate a subcategory of CSs, called finitely enabled CSs (Definition 4.7.11), and create functors between these and RESs, called $\boldsymbol{C_r}$ (Definition 4.7.5) and $\boldsymbol{R}$ (Definition 4.7.14), which form an adjunction (Theorem 4.7.17) and are inverses on a subset of finitely enabled CSs (Theorem 4.7.16).

Definition 4.7.1 (RES [84]). *A* reversible event structure *(RES) is a triple* $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$ *where* E *is the set of events,* $F \subseteq E$ *is the set of reversible events,* $\mathsf{Con} \subseteq_{\mathrm{fin}} 2^E$ *is the consistency relation, which is downwards-closed,* $\vdash \subseteq \mathsf{Con} \times 2^E \times (E \cup \underline{F})$ *is the enabling relation, and*

1. *if* $X \otimes Y \vdash e^*$ *then* $(X \cup \{e\}) \cap Y = \emptyset$*;*
2. *if* $X \otimes Y \vdash \underline{e}$ *then* $e \in X$*;*
3. *(weakening) if* $X \otimes Y \vdash e^*$, $X \subseteq X' \in \mathsf{Con}$, *and* $X' \cap Y = \emptyset$ *then* $X' \otimes Y \vdash e^*$.

Note that in examples we will leave out enablings which are implied by the weakening condition. See Figure 4.20 for an example of an RES and the corresponding CS.

Example 4.7.2. *We intend to model the reversible* π*-calculus process* $(\nu n)(\overline{a}\langle n\rangle \mid \overline{b}\langle n\rangle \mid c(x))$*. This can be achieved with an RES with a set of events* $\{\overline{a}\langle n\rangle, \overline{b}\langle n\rangle\} \cup \{c(x) \mid x \text{ is a name}\}$ *with the consistent sets being any set not containing more that one event from* $\{c(x) \mid x \text{ is a name}\}$*. The enablings are as follows:*

$\emptyset \otimes \emptyset \vdash \overline{a}\langle n\rangle$ $\quad$ $\emptyset \otimes \emptyset \vdash \overline{b}\langle n\rangle$
$\emptyset \otimes \emptyset \vdash c(x)$ *for any name* $x \neq n$
$\{\overline{a}\langle n\rangle\} \otimes \{c(x) \mid x \neq n\} \vdash c(n)$ $\quad$ $\{\overline{b}\langle n\rangle\} \otimes \{c(x) \mid x \neq n\} \vdash c(n)$
$\{\overline{a}\langle n\rangle, \overline{b}\langle n\rangle, c(n)\} \otimes \emptyset \vdash \underline{\overline{b}\langle n\rangle}$ $\quad$ $\{\overline{b}\langle n\rangle, \overline{a}\langle n\rangle, c(n)\} \otimes \emptyset \vdash \underline{\overline{a}\langle n\rangle}$
$\{\overline{b}\langle n\rangle\} \otimes \{c(n)\} \vdash \underline{\overline{b}\langle n\rangle}$ $\quad$ $\{\overline{a}\langle n\rangle\} \otimes \{c(n)\} \vdash \underline{\overline{a}\langle n\rangle}$
$\{c(x)\} \otimes \emptyset \vdash \underline{c(x)}$ *for any name* x

This RES cannot be represented by an RPES or an RAES, as they cannot express that $\overline{b}\langle n\rangle$ *can reverse if* $\overline{a}\langle n\rangle$ *and* $c(n)$ *are in the configuration, but not if* $c(n)$ *is present without* $\overline{a}\langle n\rangle$*.*

We next define a notion of morphism for RESs in Definition 4.7.3. With the exception of the requirements regarding preventing sets, it is identical to the definition of an ES morphism. We treat the preventing set similarly to (asymmetric) conflict in the previous morphisms, requiring the events of Y_1 to be either synchronised with an event from Y_0 or no event at all. It is worth noting however that, unlike the

asymmetric conflict in other event structures, the preventing set is attached to a particular enabling and can be different for different enablings of the same event.

The reason we do not say that if $f(e) \neq \bot$ and $X \otimes Y \vdash_0 e^*$ then $f(X) \otimes f(Y) \vdash_1 f(e)^*$ is that we want our definition of an RES morphism to correspond with that of an RAES morphism, and we will use $\{e' \mid e' \rhd e\}$ to define our preventing sets when translating. Since RAES morphisms allow $e_1 \rhd f(e)$ if no e_0 exists such that $f(e_0) = e_1$, we have to express the same in our definition of RES morphisms.

Definition 4.7.3 (RES morphism)**.** *Let $\mathcal{E}_0 = (E_0, F_0, \mathsf{Con}_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mathsf{Con}_1, \vdash_1)$ be RESs. An RES morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$ is a partial function $f : E_0 \to E_1$ such that:*

1. *for all $e^* \in E_0 \cup \underline{F_0}$, if $f(e) \neq \bot$ and $X_0 \otimes Y_0 \vdash_0 e^*$ then there exists a $Y_1 \subseteq E_1$ such that for all $e_0 \in E_0$, if $f(e_0) \in Y_1$ then $\underline{e_0} \in Y_0$ and $f(X_0) \otimes Y_1 \vdash_1 f(e)^*$;*
2. *for any $X_0 \in \mathsf{Con}_0$, $f(X_0) \in \mathsf{Con}_1$;*
3. *for all $e, e' \in E_0$, if $f(e) = f(e') \neq \bot$ and $e \neq e'$ then no $X \in \mathsf{Con}_0$ exists such that $e, e' \in X$.*

Proposition 4.7.4. ***RES**, where RESs are objects and RES morphisms are arrows, is a category.*

Proof. This proof is similar to previous categories. See Appendix B.20 for details. □

We define a functor $\boldsymbol{C_r}$ from **RES** to **CS** (Definition 4.7.5). We remark that [84] described single event transitions of RESs and [90] discussed extending this to multiple events. We define configurations according with the first condition of Definition 3.7.2, disregarding the second condition, as we do not want to restrict ourselves to forwards-reachable configurations. For the transition, condition i. follows from the definition of a CS. The other two conditions require any event being added or removed in the transition to be enabled at any intermediate configuration during the transition, both when all the events being removed have been removed but nothing has been added, and also when all the events have been added, but none have been removed. They also require that the same enabling can be used in all intermediate configurations.

Definition 4.7.5 (From **RES** to **CS**)**.** *The functor $\boldsymbol{C_r} : \boldsymbol{RES} \to \boldsymbol{CS}$ is defined as*

1. *$\boldsymbol{C_r}((E, F, \mathsf{Con}, \vdash)) = (E, F, \mathsf{C}, \to)$, where*
   (a) *$X \in \mathsf{C}$ if for all $X' \subseteq_{\text{fin}} X$, $X' \in \mathsf{Con}$;*
   (b) *for $X, Y \in \mathsf{C}$, $X \xrightarrow{A \cup \underline{B}} Y$ if:*
      i. *$Y = (X \setminus B) \cup A$, $A \cap X = \emptyset$, $B \subseteq X$, and $X \cup A \in \mathsf{C}$;*
      ii. *for all $e \in A$, $X' \otimes Z \vdash e$ for some X', Z such that $X' \subseteq_{\text{fin}} X \setminus B$ and $Z \cap (X \cup A) = \emptyset$;*
      iii. *for all $e \in B$, $X' \otimes Z \vdash \underline{e}$ for some X', Z such that $X' \subseteq_{\text{fin}} X \setminus (B \setminus \{e\})$ and $Z \cap (X \cup A) = \emptyset$.*
2. *$\boldsymbol{C_r}(f) = f$.*

We construct a coproduct of two RESs in Definition 4.7.6. Although this definition is a little different from those of RPES and RAES coproducts, the principle is very much the same. Similarly to having every event from E_j prevent every event from E_{1-j} in the RAES coproduct, we here put every event from E_j in the preventing set of every enabling of an event from E_{1-j}.

Definition 4.7.6 (RES coproduct)**.** *Given RESs $\mathcal{E}_0 = (E_0, \mathsf{Con}_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, \mathsf{Con}_1, \vdash_1)$, we can construct $\mathcal{E}_0 + \mathcal{E}_1 = (E, \mathsf{Con}, \vdash)$ with injections i_0 and i_1 where:*

1. *$E = \{(0, e) \mid e \in E_0\} \cup \{(1, e) \mid e \in E_1\}$;*
2. *for $e \in E_j$ and $j \in \{0, 1\}$ we have $i_j(e) = (j, e)$;*
3. *$X \in \mathsf{Con}$ iff $\exists X_0 \in \mathsf{Con}_0 . i_0(X_0) = X$ or $\exists X_1 \in \mathsf{Con}_1 . i_1(X_1) = X$;*
4. *$X \otimes Y \vdash (j, e)^*$ iff $\exists X_j, Y_j \in E_j$ such that $X_j \otimes Y_j \vdash e^*$, $i_j(X_j) = X$, and $Y = i_j(Y_j) \cup (E \setminus i_j(E_j))$.*

Proposition 4.7.7. *Given RESs $\mathcal{E}_0$ and $\mathcal{E}_1$, we have $\mathcal{E}_0 + \mathcal{E}_1$ is their coproduct in **RES**.*

Proof. This proof is similar to previous coproducts. See Appendix B.21 for details. □

We also construct a product of RESs in Definition 4.7.8 based on the ES product described in [95]. This notion of product translates better to the reversible setting than ones used for PESs. In addition to adding reverse enablings, which are treated much the same as forward enablings, our product must also deal with preventing sets, which are not present in the product of ESs. Just as in the product of CSs, we can consider the product of RES a synchronisation of two RESs. Each enabling of an event, $X \otimes Y \vdash (e_0, *)$, must correspond to an enabling of the original event $X_0 \otimes Y_0 \vdash_0 e_0$, such that $\pi_0(X) = X_0$, but Y includes every e' such that $\pi_0(e') \in Y_0$. This means that we generate an enabling for each X for which this is true, all using the same Y, so $(e_0, *)$ is enabled as long as each e'_0 in X_0 has been involved in some synchronisation, but none of the e''_0 in Y have. Enablings of $(*, e_1)$ are created in a similar way, and enablings of (e_0, e_1) require this to hold for both. When creating enablings for (e_0, e_1), the weakening condition in Definition 4.7.1 ensures that we also get enablings where an event from X_0 synchronises with an event from $E_1 \setminus (X_1 \cup Y_1)$ or vice versa.

Definition 4.7.8 (partially synchronous RES product)**.** *Given two RESs $\mathcal{E}_0 = (E_0, \mathsf{Con}_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, \mathsf{Con}_1, \vdash_1)$, we can construct $\mathcal{E}_0 \times \mathcal{E}_1 = (E, \mathsf{Con}, \vdash)$ with projections π_0 and π_1 where:*

1. $E = E_0 \times_* E_1 = \{(e, *) \mid e \in E_0\} \cup \{(*, e) \mid e \in E_1\} \cup \{(e, e') \mid e \in E_0 \text{ and } e' \in E_1\}$;
2. *for $(e_0, e_1) \in E$ and $j \in \{0, 1\}$, $\pi_j((e_0, e_1)) = e_j$;*
3. $X \in \mathsf{Con}$ *if*
   (a) $\pi_0(X) \in \mathsf{Con}_0$;
   (b) $\pi_1(X) \in \mathsf{Con}_1$;
   (c) *for all $e, e' \in X$, if $\pi_0(e) = \pi_0(e') \neq *$ or $\pi_1(e) = \pi_1(e') \neq *$ then $e = e'$.*
4. $X \otimes Y \vdash e^*$ *if* $X \in \mathsf{Con}$ *and*
   (a) *if $\pi_0(e) \neq *$ then there exists $Y_0 \subseteq E_0$ such that $\pi_0(X) \otimes Y_0 \vdash \pi_0(e)^*$ otherwise $Y_0 = \emptyset$;*
   (b) *if $\pi_1(e) \neq *$ then there exists $Y_1 \subseteq E_1$ such that $\pi_1(X) \otimes Y_1 \vdash \pi_1(e)^*$ otherwise $Y_1 = \emptyset$;*
   (c) $Y = \{e \mid \pi_0(e) \in Y_0 \text{ or } \pi_1(e) \in Y_1\}$;
   (d) *if $e^* = \underline{e}$ then $e \in X$.*

Proposition 4.7.9. *Given RESs $\mathcal{E}_0$ and $\mathcal{E}_1$, we have $\mathcal{E}_0 \times \mathcal{E}_1$ is a product in the category **RES**.*

Example 4.7.10 shows the product of two RESs, represented by CSs in Figure 4.3.

Example 4.7.10 (RES product)**.** *Consider the RESs $\mathcal{E}_0 = (E_0, F_0, \mathsf{Con}_0, \vdash_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mathsf{Con}_1, \vdash_1)$, corresponding to the CSs in Figure 4.3, where*

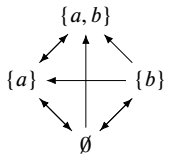
$$E_0 = \{a, b\} \qquad F_0 = E_0 \qquad \mathsf{Con}_0 = 2^{E_0}$$
$$\emptyset \otimes \emptyset \vdash_0 a \qquad \{a\} \otimes \emptyset \vdash_0 b \qquad \{a, b\} \otimes \emptyset \vdash_0 \underline{b} \qquad \{a\} \otimes \emptyset \vdash_0 \underline{a}$$

and

$$E_1 = \{c\} \qquad F_1 = E_1 \qquad \mathsf{Con}_1 = \{\emptyset, \{c\}\}$$
$$\emptyset \otimes \emptyset \vdash_1 c \qquad \{c\} \otimes \vdash_1 \underline{c}$$

The product $\mathcal{E}_0 \times \mathcal{E}_1$ is $(E, F, \mathsf{Con}, \vdash)$ where

$$E = F = \{(a, *), (b, *), (a, c), (b, c), (*, c)\}$$
$$\mathsf{Con} = \{\emptyset, \{(a, *)\}, \{(b, *)\}, \{(a, c)\}, \{(b, c)\}, \{(*, c)\}, \{(a, *), (b, *)\}$$
$$\{(a, *), (b, c)\}, \{(a, *), (*, c)\}, \{(a, c), (b, *)\}, \{(b, *), (*, c)\}, \{(a, *), (b, *), (*, c)\}\}$$
with forwards events enabled by $\emptyset \otimes \emptyset \vdash (a, *)$
$\{(a, *)\} \otimes \emptyset \vdash (b, *)$ $\{(a, c)\} \otimes \emptyset \vdash (b, *)$ $\emptyset \otimes \emptyset \vdash (a, c)$
$\{(a, *)\} \otimes \emptyset \vdash (b, c)$ $\{(a, c)\} \otimes \emptyset \vdash (b, c)$, $\emptyset \otimes \emptyset \vdash (*, c)$
and backwards events enabled by $\{(a, *)\} \otimes \emptyset \vdash \underline{(a, *)}$
$\{(b, *), (a, *)\} \otimes \emptyset \vdash \underline{(b, *)}$ $\{(b, *), (a, c)\} \otimes \emptyset \vdash \underline{(b, *)}$ $\{(a, c)\} \otimes \emptyset \vdash \underline{(a, c)}$
$\{(b, c), (a, *)\} \otimes \emptyset \vdash \underline{(b, c)}$ $\{(*, c)\} \otimes \emptyset \vdash \underline{(*, c)}$



(a) FCS $\mathcal{C}$

$\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$ where
$E = F = \{a, b\}$
$\mathsf{Con} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$
$\emptyset \otimes \emptyset \vdash a$, $\emptyset \otimes \emptyset \vdash b$,
$\{b\} \otimes \emptyset \vdash \underline{b}$, $\{a\} \otimes \{b\} \vdash \underline{a}$

(b) RES $\mathcal{E}$

Figure 4.21: An FCS and the corresponding RES such that $R(\mathcal{C}) = \mathcal{E}$ and $\mathcal{C}_r(\mathcal{E}) = \mathcal{C}$.

Applying $\mathcal{C}_r$ to an RES results in a *finitely enabled CS* (FCS). We call this 'finitely enabled' because the configurations are all determined by the set of finite configurations, and each transition from an infinite configuration X can be seen as enabled by a transition from a finite configuration X', such that every configuration between them can perform the same transition.

Definition 4.7.11. *Let $\mathcal{C} = (E, F, \mathsf{C}, \to)$ be a CS. We say that $\mathcal{C}$ is* finitely enabled *if*

1. C *is downwards-closed;*
2. $X \in \mathsf{C}$ *if and only if for all* $X' \subseteq_{\text{fin}} X$, $X' \in \mathsf{C}$;
3. *whenever* $X \xrightarrow{e^*}$, *there exists a finite configuration* $X' \subseteq_{\text{fin}} X$ *such that* $X'' \xrightarrow{e^*}$ *for all* X'' *such that* $X' \subseteq X'' \subseteq X$.

We call the category of FCSs and the CS morphisms between them **FCS**. Since this is a full subcategory of **CS**, Proposition 4.7.13 implies that we can define the product and coproduct in **FCS** in the same way as we do in **CS**.

Proposition 4.7.12. *$\mathcal{C}_r$ is a functor from **RES** to **FCS**.*

Proof. This proof is similar to previous functors. See Appendix B.23 for details. □

Proposition 4.7.13. *Given FCSs $\mathcal{C}_0$ and $\mathcal{C}_1$, both $\mathcal{C}_0 + \mathcal{C}_1$ and $\mathcal{C}_0 \times \mathcal{C}_1$ are FCSs.*

Proof. Follows from Definitions 4.7.11, 4.1.5, and 4.1.7. □

We describe a functor, R, from **FCS** to **RES** in Definition 4.7.14. An example of a mapping between an FCS and RES can be seen in Figure 4.21. The functor R works by taking each transition and creating the minimum enablings required for $\mathcal{C}_r$ to create that transition (that is, if $X \xrightarrow{A \cup \underline{B}} Y$ then we need $X \setminus B \otimes E \setminus (X \cup A) \vdash e$ for each $e \in A$, and similarly for $\underline{e} \in \underline{B}$) and adding the additional enablings required by the weakening condition of Definition 4.7.1 of RESs.

Definition 4.7.14 (From **FCS** to **RES**). *The functor $R : \boldsymbol{FCS} \to \boldsymbol{RES}$ is defined as:*

1. $R((E, F, \mathsf{C}, \to)) = (E, F, \mathsf{Con}, \vdash)$ *where $X \in \mathsf{Con}$ if $X \subseteq_{\text{fin}} X'$ for some $X' \in \mathsf{C}$ and $X \otimes Y \vdash e^*$ if:*
   (a) $X \cap Y = \emptyset$;
   (b) *if $e^* = \underline{e}$ then $e \in X$;*
   (c) *there exists an $X' \subseteq E$ such that $X' \xrightarrow{A \cup \underline{B}}$ for some A and B such that $e^* \in A \cup \underline{B}$, $X' \setminus B \subseteq X$, and $Y = E \setminus (X' \cup A)$.*
2. $R(f) = f$.

Proposition 4.7.15. *$R : \boldsymbol{FCS} \to \boldsymbol{RES}$ is a functor.*

Proof. This proof is similar to previous functors. See Appendix B.24 for details. □

As Theorem 4.7.16 states, $\boldsymbol{C}_r$ and $\boldsymbol{R}$ are inverses of each other for CSs for which the set of configurations is downwards-closed and derived from the set of finite configurations, and no transitions starting from infinite configurations, which are not part of a larger transition going through a finite configuration. In fact, if $\boldsymbol{C}$ is an FCS, then $\boldsymbol{C}_r(\boldsymbol{R}(\boldsymbol{C}))$ will always be such an FCS.

Theorem 4.7.16. *Given an FCS $\mathcal{C} = (E, F, \mathsf{C}, \to)$, we have $\boldsymbol{C}_r(\boldsymbol{R}(\mathcal{C})) = \mathcal{C}$ if whenever $X \xrightarrow{A \cup \underline{B}}$, there exists X' such that $X' \xrightarrow{A' \cup \underline{B'}}$, $X' \setminus B'$ is finite, $A \subseteq A'$, $X \cup A \subseteq X' \cup A'$, and $B \subseteq B'$.*

Proof. Let $\boldsymbol{R}(\mathcal{C}) = (E, F, \mathsf{Con}, \vdash)$ and $\boldsymbol{C}_r((E, F, \mathsf{Con}, \vdash)) = (E, F, \mathsf{C}', \to')$. Then Con is the finite subsets of the configurations of C, so since $X \in \mathsf{C}$ iff $\forall X' \subseteq_{\text{fin}} X . X' \in \mathsf{C}$, we get $\mathsf{C}' = \{C \mid \forall X \subseteq_{\text{fin}} C . X \in \mathsf{Con}\} = \mathsf{C}$. For any $X, Y, A \subseteq E$ and $B \subseteq F$, there exists a transition $X \xrightarrow{A \cup \underline{B}}{}' Y$ if and only if:

1. $Y = (X \setminus B) \cup A$, $A \cap X = \emptyset$, $B \subseteq X$, and $X \cup A \in \mathsf{C}'$;
2. for all e in A, $X' \otimes Z \vdash e$ for some X', Z such that $X' \subseteq_{\text{fin}} X \setminus B$ and $Z \cap (X \cup A) = \emptyset$;
3. for all $e \in B$, $X' \otimes Z \vdash \underline{e}$ for some X', Z such that $X' \subseteq_{\text{fin}} X \setminus (B \setminus \{e\})$ and $Z \cap (X \cup A) = \emptyset$.

Item 1 holds for transitions in any CS, including $\mathcal{C}$. For any $e^* \in A \cup \underline{B}$, if $X' \subseteq X$, $X \cap Z = \emptyset$, $X \in \mathsf{Con}$, and $X' \otimes Z \vdash e^*$, then there must exist $X'' \subseteq E$ such that $X'' \xrightarrow{A' \cup \underline{B'}}$, $e^* \in A' \cup \underline{B'}$, $X'' \setminus B' \subseteq X$, and $Z = E \setminus (X'' \cup A')$. Therefore $X \xrightarrow{A \cup \underline{B}}$.

Suppose $X \xrightarrow{A \cup \underline{B}} Y$. Then $X \setminus B \xrightarrow{A} Y$.

If $X \setminus B$ is finite, and therefore in Con, we have $X \setminus B \otimes E \setminus (A \cup X) \vdash e$ for all $e \in A$, and for all $e \in B$ we have $X \setminus (B \setminus \{e\}) \xrightarrow{A \cup \{e\}} Y$, similarly giving us $X \setminus (B \setminus \{e\}) \otimes E \setminus (A \cup X) \vdash \underline{e}$. Therefore $X \xrightarrow{A \cup \underline{B}}{}' Y$.

If $X \setminus B$ is infinite, then there exists X', A', B' such that $X' \xrightarrow{A \cup \underline{B}}$, $X \setminus B$ is finite, $A \subseteq A'$, $X \cup A \subseteq X' \cup A'$, and $B \subseteq B'$. This implies that for all $e \in A \subseteq A'$, $X' \setminus B' \otimes E \setminus (A' \cup X') \vdash e$ and for all $e \in B \subseteq B'$, $X' \setminus (B' \setminus \{e\}) \otimes E \setminus (A' \cup X') \vdash \underline{e}$. Therefore $X \xrightarrow{A \cup \underline{B}}{}' Y$.

□

We have now shown that $\boldsymbol{C}_r$ and $\boldsymbol{R}$ are inverse of each other for a broad subcategory of CSs, which includes all CSs generated by the previously described categories of event structures. However, we would like to show that even when we do not get as exact an inverse, we still have some correspondence.

We therefore now show that $\boldsymbol{R}$ and $\boldsymbol{C}_r$ form an adjunction between **RES** and **FCS**.

Theorem 4.7.17. $\boldsymbol{R} \dashv \boldsymbol{C}_r$.

Proof. This proof is similar in structure to previous proofs of adjunctions such as Proposition 3.3.11. See Appendix B.25 for full details. □

We also show in Proposition 4.7.18 that $\boldsymbol{C}_r$ preserves both coproducts and products up to isomorphism, the latter being implied by Theorem 4.7.17.

Proposition 4.7.18. *Given RESs $\mathcal{E}_0$ and $\mathcal{E}_1$, $\boldsymbol{C}_r(\mathcal{E}_0) + \boldsymbol{C}_r(\mathcal{E}_1) = \boldsymbol{C}_r(\mathcal{E}_0 + \mathcal{E}_1)$ and $\boldsymbol{C}_r(\mathcal{E}_0) \times \boldsymbol{C}_r(\mathcal{E}_1) \cong \boldsymbol{C}_r(\mathcal{E}_0 \times \mathcal{E}_1)$.*

Proof. We prove that the CSs are isomorphic in Appendix B.26. □

Just as we had functors between **PES** and **RPES** and **AES** and **RAES**, we have a functor from **ES** to **RES**, extending the mapping from [84], described in Definition 4.7.19.

Definition 4.7.19 (From **ES** to **RES**). *The functor $\boldsymbol{P}_r : \boldsymbol{ES} \to \boldsymbol{RES}$ is defined as*

1. $\boldsymbol{P}_r((E, \mathsf{Con}, \vdash)) = (E, \emptyset, \mathsf{Con}, \vdash')$ *where $X \otimes \emptyset \vdash' e$ if $X \vdash e$;*
2. $\boldsymbol{P}_r(f) = f$.

Proposition 4.7.20. *P_r is a functor from **ES** to **RES**.*

Proof. This proof is similar to previous functors. See Appendix B.27 for details. □

We also define a mapping from RAESs to RESs in Definition 4.7.21. The possibility of doing so was first discussed in [90], though a mapping was not defined. The only time we have $\boldsymbol{C}_{ra}(\mathcal{E}) = \boldsymbol{C}_r(\boldsymbol{P}_{ar}(\mathcal{E}))$ is when any set of events of the RAES which contains an infinite chain $e_0 \lhd e_1 \lhd e_2 \lhd \ldots$, also contains a finite $\lhd$-cycle. This is because, while RAES configurations require $\lhd$ to be well-founded, the set Con of the corresponding RES can only contain finite sets, such that all finite subsets of a configuration of the RES must be in Con. Obviously this does not allow us to define an RES with the same associated CS as an RAES which has an infinite chain $e_0 \lhd e_1 \lhd e_2 \lhd \ldots$ without any finite $\lhd$-cycles. We do however still get an RES, and the extra configurations will never be reachable in a finite number of transitions. It is worth noting that, while we cannot map all RAESs to a corresponding RES, thanks to the preventing set, we can get much closer than in the forward-only setting. Recall from Remark 3.7.6 that such a mapping did not exist from **AES** to **ES**, due to ESs not having a notion of prevention. Also note that, since in RAESs generated by RPESs $a \rhd b$ if and only if $b \rhd a$, we have $\boldsymbol{C}_{rp} = \boldsymbol{A}_r \circ \boldsymbol{P}_{ar} \circ \boldsymbol{C}_r$. The mapping $\boldsymbol{P}_{ar}$ from **RAES** to **RES** also acts as a functor.

Definition 4.7.21 (From **RAES** to **RES**). *The mapping $\boldsymbol{P}_{ar} : \boldsymbol{RAES} \to \boldsymbol{RES}$ is defined as*

1. $\boldsymbol{P}_{ar}((E, F, \prec, \lhd)) = (E, F, \mathsf{Con}, \vdash)$ *where:*

   $\mathsf{Con} = \{X \subseteq_{\text{fin}} E \mid \lhd \text{ is well-founded on } X\}$;

   $X \otimes Y \vdash e$ *if* $\{e' \mid e' \prec e\} \subseteq X \in \mathsf{Con}$, $Y = \{e' \mid e' \rhd e\}$, $X \cap Y = \emptyset$, *and* $e \in E$;

   $X \otimes Y \vdash \underline{e}$ *if* $\{e' \mid e' \prec \underline{e}\} \subseteq X \in \mathsf{Con}$, $Y = \{e' \mid e' \rhd \underline{e}\}$, $X \cap Y = \emptyset$, *and* $e \in F$.

2. $\boldsymbol{P}_{ar}(f) = f$.

Proposition 4.7.22. *$\boldsymbol{P}_{ar}$ is a functor from **RAES** to **RES**.*

Proof. This proof is similar to previous functors. See Appendix B.28 for details. □

Theorem 4.7.23. *Given an RAES $\mathcal{E} = (E, F, \prec, \lhd)$, such that for any $X \subseteq E$, if $\lhd$ is not well-founded on X then there exists a finite $\lhd$-cycle in X, we get $\boldsymbol{C}_r(\boldsymbol{P}_{ar}(\mathcal{E})) = \boldsymbol{C}_{ar}(\mathcal{E})$.*

Proof. We show that $\boldsymbol{C}_r(\boldsymbol{P}_{ar}(\mathcal{E}))$ has the same configurations and transitions as $\boldsymbol{C}_{ar}(\mathcal{E})$, using the condition on well-foundedness to ensure the configurations are the same. See Appendix B.29 for full details. □

What allows RESs to model RAESs even though forward-only general event structures cannot model forward-only AESs is the preventing set, which creates asymmetric conflict.

The functor $\boldsymbol{E}_b$ from **RBES** to **RES** creates an enabling of e for each conflict-free set which contains a member of all of e's bundles.

Definition 4.7.24 (**RBES** to **RES**). *The mapping $\boldsymbol{E}_b : \boldsymbol{RBES} \to \boldsymbol{RES}$ is defined as:*

1. $\boldsymbol{E}_b((E, F, \mapsto, \sharp, \rhd)) = (E, F, \mathsf{Con}, \vdash)$ *where:*

   (a) Con *is the set of finite conflict-free subsets of E;*

   (b) *For $e \in E$, $X \otimes \emptyset \vdash e$ if $X \cup \{e\} \in \mathsf{Con}$ and for all $X' \subseteq E$ such that $X' \mapsto e$, $X' \cap X \neq \emptyset$;*

   (c) *For $e \in F$, $X \otimes Y \vdash \underline{e}$ if (1) $X \in \mathsf{Con}$, (2) for all $X' \subseteq E$ such that $X' \mapsto \underline{e}$, $X' \cap X \neq \emptyset$, (3) $Y = \{e' \mid e' \rhd \underline{e}\}$, and (4) $Y \cap X = \emptyset$.*

2. $\boldsymbol{E}_b(f) = f$.

Proposition 4.7.25. *$\boldsymbol{E}_b$ is a functor from **RBES** to **RES**.*

Proof. Similar to previous functors. □

We also define a functor from **REBES** to **RES**. This causes the same problem as a functor from **RAES** to **RES**, and we may have some configurations of the generated SRES, which were not configurations of the original **REBES**.

Definition 4.7.26 (**REBES** to **RES**). *The mapping* $E_{er} : \boldsymbol{REBES} \to \boldsymbol{RES}$ *is defined as:*

1. $E_{er}((E, F, \mapsto, \rhd)) = (E, F, \mathsf{Con}, \vdash)$ *where:*

   (a) $\mathsf{Con} = \{X \subseteq_{\text{fin}} E \mid \lhd \text{ is well-founded on } X\}$;

   (b) *For* $e^* \in E \cup \underline{F}$, $X \otimes Y \vdash^* e^*$ *if* $X \in \mathsf{Con}$, *for all* $X' \subseteq E$ *such that* $X' \mapsto e^*$, $X' \cap X \neq \emptyset$, $Y = \{e' \mid e' \rhd e^*\}$, *and* $Y \cap X = \emptyset$.

2. $E_{er}(f) = f$.

Proposition 4.7.27. E_{er} *is a functor from* ***REBES*** *to* ***RES***.

Proof. Similar to previous functors. □

Our mapping from RFESs to RESs has the same problem as previous mappings to and from **RFES**, and is therefore not a functor.

Definition 4.7.28 (RFES to RES)**.**

$E_{fr}((E, F, \prec, \sharp, \rhd)) = (E, F, \mathsf{Con}, \vdash)$ *where:*

1. Con *is the set of finite conflict-free subsets of* E;

2. *for* $e \in E$, $X \otimes \emptyset \vdash e$ *if* $X \cup \{e\} \in \mathsf{Con}$ *and if* $e' \prec e$ *then* $e' \in X$ *or there exists* e'' *such that* $e'' \prec e$ *and* $e'' \sharp e'$;

3. *for* $e \in F$, $X \otimes Y \vdash \underline{e}$ *if* $X \in \mathsf{Con}$, *if* $e' \prec \underline{e}$ *then* $e' \in X$ *or there exists* e'' *such that* $e'' \prec \underline{e}$ *and* $e'' \sharp e'$, $Y = \{e' \mid e' \rhd \underline{e}\}$, *and* $Y \cap X = \emptyset$.

We have now got a categorical definition of our most expressive reversible event structures, **RES**. We have found that many of the categorical concepts of **ES** transfer well to the reversible setting despite the addition of asymmetric conflict.

4.8 Reversible Stable Event Structures

Similarly to the stable event structures (Definition 3.7.4), we define a subcategory of RESs called *reversible stable event structures* (RSES) in Definition 4.8.1, with the category **RSES** consisting of RSESs and the RES morphisms between them. In RSESs, the causation is stable, meaning we can tell in any given configurations, which events caused each other. Stability is particularly interesting in a causally reversible setting, since whether an event caused another is what determines whether it can reverse. RSESs and SESs are defined identically, with the exception that in an RSES the preventing sets are included as well, and treated in much the same way as the enabling sets. Figure 4.21b showed an RSES. Though we needed a non-stable RES to model the process described in Example 4.7.2, RSESs are still useful for other cases, as illustrated by Example 4.8.2. We also define a stable CS (SCS), and a new functor from SCSs to RSESs called $\boldsymbol{R'}$ (Definition 4.8.7). Functors $\boldsymbol{R'}$ and $\boldsymbol{C}_r$ act as inverses on SCSs (Proposition 4.8.15), and on RSESs in normal form (Definition 4.8.19).

Like in an SES, an event in a configuration of an RSES will have a single unique possible cause as long as the configuration has been reached by only going forwards. Also like the SES for the PES, mapping an RPES, RAES, or RFES into an RES always results in an RSES, as stated in Proposition 4.8.3. In addition, the subcategory is closed under RES coproduct and product. Since **RSES** consisting of RSESs and the RES morphisms between them is a full subcategory, we can then construct RSES products and coproducts in the same manner as we do RES products and coproducts.

Definition 4.8.1 (Reversible SES). *A reversible* stable *event structure (RSES) is an RES* $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$ *such that for all* $e^* \in E \cup \underline{F}$, *if* $X \otimes Y \vdash e^*$, $X' \otimes Y' \vdash e^*$, *and* $X \cup X' + e^* \in \mathsf{Con}$, *then* $X \cap X' \otimes Y \cap Y' \vdash e^*$.

Example 4.8.2. *Consider a situation where we want to model a computation $e_0, e_1, e_2, \ldots$ which can be forced to reverse if it encounters one of a number of different errors $\{\mathsf{error}_0, \mathsf{error}_1, \ldots, \mathsf{error}_n\}$. We do this with an RES with events $\{e_0, e_1, \ldots\} \cup \{\mathsf{error}_0, \mathsf{error}_1, \ldots, \mathsf{error}_n\}$, consistent sets $\mathsf{Con} = \{X \mid \nexists i, j.\{\mathsf{error}_i, \mathsf{error}_j\} \subseteq X\}$, and the enablings:*

$\{e_k \mid k < i\} \otimes \{\mathsf{error}_j \mid 0 \leq j \leq n\} \vdash e_i$ *for all* e_i,

$\emptyset \otimes \emptyset \vdash \mathsf{error}_j$ *and* $\{\mathsf{error}_j\} \otimes \{e_0\} \vdash \underline{\mathsf{error}_j}$ *for all* error_j,

and $\{\mathsf{error}_j\} \otimes \{e_k \mid k > i\} \vdash \underline{e_i}$ *for all combinations of* i *and* j.

This RES will be stable.

We show that the mappings from other categories of reversible event structures to **RES** create RSESs.

Proposition 4.8.3. *If $\mathcal{E} = (E, F, \prec, \lhd)$ is an RAES then $P_{ar}(\mathcal{E})$ is an RSES.*

Proof. Recall from Proposition 4.7.22, that $P_{ar}(\mathcal{E}) = (E, F, \mathsf{Con}, \vdash)$ is an RES.

We need to show that for all $e^* \in E \cup \underline{F}$, if $X \otimes Y \vdash e^*$, $X' \otimes Y' \vdash e^*$, and $X \cup X' + e^* \in \mathsf{Con}$ then $X \cap X' \otimes Y \cap Y' \vdash e^*$.

Suppose $e^* = e$. Then $\{e' \in E \mid e' \prec \underline{e}\} \subseteq X, X' \in \mathsf{Con}$ and $Y = Y' = \{e' \in E \mid e' \rhd \underline{e}\}$. Since $\lhd$ is well-founded on $X \cup X' + e^*$, X and X' are finite, and $e' \rhd e$ implies $e' \notin X \cup X'$, we know that $\lhd$ is well-formed on $X \cap X'$. Hence $\{e' \in E \mid e' \prec e\} \subseteq X \cap X' \in \mathsf{Con}$, implying that $X \cap X' \otimes Y \vdash e$ as required.

Suppose $e^* = \underline{e}$. Then $\{e' \in E \mid e' \prec \underline{e}\} \subseteq X, X' \in \mathsf{Con}$ and $Y = Y' = \{e' \in E \mid e' \rhd \underline{e}\}$ and the logic is similar to above. □

Proposition 4.8.4. *Given an RFES $\mathcal{E} = (E, F, \prec, \sharp, \rhd)$, we have that $E_{fr}((E, F, \prec, \sharp, \rhd))$ is a RSES.*

Proof. If $X \otimes Y \vdash e^*$, $X' \otimes Y' \vdash e^*$, and $X \cup X' + e^* \in \mathsf{Con}$ then, $Y = Y'$, $X \cup X'$ is conflict-free, meaning there must exist an $X'' \subseteq (X \cap X')$ such that for all $e' \in X''$, $e' \prec e^*$ and if $e'' \notin X''$ and $e'' \prec e^*$ then there exists $e''' \in X''$ such that $e'' \sharp e'''$. This means $X \cap X' \otimes Y \cap Y' \vdash e^*$. □

Proposition 4.8.5. *Given two RSESs $\mathcal{E}_0$ and $\mathcal{E}_1$, we have that $\mathcal{E}_0 + \mathcal{E}_1$ is an RSES.*

Proof. Let $\mathcal{E}_0 = (E_0, F_0, \mathsf{Con}_0, \vdash_0)$, $\mathcal{E}_1 = (E_1, F_1, \mathsf{Con}_1, \vdash_1)$, and $\mathcal{E}_0 + \mathcal{E}_1 = (E, F, \mathsf{Con}, \vdash)$. Then suppose $X \otimes Y \vdash e^*$, $X' \otimes Y' \vdash e^*$, and $X \cup X' + e^* \in \mathsf{Con}$. Then $e^* = (j, e_j)^*$ and $\exists X_j, Y_j \in E_j$ such that $X_j \otimes Y_j \vdash e^*$, $i_j(X_j) = X$, $Y = i_j(Y_j) \cup (E \setminus i_j(E_j))$ and $\exists X'_j, Y'_j \in E_j$ such that $X'_j \otimes Y'_j \vdash e^*$, $i_j(X'_j) = X'$, $Y' = i_j(Y'_j) \cup (E \setminus i_j(E_j))$. Since $X \cup X' + e^* \in \mathsf{Con}$ implies $X_j \cup X'_j + e^*_j \in \mathsf{Con}_0$, we get $X_j \cap X'_j \otimes Y_j \vdash e^*_j$. Therefore $X \cap X' \otimes Y \vdash e^*$. □

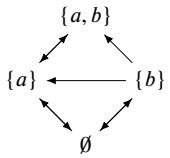
Proposition 4.8.6. *Given two RSESs $\mathcal{E}_0$ and $\mathcal{E}_1$, we have that $\mathcal{E}_0 \times \mathcal{E}_1$ is an RSES.*

Proof. We prove that $\mathcal{E}_0 \times \mathcal{E}_1$ fulfils the conditions of Definition 4.8.1 in Appendix B.30. □

Similarly, we also define a *stable configuration system* in Definition 4.8.7. Again we want events to have a unique cause in any configuration. Condition (a) states that if $X_1 \subseteq X_3$ and both X_1 and X_3 can do a transition, then so can every configuration in between them, since this means X_1 is unambiguously what enables this transition in X_3. Condition (b) is very similar to Definition 3.7.4, but referring to configurations rather than enabling sets. Condition (c) ensures that if $X_1 \cup X_2$ can do a transition, and X_1 and X_2 can each do the same transition individually, then that transition is caused by $X_1 \cap X_2$ in all those configurations. Finally, condition (d) states that if two transitions can both happen in any order between two states, then they are independent and can therefore happen simultaneously.

Definition 4.8.7 (Stable CS)**.** *A* stable *CS (SCS) is a CS $C = (E, F, \mathsf{C}, \rightarrow)$ such that:*

1. *C is downwards closed;*
2. *For $X_1, X_2, X_3 \in \mathsf{C}$, $A \subseteq E$, and $B \subseteq F$:*
   - *(a) if $X_1 \subseteq X_2 \subseteq X_3$, and there exist transitions $X_1 \xrightarrow{A \cup \underline{B}}$ and $X_3 \xrightarrow{A \cup \underline{B}}$, then $X_2 \xrightarrow{A \cup \underline{B}}$;*
   - *(b) if $((X_1 \cup X_2) \setminus B) \cup A \in \mathsf{C}$, and there exist transitions $X_1 \xrightarrow{A \cup \underline{B}}$ and $X_2 \xrightarrow{A \cup \underline{B}}$, then $X_1 \cap X_2 \xrightarrow{A \cup \underline{B}}$;*



(a) Non-stable CS $\mathcal{C}$

$\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$ where
$E = F = \{a, b\}$
$\mathsf{Con} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$
$\emptyset \otimes \emptyset \vdash a$,
$\{a\} \otimes \emptyset \vdash b$,
$\emptyset \otimes \{a\} \vdash b$,
$\{b\} \otimes \emptyset \vdash \underline{b}$, $\{a\} \otimes \{b\} \vdash \underline{a}$

(b) Non-stable RES $\mathcal{E}$

Figure 4.22: Example of a non-stable CS and the corresponding RES.

(c) if $X_1 \cap X_2 \subseteq X_3 \subseteq X_1 \cup X_2$, $(X_3 \cup A) \setminus B \in \mathsf{C}$, *and there exist transitions* $X_1 \xrightarrow{A \cup \underline{B}}$, $X_2 \xrightarrow{A \cup \underline{B}}$, *and* $X_1 \cap X_2 \xrightarrow{A \cup \underline{B}}$, *then* $X_3 \xrightarrow{A \cup \underline{B}}$;

(d) if $X_0, X_1, X_2, X_3 \in \mathsf{C}$, $A_0, A_1, B_0, B_1 \subseteq E$ *and there exist transitions* $X_0 \xrightarrow{A_0 \cup \underline{B_0}} X_1$, $X_0 \xrightarrow{A_1 \cup \underline{B_1}} X_2$, $X_1 \xrightarrow{A_1 \cup \underline{B_1}} X_3$, *and* $X_2 \xrightarrow{A_0 \cup \underline{B_0}} X_3$, *then* $X_0 \xrightarrow{A_0 \cup A_1 \cup \underline{B_0 \cup B_1}} X_3$.

Since the causation in RPESs, RAESs, RBESs, REBESs, and RFESs is stable, they all map to stable CSs.

Proposition 4.8.8. *Given an RAES* $\mathcal{E} = (E, F, \prec, \lhd)$, $\boldsymbol{C}_{ra}(\mathcal{E})$ *is an SCS.*

Proof. Follows from Definitions 4.3.1, 4.3.2, and 4.8.7. □

Proposition 4.8.9. *If* $\mathcal{E} = (E, F, \prec, \sharp, \rhd)$ *is an RFES, then* $\boldsymbol{C}_{fr}(\mathcal{E})$ *is a SCS.*

Proof. We show that $\boldsymbol{C}_{fr}(\mathcal{E})$ fulfils the conditions of Definition 4.8.7 in Appendix B.31. □

Proposition 4.8.10. *Given SCSs* $\mathcal{C}_0$ *and* $\mathcal{C}_1$, *both* $\mathcal{C}_0 + \mathcal{C}_1$ *and* $\mathcal{C}_0 \times \mathcal{C}_1$ *are SCSs.*

Proof. Follows from Definitions 4.1.5 and 4.8.7. □

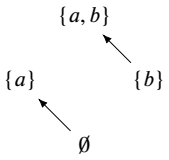
Figure 4.21a shows a stable CS. One way to make it non-stable would be to remove the transition from $\emptyset$ to $\{a, b\}$, since that would violate item 2d, giving us the non-stable CS from Example 4.1.2, recalled in Figure 4.22 with a corresponding RES added. This also makes the corresponding RES non-stable, as evidenced by the fact that $\{a\} \otimes \emptyset \vdash b$ and $\emptyset \otimes \{a\} \vdash b$, but not $\emptyset \otimes \emptyset \vdash b$.

We previously defined a functor $\boldsymbol{R}$ from **FCS** to **RES** (Definition 4.7.14), but some CSs can be represented by more than one RES. Consider, for example, the one seen in Figure 4.23a, where the two transitions could be caused by the same or different enablings. The functor presented in Definition 4.7.14 gives us the RES in Figure 4.23b. Note that while the FCS is stable, the RES generated by $\boldsymbol{R}$ is not. We would like a functor which maps SFCSs to RSESs, and we therefore define a new functor specifically for stable FCSs in Definition 4.8.11, which gives us the RSES in Figure 4.23c. Functor $\boldsymbol{R}$ will always create an enabling for each transition (with the required subenablings caused by the weakening condition of Definition 4.7.1), while $\boldsymbol{R}'$ finds a maximum X and a minimum X' such that for $X' \subseteq X'' \subseteq X$, if $X'' \in \mathsf{C}$ and $X'' + e^* \in \mathsf{C}$ then $X'' \xrightarrow{\{e^*\}}$, and generates one enabling (with required subenablings) for all those configurations. For this reason $\boldsymbol{R}'$ does not preserve morphisms on non-stable FCSs, and we therefore only use it on stable ones. However, $\boldsymbol{R}'$ also gives us the ability to model transitions from infinite sets which are not a part of a larger transition from a finite set as described in Theorem 4.7.16.

Definition 4.8.11 (From **SFCS** to **RSES**). *The functor* $\boldsymbol{R}' : \boldsymbol{SFCS} \to \boldsymbol{RSES}$ *is defined as:*

1. $\boldsymbol{R}'((E, F, \mathsf{C}, \to)) = (E, F, \mathsf{Con}, \vdash)$ *where* $\mathsf{Con} = \{X \mid X \subseteq_{\text{fin}} X' \in \mathsf{C}\}$ *and for all* X, X', e *such that* $X' \in \mathsf{C}$, $e \in E$, *and* $X \supseteq X'$ *if:*

   (a) if $X + e^* \in \mathsf{C}$ *then* $X \xrightarrow{\{e^*\}}$;



(a) CS C

$R(C) = (E, F, \mathsf{Con}, \vdash)$ where
$E = \{a, b\}$
$F = \emptyset$
$\mathsf{Con} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$
$\emptyset \otimes \{b\} \vdash a$
$\{b\} \otimes \emptyset \vdash a$

(b) RES $R(C)$

$R'(C) = (E, F, \mathsf{Con}, \vdash)$ where
$E = \{a, b\}$
$F = \emptyset$
$\mathsf{Con} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$
$\emptyset \otimes \emptyset \vdash a$

(c) RES $R'(C)$

Figure 4.23: Example of a CS and the two different corresponding RESs.

(b) if $X', X' + e^* \in \mathsf{C}$ *then* $X' \xrightarrow{\{e^*\}}$, *and whenever* $X' \subseteq X'' \subseteq X$ *and* $X'', X'' + e^* \in \mathsf{C}$, *there exists a transition* $X'' \xrightarrow{\{e^*\}}$;

(c) no $X'' \subsetneq X'$ *exists such that if* $X'', X'' + e^* \in \mathsf{C}$ *then* $X'' \xrightarrow{\{e^*\}}$, *and whenever* $X'' \subseteq X''' \subseteq X$ *and* $X''', X''' + e^* \in \mathsf{C}$ *there exists a transition* $X''', X''' + e^* \xrightarrow{\{e^*\}}$;

(d) no $X'' \supsetneq X$ *exists such that if* $X'', X'' + e^* \in \mathsf{C}$ *then* $X'' \xrightarrow{\{e^*\}}$, *and whenever* $X' \subseteq X''' \subseteq X''$ *and* $X''', X''' + e^* \in \mathsf{C}$ *there exists a transition* $X''' \xrightarrow{\{e^*\}}$;

(e) there exists $\mathbf{C} \subseteq \mathsf{C}$, *such that* $\bigcup \mathbf{C} = X$, *and for all* $X'' \in \mathbf{C}$, $X'' \xrightarrow{\{e^*\}}$ *and* $X' \subseteq X''$.

then we get the following enablings:

(a) if $e^* = e$, *then for all* $X'' \in \mathsf{Con}$ *such that* $X' \subseteq X'' \subseteq X \cup \{e\}$, $X'' \otimes E \setminus (X \cup \{e\}) \vdash e$;

(b) if $e^* = \underline{e}$, *then for all* $X'' \in \mathsf{Con}$ *such that* $X' \subseteq X'' \subseteq X$, $X'' \otimes E \setminus X \vdash \underline{e}$.

2. $R'(f) = f$.

Proposition 4.8.12. *R' is a functor from **SFCS** to **RSES**.*

Proof. We prove this in Appendix B.32. The main difficulty is proving that $R'(C)$ of an SFCS C is an RSES. □

Proposition 4.8.13. *If $\mathcal{E}$ is an RSES then $C_r(\mathcal{E})$ is an SFCS.*

Proof. We show in Appendix B.33 that $C_r(\mathcal{E})$ fulfils the conditions of an SFCS. □

Though R' and C_r are inverses on SFCSs, they are not inverses on all RSESs, only those whose enablings satisfy Definition 4.8.14 below. As Proposition 4.8.15 suggests, only one such RSES exists per SFCS, but there can be one or more non-stable RESs with *necessary* and *minimal* enablings which also generates the same SFCS, as Figure 4.23 shows.

An enabling is considered necessary if it is not implied by other enablings. If for example we have an enabling $\{a\} \otimes \emptyset \vdash b$ then also having $\{a\} \otimes \{c\} \vdash b$ is unnecessary, as there cannot exist any configurations in which $\{a\} \otimes \{c\} \vdash b$ allows us to do b, but $\{a\} \otimes \emptyset \vdash b$ does not. Additionally, we consider an enabling necessary if it is needed because of a necessary enabling and the weakening condition of Definition 4.7.1.

An enabling is considered minimal if every event which cannot potentially be in the same configuration as both the enabling set and the event being enabled is in the preventing set. For example $\emptyset \otimes \emptyset \vdash b$ is not minimal if there do not exist any configurations of which $\{a, b\}$ is a subset, as the enabling $\emptyset \otimes \{a\} \vdash b$ would have had the same effect with a larger preventing set.

Definition 4.8.14. *Given an RES, $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$, an enabling $X \otimes Y \vdash e^*$ is*

1. necessary *if there exists an $X' \subseteq X$ such that $X' \otimes Y \vdash e^*$ and there does not exist an enabling $X' \otimes Y' \vdash e^*$ for $Y' \subsetneq Y$;*

2. minimal *if for every e' such that $X \cup \{e, e'\} \notin \mathsf{C}$, $e' \in Y$.*

If all the enablings in $\mathcal{E}$ are necessary and minimal, we say that $\mathcal{E}$ is in normal form*, and we define the category **nfRES** (resp. **nfRSES**) of RESs (resp. RSESs) in normal form and the RES morphisms between them.*

Proposition 4.8.15.

1. *If $\mathcal{E}$ is an RSES in normal form, then $R'(C_r(\mathcal{E})) = \mathcal{E}$.*
2. *If C is an SFCS, then $C_r(R'(C)) = C$.*

Proof. We prove the two statements separately in Appendix B.34. □

Proposition 4.8.16. *If C is an SFCS, then $R'(C)$ is in normal form.*

Proof. Let $C = (E, F, \mathsf{C}, \to)$ be an SFCS and let $R'(C) = \mathcal{E} = (E, F, \mathsf{Con}, \vdash)$. Consider an enabling $X \otimes Y \vdash e^*$.

We first show that it is necessary. If $e^* = e$ then there exists an $X_1 \subseteq X$, such that whenever $X_1 \subseteq X_1' \subseteq E \setminus Y$ and $X_1', X_1' \cup \{e\} \in \mathsf{C}$ then $X_1' \xrightarrow{\{e\}}$, no $X_2 \subsetneq X_1$ exists such that whenever $X_2 \subseteq X_2' \subseteq E \setminus Y$ and $X_2', X_2' \cup \{e\} \in \mathsf{C}$ then $X_2' \xrightarrow{\{e\}}$, and no $X_3 \supsetneq E \setminus (Y)$ exists such that whenever $X_1 \subseteq X_3' \subseteq X_3$ and $X_3', X_3' \cup \{e\} \in \mathsf{C}$ then $X_3' \xrightarrow{\{e\}}$. Hence that $X_1 \in \mathsf{Con}$ because $X_1 \subseteq X \in \mathsf{Con}$ and there cannot exist an enabling $X_1 \otimes Y'$ for $Y' \subsetneq Y$.

If $e^* = \underline{e}$ the proof is similar.

The enabling $X \otimes Y \vdash e^*$ is also minimal, since C is downwards-closed and there exists $C \subseteq \mathsf{C}$, such that $\bigcup C = E \setminus Y$, and for all $X' \in C$, $X' \xrightarrow{\{e^*\}}$ and $X \subseteq X'$. □

Corollary 4.8.17. ***nfRSES*** *and* ***SFCS*** *are isomorphic.*

Proof. Implied by Propositions 4.8.15 and 4.8.16. □

This gives us a tight correspondence between **nfRSES** and **SFCS** similar to **PES** and **Dom** in the forward-only setting.

We showed in Propositions 4.8.5 and 4.8.6 that the coproduct and product of two stable RESs will always be stable. We now show that the coproduct and product of RESs in normal form will also be in normal form.

Proposition 4.8.18. *Given RESs $\mathcal{E}_0$ and $\mathcal{E}_1$ in normal form, both $\mathcal{E}_0 + \mathcal{E}_1$ and $\mathcal{E}_0 \times \mathcal{E}_1$ are in normal form.*

Proof. Follows from Definitions 4.7.6, 4.7.8, and 4.8.14. □

We define a way of mapping from RESs to normal form RESs in Definition 4.8.19, and in the following propositions show that the RES generated by this mapping will map to the same CS as the original. To get an RES in normal form, we first minimise the enablings by finding one not required by the weakening condition of Definition 4.7.1, $X' \otimes Y' \vdash e^*$, and then adding any event e', for which $X' \cup \{e, e'\} \notin \mathsf{Con}$ to Y', and considering this enabling as well as those implied by it and the weakening condition part of $\vdash_{\min}$. Out of these minimised enablings, we then take the necessary (Definition 4.8.14) ones, and those required by the necessary ones and the weakening condition.

Definition 4.8.19 (RES normaliser N)**.** *Given an RES $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$, we define the minimised enablings of $\mathcal{E}$, $\vdash_{\min}$ as $X \otimes Y \vdash_{\min} e^*$ if there exist $X', Y' \subseteq E$ such that:*

1. *$X \in \mathsf{Con}$ and $(X \cup \{e\}) \cap Y = \emptyset$;*
2. *$X' \otimes Y' \vdash e^*$;*
3. *$X' \subseteq X$;*
4. *no $X'' \subsetneq X'$ exists such that $X'' \otimes Y' \vdash e^*$;*
5. *$e' \in Y$ iff $e \in Y'$ or $X' \cup \{e, e'\} \notin \mathsf{Con}$.*

We say that $X' \otimes Y' \vdash e^*$ generated $X \otimes Y \vdash_{\min} e^*$.

We then take the necessary minimised enablings, $X \otimes Y \vdash_{\mathsf{nec}} e^*$*, such that*

1. $X \otimes Y \vdash_{\min} e^*$;
2. *there exists an* $X_{\min} \subseteq X$ *such that:*
   (a) $X_{\min} \otimes Y \vdash_{\min} e^*$;
   (b) *no* $X' \subseteq X_{\min}$ *and* $Y' \subseteq Y$ *exist such that* $X' \otimes Y' \vdash_{\min} e^*$ *and not both* $X' = X_{\min}$ *and* $Y' = Y$.

We define the functor $N : \boldsymbol{RES} \to \boldsymbol{nfRES}$ *as:*

1. $N((E, F, \mathsf{Con}, \vdash)) = (E, F, \mathsf{Con}, \vdash_{\mathsf{nec}})$;
2. $N(f) = f$.

We show that N maps from RESs to RESs in normal form, which can be represented by the same CS (Propositions 4.8.20 and 4.8.21).

Proposition 4.8.20. *N is a functor from $\boldsymbol{RES}$ to $\boldsymbol{nfRES}$.*

Proof. We prove this in Appendix B.35. The main challenge is proving that generates an nfRES. □

Proposition 4.8.21. *Given an RES* $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$*, we have that* $\boldsymbol{C}_r(\mathcal{E}) = \boldsymbol{C}_r(N(\mathcal{E}))$.

Proof. Suppose $\boldsymbol{C}_r(\mathcal{E}) = (E, F, \mathsf{C}, \to)$, $N(\mathcal{E}) = (E, F, \mathsf{Con}, \vdash')$, and $\boldsymbol{C}_r(N(\mathcal{E})) = (E, F, \mathsf{C}, \to')$. Then whenever $X \xrightarrow{A \cup \underline{B}} Y$, we have $Y = (X \setminus B) \cup A$, $A \cap X = \emptyset$, $B \subseteq X$, and $X \cup A \in \mathsf{C}$.

There exist X' and Z such that $X' \otimes Z \vdash e$, $X' \subseteq_{\text{fin}} X$, and $Z \cap (X \cup A) = \emptyset$. Therefore we get that $X' \otimes (Z \cup \{e' \mid X' \cup \{e, e'\} \notin \mathsf{Con}\}) \vdash_{\min} e$ and therefore $X' \otimes Z'' \vdash_{\mathsf{nec}} e$ for $Z'' \subseteq Z \cup \{e' \mid X' \cup \{e, e'\} \notin \mathsf{Con}\}$.

Since $X \cup A \in \mathsf{C}$, we have for all $X'' \subseteq_{\text{fin}} X \cup A$, $X'' \in \mathsf{Con}$, and therefore $(\{e' \mid X' \cup \{e, e'\} \notin \mathsf{Con}\}) \cap (X \cup A) = \emptyset$.

We use similar logic to show that for all $e \in B$, $X' \otimes Z \vdash' \underline{e}$ for some X' and Z such that $X' \subseteq_{\text{fin}} X \setminus (B \setminus \{e\})$ and $Z \cap (X \cup A) = \emptyset$.

If $X \xrightarrow{A \cup \underline{B}}{}' Y$ then $Y = (X \setminus B) \cup A$, $A \cap X = \emptyset$, $B \subseteq X$, and $X \cup A \in \mathsf{C}$.

There exist X' and Z such that $X' \otimes Z \vdash_{\mathsf{nec}} e$, $X' \subseteq_{\text{fin}} X$, and $Z \cap (X \cup A) = \emptyset$. Hence there exist $X'' \subseteq X'$ and $Z' \subseteq Z$ such that $X'' \otimes Z' \vdash e$.

We use similar logic to show that for all $e \in B$, $X' \otimes Z \vdash \underline{e}$ for some X' and Z such that $X' \subseteq_{\text{fin}} X \setminus (B \setminus \{e\})$ and $Z \cap (X \cup A) = \emptyset$.

□

This means N does not affect nfRSESs and on other RSESs, N corresponds to mapping to a CS and using $\boldsymbol{R}'$ to map back to an RSES.

Corollary 4.8.22. *If $\mathcal{E}$ is an RSES in normal form, then $N(\mathcal{E}) = \mathcal{E}$.*

Proof. Implied by Propositions 4.8.15, 4.8.20, and 4.8.21. □

Corollary 4.8.23. *If $\mathcal{E}$ is an RSES, then $N(\mathcal{E}) = \boldsymbol{R}'(\boldsymbol{C}_r(\mathcal{E}))$.*

Proof. Implied by Propositions 4.8.15 and 4.8.21. □

In some cases, a non-stable RES which maps to a stable CS can become stable when normalised, as in Example 4.8.24. In other cases, such as the one seen in Figure 4.23b, a non-stable RES in normal form can map to a stable CS.

Example 4.8.24. *Consider the RES* $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$*, where*

$$\begin{aligned} E &= \{a, b, c\} \\ F &= \emptyset \\ \mathsf{Con} &= \{\emptyset, \{a\}\} \\ \emptyset &\otimes \{b\} \vdash a \\ \emptyset &\otimes \{c\} \vdash a \end{aligned}$$

This is clearly neither normal nor stable, but $\boldsymbol{N}(\mathcal{E})$ *instead only has the enabling* $\emptyset \otimes \{b, c\} \vdash a$*, which makes it both normal and stable.*

We have now defined the categories and functors seen in Figure 4.1. However, since most reversible systems deal with causal or at least cause-respecting reversibility, we are interested in studying those subcategories further.

4.9 Cause-Respecting and Causal RESs and CSs

We now define cause-respecting and causal subcategories, **crRES** and **CRES**, of **RES** in Definitions 4.9.1 and 4.9.2. We show that the previously introduced functors preserve these properties between the categories. The definitions are based on the idea that if $\underline{e}$ is enabled in a configuration $X \cup \{e\}$, then e is enabled in X.

Definition 4.9.1 (Cause-respecting RES)**.** *An RES* $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$ *is* cause-respecting *if for any* $X \in \mathsf{Con}$*,* $Y \subseteq E$*, and* $e \in F$*, if* $X \otimes Y \vdash \underline{e}$*, then* $X \setminus \{e\} \otimes Y' \vdash e$ *for some* $Y' \subseteq Y$*.*

Definition 4.9.2 (Causal RES)**.** *An RES* $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$ *is* causal *if, given* $X \in \mathsf{Con}$*,* $Y \subseteq E$*, and* $e \in F$ *such that* $X \cup \{e\} \in \mathsf{Con}$*,* $X \otimes Y \vdash e$ *iff* $X \cup \{e\} \otimes Y \vdash \underline{e}$*.*

Proposition 4.9.3. *If* $\mathcal{E} = (E, F, \prec, \sharp, \rhd)$ *is a crRFES then* $\boldsymbol{E}_{rf}(\mathcal{E}) = (E, F, \mathsf{Con}, \vdash)$ *is a crRES.*

Proof. If $\mathcal{E} = (E, F, \prec, \sharp, \rhd)$ is a crRFES then for any $e, e' \in E$, if $e' \in X \in m_{RES}(e)$ then by definition $e' \prec e$. This means $e \rhd \underline{e'}$, and consequently whenever $X' \otimes Y' \vdash \underline{e'}$, we have $e \in Y'$.

□

We furthermore define a cause-respecting and causal CS (crCS and CaCS) in much the same way as a CRES in Definitions 4.9.4 and 4.9.5. The categories **crCS** and **CCS** consist of respectively crCSs and CaCSs, and the CS morphisms between them. The definition of a causal CS is basically the looping condition from [35], as the square condition from [35] is already part of the definition of a CS.

Definition 4.9.4 (Cause-respecting CS)**.** *A CS* $C = (E, F, \mathsf{C}, \to)$ *is* cause-respecting *if for any* $X \in \mathsf{C}$ *and* $B \subseteq F$*, if* $X \xrightarrow{A \cup \underline{B}} (X \cup A) \setminus B$*, then* $X \setminus B \xrightarrow{A \cup B} X \cup A$*.*

Definition 4.9.5 (Causal CS)**.** *A CS* $\mathcal{E} = (E, F, \mathsf{C}, \to)$ *is* causal *if for any configuration* $X \in \mathsf{C}$*,* $A, B \subseteq F$*, if* $X \xrightarrow{A \cup \underline{B}} (X \cup A) \setminus B$*, then* $(X \cup A) \setminus B \xrightarrow{B \cup \underline{A}} X$*.*

We show that $\boldsymbol{C}_r$ (Definition 4.7.5), $\boldsymbol{R}$ (Definition 4.7.14), and $\boldsymbol{R}'$ (Definition 4.8.11) preserve the cause-respecting (Proposition 4.9.6) and causal (Proposition 4.9.7) properties.

Proposition 4.9.6.

1. *If* $\mathcal{E}$ *is a crRES, then* $\boldsymbol{C}_r(\mathcal{E})$ *is a crFCS.*
2. *If* C *is a crFCS, then* $\boldsymbol{R}(\mathcal{E})$ *is a crRES.*
3. *If* C *is a crSFCS, then* $\boldsymbol{R}'(\mathcal{E})$ *is a crRSES.*

Proof. We prove the three statements separately in Appendix B.36. □

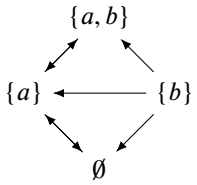
Proposition 4.9.7.

1. *If $\mathcal{E}$ is a CRES then $\boldsymbol{C}_r(\mathcal{E})$ is a CaFCS.*
2. *If $\boldsymbol{C}$ is a CaFCS then $\boldsymbol{R}(\mathcal{E})$ is a CRES.*
3. *If $\boldsymbol{C}$ is a CaSFCS then $\boldsymbol{R}'(\boldsymbol{C})$ is a CRSES.*

Proof. We prove the three statements separately in Appendix B.37, similarly to the previous proposition. □

The cause-respecting and causal RPESs, RAESs, and RFESs cannot always be modelled by cause-respecting and causal CSs and RESs, as shown by Example 4.9.8. However, the reachable part of the CS is cause-respecting (resp. causal) and forwards-reachable, as proved in Theorem 4.54 and Propositions 3.37, 3.38, and 4.57 of [82]. We therefore define reachably cause-respecting and reachably causal CSs (rcrCS and rCaCS) in Definition 4.9.9. We define categories **rcrCS** and **rCaCS** of rcrCSs and rCaCSs and the CS morphisms between them.

Example 4.9.8. *Consider the RAES $\mathcal{E} = (E, F, \prec, \lhd)$ where $E = F = \{a, b\}$, $a \prec b$, $a \prec \underline{a}$, $b \prec \underline{b}$, and $b \rhd \underline{a}$. This is clearly causal, however when mapping to an RES, $\boldsymbol{P}_{ar}(\mathcal{E}) = (E, F, \mathsf{Con}, \vdash)$, we get the enablings $\emptyset \otimes \emptyset \vdash a$, $\{a\} \otimes \emptyset \vdash b$, $\{b\} \otimes \emptyset \vdash \underline{b}$, and $\{a\} \otimes \{b\} \vdash \underline{a}$. Clearly this is not causal. This is also expressed in the CS in the following diagram:*



Here b can be reversed from any configuration containing b, but a can only be reversed if b is not present. Conversely, a can be added to any configuration not containing a, but b can only be added if a is present. However, this gives us the unreachable configuration $\{b\}$, from which we have three transitions without corresponding reverse transitions.

Definition 4.9.9 (Reachably cause-respecting and causal CS)**.**

1. *A CS $(E, F, \mathsf{C}, \rightarrow)$ is* reachably cause-respecting *if for any reachable configuration $X \in \mathsf{C}$, $A \subseteq E$, and $B \subseteq F$, if $X \xrightarrow{A \cup \underline{B}} (X \cup A) \setminus B$, then $X \setminus B \xrightarrow{A \cup B} X \cup A$.*
2. *A CS $\mathcal{E} = (E, F, \mathsf{C}, \rightarrow)$ is* reachably causal *if for any reachable configuration $X \in \mathsf{C}$, $A, B \subseteq F$, if $X \xrightarrow{A \cup \underline{B}} (X \cup A) \setminus B$, then $(X \cup A) \setminus B \xrightarrow{B \cup \underline{A}} X$.*

Clearly, $\boldsymbol{C}$ is reachably cause-respecting or causal if and only if $\mathsf{Reach}(\boldsymbol{C})$ is cause-respecting or causal.

Proposition 4.9.10.

1. *If $\mathcal{E}$ is a crRAES, then $\boldsymbol{C}_{ra}(\mathcal{E})$ is an rcrCS.*
2. *If $\mathcal{E}$ is a CRAES, then $\boldsymbol{C}_{ra}(\mathcal{E})$ is an rCaCS.*

Proof.

1. Suppose $\mathcal{E}$ is a crRAES. Then for any reachable configuration X of $\boldsymbol{C}_{ra}(\mathcal{E})$, if $X \xrightarrow{\underline{B}} X \subseteq B$, then $X \setminus B \xrightarrow{B} X$ [82]. As $\boldsymbol{C}_{ra}(\mathcal{E})$ is also an SCS (Proposition 4.8.8), we get from item 4 of Definition 4.1.1 of a CS and Item 2d of Definition 4.8.7 of an SCS, that $\boldsymbol{C}_{ra}(\mathcal{E})$ is an rcrCS.
2. Suppose $\mathcal{E}$ is a CRAES, then $\boldsymbol{C}_{ra}(\mathcal{E})$ is an rCaCS according to [82].

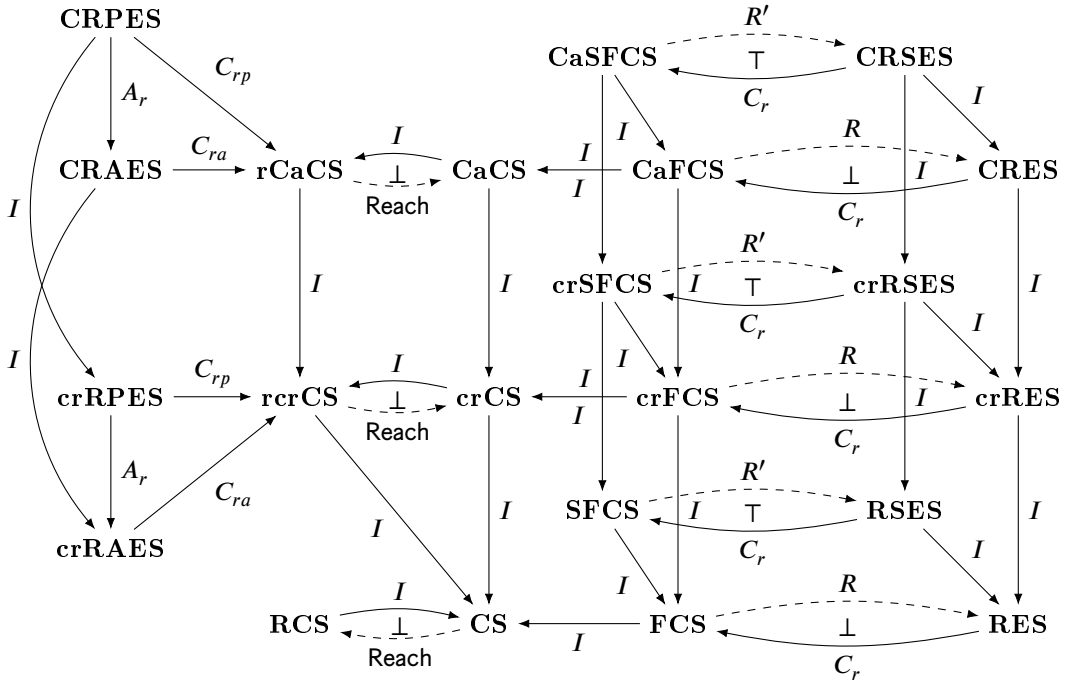


Figure 4.24: Categories of prime and asymmetric reversible event structures and their cause-respecting and causal subcategories and functors between them. Note that while there of course exist stable subcategories of **rcrCS** and **rCaCS**, we have excluded them in order to reduce clutter. Dashed lines indicate functors which do not commute with the rest of the diagram. The definitions of the categories and functors can be seen in the following table. Those not listed in the table are intersections of other categories, e.g. CRSES is an RES which is both causal and stable, fulfilling both Definitions 4.8.1 and 4.9.2.

CS	Definition 4.1.1	**RSES**	Definition 4.8.1	Reach	Definition 4.1.20
RCS	Definition 4.1.19	**crRES**	Definition 4.9.1	C_{rp}	Definition 4.2.4
CRPES	Definition 4.2.18	**CRES**	Definition 4.9.2	C_{ra}	Definition 4.3.2
crRPES	Definition 4.2.18	**crCS**	Definition 4.9.4	A_r	Definition 4.3.15
CRAES	Definition 4.3.17	**CaCS**	Definition 4.9.5	C_r	Definition 4.7.5
crRAES	Definition 4.3.17	**rCaCS**	Definition 4.9.9	R	Definition 4.7.14
RES	Definition 4.7.1	**rcrCS**	Definition 4.9.9	R'	Definition 4.8.11
FCS	Definition 4.7.11				

□

Proposition 4.9.11. *If $\mathcal{E} = (E, F, \prec, \sharp, \rhd)$ is a CRFES, then $\mathbf{C}_{fr}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$ is a rCaCS.*

Proof. We prove that $(E, F, \mathsf{C}, \rightarrow)$ has the required transitions in Appendix B.38. □

This gives us the causal and cause-respecting categories seen in Figure 4.24. The figure illustrates that properties such as being causal or cause-respecting are preserved by our functors. It also shows which of our categories are subcategories of or can be represented by other categories. For example the subcategory **crRSES** contains RESs which are both cause-respecting and stable, and is therefore a subcategory of both **RSES** and **crRES**. This figure can also be used when considering what to use when modelling a system with true concurrency. For example, in [84], RESs were used to model biochemical systems with complex causal relationships between the creating and dissolving bonds between molecules. This figure shows that these systems can also be modelled by FCSs, but as they are non-causal, not by crRESs. If one wanted to work further with these models, it would be worth investigating whether such systems are always stable.

4.10 Conclusion

Event structures are a popular model of true concurrency, of which there exist multiple variants [8, 21, 64, 77, 95]. More recently, reversible variants of some of these have been defined and used to describe reversible concurrent systems [82, 84].

In this chapter we have defined categorically the configuration systems and reversible event structures introduced by [82, 84]. This has allowed us to verify that the mappings between these structures are functors. We have also defined categories of event structures for which no previous reversible variants have been defined, namely reversible bundle, extended bundle, and flow event structures. We have shown that we can map between these similarly to their forward-only variants presented in Chapter 3, with the key exception that reversible general event structures include asymmetric conflict.

In addition we have discovered a close correspondence between the reversible general event structures of [84] and the configuration systems of [82]. In particular we showed that RESs could be normalised and stable CSs and normal form stable RESs are isomorphic (Corollary 4.8.17). This resembles the relationship between PESs and Domains seen in Chapter 3.

We found that the notion of morphism used by flow event structures [30] was harder than the others to transfer to a reversible setting because it relies directly on the definition of configurations. The closest thing the reversible event structures have to the configurations of the forwards-only event structures we saw in Chapter 3 is the reachable configurations of their CS representation, but these are not immediately obvious from the definition of the event structure. Our RFES category therefore does not have any outgoing or incoming functors, only mappings which do not preserve morphisms.

Reversible event structures are unique among reversible formalisms in that they feature non-causal reversibility, the ability for an event to reverse from a configuration in which it has caused another event. Since most reversible systems only allow causal reversibility, we have defined causal subcategories of the categories of reversible event structures. As we showed in Example 4.5.1, non-causal reversibility is necessary for using events to model common control predicates such as rollback [58]. We consider this an important advantage over other formalisms previously used for describing true-concurrency semantics of reversible process-calculi such as rigid families [34] and configuration structures [4].

With the categories of reversible event structures and configuration systems seen in Figure 4.1 now defined, we can decide which of them to use for our event structure semantics. As we have not been able to define products of RPESs, RAESs, describing the semantics of parallel composition in those will likely be difficult. Our category of RFESs caused us problems since the morphisms do not map to CS morphisms, so we also decide not to use those. This leaves RBESs as the least expressive, and therefore likely easiest to work with of the remaining. In addition any RBES semantics will be easily converted to REBES semantics when we want to use asymmetric conflict to model control predicates as in Example 4.5.1. We could also use RESs, but we do not see where their added expressiveness would be necessary.

The key result of this chapter is the close correspondence between CSs and RESs, and even closer between nfRSESs and SFCSs described by Theorem 4.7.16 and Corollary 4.8.17.

Chapter 5

Event structure semantics of (controlled) reversible CCS

Reversible process calculi have been studied in works such as [33, 35, 37, 59, 61, 80]. One feature of such reversible processes compared to forward-only processes is their sensitivity to true concurrency distinctions [79]. For instance, using CCS notation, the processes $a|b$ and $a.b + b.a$, which are respectively a parallel composition and a choice between two orderings of events, are equivalent under interleaving semantics; however in a reversible setting we can distinguish them by noting that $a|b$ allows us to perform a followed by b and then to reverse a, which is impossible for $a.b + b.a$. This motivates us to use event structures to formulate a truly concurrent semantics of a reversible process calculus.

Two reversible forms of CCS have been proposed: RCCS [35] and CCSK [80] (both recalled in Chapter 2). RCCS creates separate memories to store past (executed) actions, which is known as dynamic reversibility. On the other hand, CCSK annotates past actions with keys within the processes themselves, which is known as static reversibility. We formulate event structure semantics for CCSK rather than RCCS, since the semantics for past and future actions can be defined in a similar manner, rather than having to encompass both processes and memories. We note that Lanese *et al.* [57, 68] showed that RCCS and CCSK can be encoded in one another, meaning one can use their encoding in conjunction with our event structure semantics to obtain an event structure semantics for RCCS. The encoding from RCCS to CCSK generates a process with an isomorphic LTS. However, as detailed in Chapter 2, RCCS reachable processes are not closed under parallel composition. Since RCCS is non-compositional, [57] argues that any results regarding the compositionality of the encoding are meaningless. This means we cannot combine this encoding with denotational semantics to get a direct mapping from RCCS to event structures, but will need to go through CCSK.

Event structures have been used for modelling forward-only process calculi [21, 32, 94]. Cristescu *et al.* [34] used rigid families [27], related to event structures, to describe the semantics of Rπ [33]. However, their semantics require a process to first reverse all actions to find the original process, map this process to a rigid family, and then apply each of the reversed memories in order to reach the current state of the process. Aubert and Cristescu [4] used a similar approach to describe the semantics of RCCS processes without auto-concurrency, auto-conflict, or recursion as configuration structures. By contrast, we use the causal subcategory of RBESs to map a CCSK process (with auto-concurrency, auto-conflict, and recursion) with past actions directly to a (reversible) event structure in a strictly denotational fashion. When using our semantics in conjunction with the encoding from [57, 68] to obtain true concurrency semantics of RCCS, one can consider the encoding equivalent to reversing the past actions in order to reach the original structure of the process in [4]. Out of the categories of reversible event structures presented in the previous chapter, we choose RBESs because they allow an event to have multiple possible conflicting causes. This makes it possible to model parallel composition without having one action correspond to multiple events, which becomes particularly important once we add controlled reversibility to CCSK. This is also possible in REBESs, RFESs, RESs, or SRESs, but we do not need the additional expressiveness offered by them. If one wants the event structure semantics of CCSK expressed in either of those then the mappings presented in Definitions 4.5.13, 4.6.15, or 4.7.24 can be applied to the RBES created by the denotational semantics.

CCSK uses uncontrolled causal reversibility, which has the drawback of allowing a process to get into a loop doing and undoing the same action indefinitely; there is no control on whether or when a computation reverses. We modify CCSK to control reversibility by adding the *rollback* introduced for Roll-π in [58]. In Roll-CCSK every action receives a tag γ, and the process only reverses when reaching a $\mathsf{roll}\ \gamma$ primitive, upon which the action tagged with γ, together with all actions caused by it, are reversed. As in Roll-π, the rollback in Roll-CCSK is *maximally permissive*, meaning that any subset of reached rollbacks may be executed, even if one of them rolls back the actions leading to another. The operational semantics of rollback work somewhat differently in Roll-CCSK from Roll-π, since Roll-π has a set of memories describing past actions in addition to a π-calculus process, while CCSK has the past actions incorporated into the structure of the process, meaning that it is harder to know whether one has found all the actions necessary to reverse. Roll-CCSK allows recursion using binding on tags.

Once a $\mathsf{roll}\ \gamma$ event has happened, we need to ensure that not only are the events caused by the γ-tagged action a_γ able to reverse, but they cannot re-occur until the rollback is complete, at which point the $\mathsf{roll}\ \gamma$ event is reversed. This requires us to model *asymmetric* conflict between $\mathsf{roll}\ \gamma$ and events caused by a_γ (apart from $\mathsf{roll}\ \gamma$ itself). We use REBESs to give an event structure semantics of rollback in Section 5.2. Using REBESs allows us to easily translate semantics from RBES to REBES, since they are similar. RESs also include asymmetric conflict, but we do not need the additional expressiveness given by different causes having different preventing sets. Note that we do not restrict ourselves to the causal subcategory of REBESs, since reversibility in Roll-CCSK is not necessarily causal. An action a_γ tagged with γ is a cause of $\mathsf{roll}\ \gamma$, but we want a_γ to reverse before $\mathsf{roll}\ \gamma$ does.

Section 5.1 defines the event structure semantics of CCSK. Section 5.2 defines Roll-CCSK, a version of CCSK where reversibility is controlled by rollback, and its operational semantics. And Section 5.3 uses reversible extended bundle event structures to describe the event structure semantics of Roll-CCSK.

5.1 Event Structure Semantics of CCSK

We will now use the RBESs described in Section 4.4 to describe the semantics of CCSK. Unlike the event structure semantics of forward-only CCS [21, 94], our semantics will generate both an event structure and an *initial configuration* containing all the events corresponding to past actions. This means that if $P \to P'$ then P and P' will be described by the same, or at least isomorphic, event structures with different initial states.

First we define the operators we will use in the semantics, particularly restriction, parallel composition, choice, and action prefixes. Restriction is done by simply removing any events associated with the restricted action.

Definition 5.1.1 (Restriction). *Given an LRBES, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, restricting $\mathcal{E}$ to $E' \subseteq E$ creates $\mathcal{E} \restriction E' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$ where:*

1. $F' = F \cap E'$;
2. $\mapsto' = \mapsto \cap (\mathcal{P}(E') \times (E' \cup \underline{F'}))$;
3. $\sharp' = \sharp \cap (E' \times E')$;
4. $\rhd' = \rhd \cap (E' \times \underline{F'})$;
5. $\lambda' = \lambda \restriction_{E'}$;
6. $\mathsf{Act} = \mathsf{ran}(\lambda')$.

Parallel composition uses the product of RBESs, labels any event corresponding to a synchronisation τ, and removes any invalid events describing an impossible synchronisation.

Definition 5.1.2 (Parallel). *Given LRBESs $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$, their parallel composition is $\mathcal{E}_0 || \mathcal{E}_1 = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act}) \restriction \{e \mid \lambda(e) \neq 0\}$ where:*
$(E, F, \mapsto, \sharp, \rhd) = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0) \times (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1)$

$$\lambda(e) = \begin{cases} \lambda_0(e_0) & \text{if } e = (e_0, *) \\ \lambda_1(e_1) & \text{if } e = (*, e_1) \\ \tau & \text{if } e = (e_0, e_1) \text{ and } \lambda_0(e_0) = \overline{\lambda_1(e_1)} \\ 0 & \text{if } e = (e_0, e_1) \text{ and } \lambda_0(e_0) \neq \overline{\lambda_1(e_1)} \end{cases}$$

$\mathsf{Act} = \mathsf{Act}_0 \cup \mathsf{Act}_1 \cup \{0, \tau\}$

Choice, which act as a coproduct of LRBESs, simply uses the coproduct of RBESs, and defines the labels as expected.

Definition 5.1.3 (Choice). *Given LRBESs $\mathcal{E}_0$ and $\mathcal{E}_1$, the choice between them is $\mathcal{E}_0 + \mathcal{E}_1 = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ where:*

$\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$

$(E, F, \mapsto, \sharp, \rhd) = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0) + (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1)$

$\lambda(j, e) = \lambda_j(e)$

$\mathsf{Act} = \mathsf{Act}_0 \cup \mathsf{Act}_1$

Proposition 5.1.4. *If $\mathcal{E}_0$ and $\mathcal{E}_1$ are LRBESs, then $\mathcal{E} = \mathcal{E}_0 + \mathcal{E}_1$ with injections i_0 and i_1 such that $i_j(j, e) = e$ is their coproduct.*

Proof. We prove that we have the required morphisms in Appendix C.1. □

Causally prefixing an action onto an event structure means the new event causes all other events and is prevented from reversing by all other events.

Definition 5.1.5 (Causal Prefix). *Given an LRBES $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, an event $e \notin E$, and a label α, we add e labelled α to the beginning of $\mathcal{E}$ to get $\alpha(e).\mathcal{E} = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$ where:*

1. $E' = E \cup e$
2. $F' = F \cup e$
3. $\mapsto' = \mapsto \cup (\{\{e\}\} \times (E \cup \{\underline{e}\}))$
4. $\sharp' = \sharp$
5. $\rhd' = \rhd \cup (E \times \{\underline{e}\})$
6. $\lambda' = \lambda[e \mapsto \alpha]$
7. $\mathsf{Act}' = \mathsf{Act} \cup \{\alpha\}$

Now that we have defined the main operations of the process calculus, we define the event structure semantics in Table 5.1. We do this using rules of the form $\{\!|P|\!\}_l = \langle \mathcal{E}, \mathsf{Init}, k \rangle$ wherein l is the level of unfolding, which we use to model recursion, $\mathcal{E}$ is an LRBES, Init is the initial configuration, and $k : \mathsf{Init} \to \mathcal{K}$ is a function assigning communication keys to the past actions, which we use in parallel composition to determine which synchronisations of past actions to put in Init.

Note that the only difference between a future and a past action is, that the event corresponding to a past action is put in the initial state and given a communication key.

Example 5.1.6. *The CCSK process $a.b \mid \overline{a}$ can be represented by the RBES with events labelled a, $\overline{a}$, τ, and b, the bundle $\{a, \tau\} \mapsto b$, the conflicts $a \sharp \tau$ and $\overline{a} \sharp \tau$, and the preventions $b \rhd \underline{a}$ and $b \rhd \underline{\tau}$, creating the LRBES seen in Figure 5.1.*

We say that $\{\!|P|\!\} = \sup_{l \in \mathcal{N}} \{\!|P|\!\}_l$. This means we need to show that there exists such a least upper bound of the levels of unfolding. As shown in [45], ordering closed BESs by restriction produces a complete partial order. Since our LRBESs do not have overlapping bundles ($X \mapsto e^*$ and $X' \mapsto e^*$ implies $X \neq X'$ or $X \cap X' = \emptyset$) they are closed, and we can use a similar ordering.

Definition 5.1.7 (Ordering of LRBESs). *Given LRBESs $\mathcal{E}_0$ and $\mathcal{E}_1$, we get $\mathcal{E}_0 \leq \mathcal{E}_1$ if $\mathcal{E}_0 = \mathcal{E}_1 \upharpoonright E_0$.*

$\{\!|0|\!\}_l =$ $\langle(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset),\emptyset,\emptyset\rangle$

$\{\!|P_0 + P_1|\!\}_l =$ $\langle \mathcal{E}_0 + \mathcal{E}_1, \mathsf{Init}, k\rangle$ where
For $i \in \{0,1\}$, $\{\!|P_i|\!\}_l = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i\rangle$
$\mathsf{Init} = \{(j,e) \mid j \in \{0,1\} \text{ and } e \in \mathsf{Init}_j\}$
$k((j,e)) = k_j(e)$ if $e \in \mathsf{Init}_j$

$\{\!|\alpha.P|\!\}_l =$ $\langle \alpha(e).(E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}),\mathsf{Init},k\rangle$ for e fresh for E where
$\{\!|P|\!\}_l = \langle (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}),\mathsf{Init},k\rangle$

$\{\!|\alpha[m].P|\!\}_l =$ $\langle \alpha(e).(E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}),\mathsf{Init}\cup\{e\},k[e\mapsto m]\rangle$ for e fresh for E where
$\{\!|P|\!\}_l = \langle (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}),\mathsf{Init},k\rangle$

$\{\!|P_0 \mid P_1|\!\}_l =$ $\langle (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}),\mathsf{Init},k\rangle$ where
For $i \in \{0,1\}$, $\{\!|P_i|\!\}_l = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i\rangle$
$(E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}) = \mathcal{E}_0 || \mathcal{E}_1$
$\mathsf{Init} = \{(e_0,e_1) \mid e_0 \in \mathsf{Init}_0, e_1 \in \mathsf{Init}_1, k_0(e_0) = k_1(e_1)\}\cup$
$\{(*,e_1) \mid e_1 \in \mathsf{Init}_1 \text{ and } \nexists e_0 \in \mathsf{Init}_0.\lambda_0(e_0) = \overline{\lambda_1(e_1)} \text{ and } k_0(e_0) = k_1(e_1)\}\cup$
$\{(e_0,*) \mid e_0 \in \mathsf{Init}_0 \text{ and } \nexists e_1 \in \mathsf{Init}_1.\lambda_0(e_0) = \overline{\lambda_1(e_1)} \text{ and } k_0(e_0) = k_1(e_1)\}$
$$k(e) = \begin{cases} k_0(e_0) & \text{if } e = (e_0,*) \\ k_1(e_1) & \text{if } e = (*,e_1) \\ k_0(e_0) & \text{if } e = (e_0,e_1) \quad -\textit{ note that } k_0(e_0) = k_1(e_1) \end{cases}$$

$\{\!|P \setminus A|\!\}_l =$ $\langle \mathcal{E} \upharpoonright \{e \mid \lambda(e) \notin \mathcal{A}\}, \mathsf{Init} \cap \{e \mid \lambda(e) \notin \mathcal{A}\}, k \upharpoonright \{e \mid \lambda(e) \notin \mathcal{A}\}\rangle$ where
$\{\!|P|\!\}_l = \langle \mathcal{E},\mathsf{Init},k\rangle$
$\mathcal{A} = A \cup \overline{A}$

$\{\!|P[f]|\!\}_l =$ $\langle (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}),\mathsf{Init},k\rangle$ where
$\{\!|P|\!\}_l = \langle (E,F,\mapsto,\sharp,\rhd,\lambda',\mathsf{Act}'),\mathsf{Init},k\rangle$
$\mathsf{Act} = f(\mathsf{Act}')$
$\lambda = f \circ \lambda'$

$\left\{\!\left|A\langle\tilde{b}\rangle\right|\!\right\}_0 =$ $\langle(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset),\emptyset,\emptyset\rangle$

$\left\{\!\left|A\langle\tilde{b}\rangle\right|\!\right\}_l =$ $\left\{\!\left|P_A\{^{\tilde{b}}/_{\tilde{a}}\}\right|\!\right\}_{l-1}$ where $A(\tilde{a}) = P_A$

Table 5.1: RBES-semantics of CCSK

We can see that $\leq$ is a partial order with the empty LRBES as its minimum.

Proposition 5.1.8. *Any ω-chain $\mathcal{E}_0 \leq \mathcal{E}_1 \leq \mathcal{E}_2 \ldots$ has a least upper bound $\mathcal{E} = (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act})$ where:*

1. $E = \bigcup_{n\in\omega} E_n$;
2. $F = \bigcup_{n\in\omega} F_n$;
3. $X \mapsto e^*$ *if for all $n \in \omega$ such that $e \in E_n$, we have $(X \cap E_n) \mapsto e^*$;*
4. $\sharp = \bigcup_{n\in\omega} \sharp_n$;
5. $\rhd = \bigcup_{n\in\omega} \rhd_n$;
6. $\lambda(e) = l$ *if there exists $n \in \omega$ such that $\lambda_n(e) = l$;*
7. $\mathsf{Act} = \bigcup_{n\in\omega} \mathsf{Act}_n$;

Proof. We show that this is least upper bound in Appendix C.2. □



Figure 5.1: LRBES and configurations of the process in Example 5.1.6

This means that, given a set of events A, with $\mathbb{E}_A$ being the set of LRBESs $(E, F, \mapsto, \sharp, \triangleright, \lambda, \mathsf{Act})$ such that $E \subseteq A$, $(\mathbb{E}_A, \leq)$ is a complete partial order. We then need to show that our operations of restriction (Lemma 5.1.9), parallel composition (Lemma 5.1.10), choice (Lemma 5.1.11), and action prefix (Lemma 5.1.12) are monotonic.

Lemma 5.1.9.

1. *If $\mathcal{E}_0 \leq \mathcal{E}_1$ and $E \subseteq E_1$, then $\mathcal{E}_0 \upharpoonright (E \cap E_0) \leq \mathcal{E}_1 \upharpoonright E$.*
2. *For any ω-chain $\mathcal{E}_0 \leq \mathcal{E}_1 \leq \mathcal{E}_2 \ldots$ and $E \subseteq \bigcup_{n\in\omega} E_n$, we have $(\bigsqcup_{n\in\omega} \mathcal{E}_n) \upharpoonright E = \bigsqcup_{n\in\omega} (\mathcal{E}_n \upharpoonright (E \cap E_n))$.*

Proof. Follows from Definition 5.1.7. □

Lemma 5.1.10.

1. *Given LRBESs $\mathcal{E}_0 \leq \mathcal{E}_1$ and $\mathcal{E}$, $\mathcal{E}_0 || \mathcal{E} \leq \mathcal{E}_1 || \mathcal{E}$.*
2. *For any ω-chain $\mathcal{E}_0 \leq \mathcal{E}_1 \leq \mathcal{E}_2 \ldots$ and LRBES $\mathcal{E}$, we have $(\bigsqcup_{n\in\omega} \mathcal{E}_n) || \mathcal{E} = \bigsqcup_{n\in\omega} (\mathcal{E}_n || \mathcal{E})$.*

Proof. Follows from Definitions 5.1.2 and 5.1.7. □

Lemma 5.1.11.

1. *Given LRBESs $\mathcal{E}_0 \leq \mathcal{E}_1$ and $\mathcal{E}$, $\mathcal{E}_0 + \mathcal{E} \leq \mathcal{E}_1 + \mathcal{E}$.*
2. *For any ω-chain $\mathcal{E}_0 \leq \mathcal{E}_1 \leq \mathcal{E}_2 \ldots$ and LRBES $\mathcal{E}$, we have $(\bigsqcup_{n\in\omega} \mathcal{E}_n) + \mathcal{E} = \bigsqcup_{n\in\omega} (\mathcal{E}_n + \mathcal{E})$.*

Proof. Follows from Definitions 5.1.3 and 5.1.7. □

Lemma 5.1.12.

1. *Given LRBESs $\mathcal{E}_0 \leq \mathcal{E}_1$ an event $e \notin E_1$, and a label α, $\alpha(e).\mathcal{E}_0 \leq \alpha(e).\mathcal{E}_1$.*
2. *For any ω-chain $\mathcal{E}_0 \leq \mathcal{E}_1 \leq \mathcal{E}_2 \ldots$ an event $e \notin E_1$, and a label α, we have $\alpha(e).(\bigsqcup_{n\in\omega} \mathcal{E}_n) = \bigsqcup_{n\in\omega} (\alpha(e).\mathcal{E}_n)$.*

Proof. Follows from Definitions 5.1.1 and 5.1.7. □

Knowing our operations are monotonic, we can show that we use our ordering on the level of unfolding of a process, and so $\{\!|P|\!\} = \sup_{l\in\mathcal{N}} \{\!|P|\!\}_l$ exists for any P.

Proposition 5.1.13 (Unfolding)**.** *Given a forwards-reachable process P and a level of unfolding k, if $\{\!|P|\!\}_k = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ and $\{\!|P|\!\}_{k-1} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$, then $\mathcal{E}' \leq \mathcal{E}$, $\mathsf{Init} = \mathsf{Init}'$, and $k = k'$.*

Proof. We have proved in Lemmas 5.1.9-5.1.12 that all the operations used for defining the $\mathcal{E}$ and $\mathcal{E}'$ are monotonic, so clearly $\mathcal{E}' \leq \mathcal{E}$, and since P has been generated from a standard process, we cannot have any $\alpha[m]$ inside a recursion, as it would have to have been unfolded first. □

In order to prove that our event structure semantics correspond with the operational semantics for CCSK defined in [83] we first show that event structures generated by our semantics are causal according to Definition 4.4.13.

Proposition 5.1.14. *Given a process P such that $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E}$ is causal.*

Proof. We prove this by structural induction on P in Appendix C.3. □

We then show that structurally congruent processes will generate isomorphic event structures.

Proposition 5.1.15 (Structural Congruence)**.** *Given processes P and P' such that $P \equiv P'$, $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, and $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$, there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $f(\mathsf{Init}) = \mathsf{Init}'$ and for all $e \in \mathsf{Init}$, $k(e) = k'(f(e))$.*

Proof. We prove this by case analysis on structural congruence in Appendix C.4. □

Before we show our correspondence between actions in the process and in the event structure, we show that a reachable process has an empty initial state if and only if it does not contain any past actions.

Lemma 5.1.16 (standard)**.** *Given a process P such that $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ and there exists a standard process Q such that $Q \to^* P$, we have $\mathsf{std}(P)$ if and only if $\mathsf{Init} = \emptyset$.*

Proof. As the only rule which can add events to an empty Init is $\{\!|\alpha[m].P|\!\}$, clearly $\mathsf{Init} = \emptyset$ if $\mathsf{std}(P)$.

If $I = \emptyset$, then clearly we cannot have any $\alpha[m]$ in P, which are not guarded by a restriction on α. But if such a restricted communication has occurred in P, then there must exist a parallel $\overline{a}[m]$ inside the same restriction, meaning the corresponding event $(e_\alpha, e_{\overline{\alpha}})$ has the label τ, not α, and would therefore be in Init. Therefore we must have $\mathsf{std}(P)$. □

We now show that reachable processes have conflict-free initial states.

Lemma 5.1.17 (Reachable)**.** *If P is forwards-reachable and $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, then Init is conflict-free in $\mathcal{E}$.*

Proof. We prove this by structural induction on P in Appendix C.5. □

Finally we show in Theorems 5.1.18 and 5.1.19 that given a process P with a conflict-free initial state, including any reachable process, there exists a transition $P \xrightarrow{\mu} P'$ if and only if the event structure corresponding to P is isomorphic to the event structure corresponding to P' and an event e labelled μ exists such that e is available in P's initial state, and P''s initial state is P's initial state with e added.

Theorem 5.1.18. *If $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $\mathcal{C}_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \to)$, Init is conflict-free, and there exists a transition $P \xrightarrow{\mu[m]} P'$ such that $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$, then there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ and a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

Proof. We prove this by structural induction on $P \xrightarrow{\mu[m]} P'$ in Appendix C.6 using Proposition 5.1.15. We describe the morphisms f and g in each case and find the event e with the correct label. we can define the isomorphisms f and g by simple induction on P, and in most cases it is straightforward to find e and prove the remaining conditions hold. In the synchronisation case where $P = Q \mid R$, $P' = Q' \mid R'$, $Q \xrightarrow{\alpha[m]} Q'$, $R \xrightarrow{\overline{\alpha}[m]} R'$, and $\mu = \tau$, we need to use Lemma 5.2 of [80] to show that Init does not contain any events in conflict with e. □

Having shown that each forwards transition in the operational semantics corresponds to one in the generated event structure, we now show the inverse.

Theorem 5.1.19. *Let P be a reachable process. If $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $C_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, Init is conflict-free, and there exists a transition $\mathsf{Init} \xrightarrow{e} X$ in $C_{br}(\mathcal{E})$, then there exists a a key m and a transition $P \xrightarrow{\lambda(e)[m]} P'$, such that $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$ and there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ such that $f \circ k' = k[e \mapsto m]$ and $f(X) = \mathsf{Init}'$.*

Proof. We prove this by structural induction on P in Appendix C.7. We can always generate a fresh m, and it is straightforward from the semantics to show that the transition $P \xrightarrow{\lambda(e)[m]} P'$ exists for such a fresh m. We again describe f and g inductively on P, and show that they are isomorphisms using similar methods to Theorem 5.1.18, since they are the same morphisms in both proofs. □

Corollary 5.1.20. *Given a process P such that $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, Init is forwards-reachable in $\mathcal{E}$ if and only if P is forwards-reachable.*

Since we showed in Proposition 5.1.14 that any event structures generated by processes are causal, it follows that we get a similar correspondence between the reverse transitions of processes and event structures.

Theorem 5.1.21. *If $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $C_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, Init is conflict-free, and there exists a transition $P \overset{\mu[m]}{\rightsquigarrow} P'$ such that $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$, then there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ and a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

Proof. Implied by Proposition 5.1.14, Theorem 5.1.18, and Corollary 5.1.20. □

Theorem 5.1.22. *If $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $C_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, Init is conflict-free, and there exists a transition $\mathsf{Init} \xrightarrow{e} X$ in $C_{br}(\mathcal{E})$, then there exists a a key m and a transition $P \overset{\lambda(e)[m]}{\rightsquigarrow} P'$, such that $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$ and there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ such that $f \circ k' = k[e \mapsto m]$ and $f(X) = \mathsf{Init}'$.*

Proof. Implied by Proposition 5.1.14, Theorem 5.1.19, and Corollary 5.1.20. □

We have now proved operational correspondence between the operational semantics of CCSK and the event structure semantics presented in this section. These are the first true concurrency semantics of a reversible process calculus to map directly from a process with past actions to a model of true concurrency with an initial state. It is therefore key that any state of the same process maps to the same event structure with a different initial state, showing that both for the process and the generated event structure, any reachable state is reachable from any other reachable state.

Proving operational correspondence on reverse actions was made simpler by both the calculus and the generated event structures having uncontrolled causal reversibility. However, uncontrolled reversibility comes with problems, in that it allows a process or event structure to do and undo the same action indefinitely. We would therefore like CCSK to have a way to control its reversibility.

5.2 Roll-CCSK

In this section we add a rollback operator, similar to that of Roll-π, to CCSK. The semantics of Roll-π are not directly translatable to CCSK, as they make use of the fact that one can know, when looking at a memory, whether the communication it was associated with was with another process or not, and therefore, for a given subprocess P and a memory m, one knows whether all the memories and subprocesses caused by m are part of P. In CCSK, this is not the case, as the roll in the subprocess $\alpha_\gamma[n].\mathsf{roll}\ \gamma$, where γ is a *tag* denoting which rollback rolls back which action, may or may not require rolling back the other end of the α communication, and all actions caused by it. This makes it significantly harder to know when to stop rolling back, as (unless all the actions needing to be rolled back were τ-actions) we do not know that the roll is complete. We therefore need to check at every instance of parallel composition whether any communication with actions being rolled back has taken place, and if so roll back those actions and all actions caused by them. This may include rolling back additional actions from the subprocess containing the roll, as in

$a[n_1].\overline{b}[n_2] \mid c[n_3].(\overline{a}_\gamma[n_1].\mathsf{roll}\ \gamma \mid b[n_2])$, where it does not become clear that $b[n_2]$ needs to be reversed during the roll until the outer parallel composition. Additionally, roll-π uses asymmetric communication, meaning the memories that need to be reversed form a sequence, as opposed to one communication potentially causing more actions at both ends. This causes further problems when trying to define low-level semantics, and we therefore only define these through event structures.

The syntax of Roll-CCSK is as follows:

$$P ::= \alpha_\gamma.P \mid \alpha_\gamma[n].P \mid P_0 + P_1 \mid P_0|P_1 \mid P \setminus A \mid P[f] \mid A\langle \tilde{b}, \tilde{\gamma} \rangle \mid 0 \mid \mathsf{roll}\ \gamma \mid \mathsf{rolling}\ \gamma$$

Most of the syntax is the same as CCSK and CCS, but adding tags and rolls as described above, and $\mathsf{rolling}\ \gamma$, which denotes a roll in progress, the necessity of which is justified later. From now on we will use $\alpha.P$ to denote $\alpha_\gamma.P$ where no $\mathsf{roll}\ \gamma$ exists in P. We also add a tuple of tags to our process constants, which get substituted just like the actions in a new structural congruence rule:

$$A\langle \tilde{b}, \tilde{\delta} \rangle \equiv P_A\{{}^{\tilde{b}}/_{\tilde{a}}\, {}^{\tilde{\delta}}/_{\tilde{\gamma}}\} \text{ if } A(\tilde{a}, \tilde{\gamma}) = P_A$$

Before defining our semantics of rollback, we first define causal dependence and projection similarly to [58], which we will use to define our semantics.

Definition 5.2.1 (Causal dependence)**.** *Let P be a process. Then the binary relation $\leq_P$ is the smallest relation on $\mathsf{key}(P)$ satisfying:*

- *If there exists a process P' and past actions $\alpha_\gamma[n]$ and $\beta_{\gamma'}[m]$ such that $\alpha_\gamma[n].P'$ is a subprocess of P and $\beta_{\gamma'}[m]$ occurs in P' then $n \leq_P m$.*
- *$\leq_P$ is reflexively and transitively closed.*

Definition 5.2.2 (Projection)**.** *Given a Process P and a set of keys C, the projection $P_{\star C}$ is defined as:*

$(\alpha_\gamma[n].P)_{\star C} = \alpha_\gamma[n].(P_{\star C})$ *if* $n \notin C$	$0_{\star C} = 0$	$(P \setminus A)_{\star C} = (P_{\star C}) \setminus A$
$(\alpha_\gamma[n].P)_{\star C} = \alpha_\gamma.(P_{\star C}\{{}^{\mathsf{roll}\ \gamma}/_{\mathsf{rolling}\ \gamma}\})$ *if* $n \in C$	$\mathsf{roll}\ \gamma_{\star C} = \mathsf{roll}\ \gamma$	$\mathsf{rolling}\ \gamma_{\star C} = \mathsf{rolling}\ \gamma$
$(P_0 \mid P_1)_{\star C} = P_{0_{\star C}} \mid P_{1_{\star C}}$	$A\langle \tilde{b}, \tilde{\gamma} \rangle_{\star C} = A\langle \tilde{b}, \tilde{\gamma} \rangle$	$(P[f])_{\star C} = P_{\star C}[f]$
$(P_0 + P_1)_{\star C} = P_{0_{\star C}} + P_{1_{\star C}}$	$(\alpha_\gamma.P)_{\star C} = \alpha_\gamma.(P_{\star C})$	

Much as in [58] we do our rollback in two steps, the first triggering the rollback, and the second actually performing the roll, in order to ensure that we can start multiple rollbacks at the same time. For example, in the process $(a_\gamma.(\overline{d}.0 \mid c.\mathsf{roll}\ \gamma) \mid b_{\gamma'}.(\overline{c} \mid d.\mathsf{roll}\ \gamma') \mid \overline{a} \mid \overline{b}) \setminus \{a, b, c, d\}$, similar to an example from [58], we will otherwise never be able to roll all the way back to the beginning, as rolling back a_γ will stop us from reaching $\mathsf{roll}\ \gamma'$ and vice versa. Tables 5.2 and 5.3 show the new rules for reversing actions in Roll-CCSK. The rules for forwards actions are still the same as in Table 2.2.

Example 5.2.3. *Consider the process*

$$(a_\gamma[n_a].(\overline{d}[n_d].0 \mid c[n_c].\mathsf{roll}\ \gamma) \mid b_{\gamma'}[n_b].(\overline{c}[n_c] \mid d[n_d].\mathsf{roll}\ \gamma') \mid \overline{a}[n_a] \mid \overline{b}[n_b]) \setminus \{a, b, c, d\}$$

We would like to roll this process back to its initial state, but performing one of the rolls would disable the other, since it would reverse the communications on c and d. We therefore say that we first need to trigger the rolls,

$$a_\gamma[n_a].(\overline{d}[n_d].0 \mid c[n_c].\mathsf{roll}\ \gamma) \mid b_{\gamma'}[n_b].(\overline{c}[n_c] \mid d[n_d].\mathsf{roll}\ \gamma') \mid \overline{a}[n_a] \mid \overline{b}[n_b]) \setminus \{a, b, c, d\}$$
$$\xrightarrow{\mathsf{start\ roll}\ \gamma\ \ \mathsf{start\ roll}\ \gamma'} (a_\gamma[n_a].(\overline{d}[n_d].0 \mid c[n_c].\mathsf{rolling}\ \gamma) \mid b_{\gamma'}[n_b].(\overline{c}[n_c] \mid d[n_d].\mathsf{rolling}\ \gamma') \mid \overline{a}[n_a] \mid \overline{b}[n_b]) \setminus \{a, b, c, d\}$$

meaning that the rolls can now be executed even if the preceding actions have been reversed, so we can do

$$a_\gamma[n_a].(\overline{d}[n_d].0 \mid c[n_c].\mathsf{rolling}\ \gamma) \mid b_{\gamma'}[n_b].(\overline{c}[n_c] \mid d[n_d].\mathsf{rolling}\ \gamma') \mid \overline{a}[n_a] \mid \overline{b}[n_b]) \setminus \{a, b, c, d\}$$
$$\xrightarrow{\mathsf{roll}\ n_b} (a_\gamma[n_a].(\overline{d}.0 \mid c.\mathsf{rolling}\ \gamma) \mid b_{\gamma'}.(\overline{c} \mid d.\mathsf{roll}\ \gamma') \mid \overline{a}[n_a] \mid \overline{b}) \setminus \{a, b, c, d\}$$

Since $\mathsf{rolling}\ \gamma$ has been triggered it can now be executed despite c having been reversed,

$$a_\gamma[n_a].(\overline{d}.0 \mid c.\mathsf{rolling}\ \gamma) \mid b_{\gamma'}.(\overline{c} \mid d.\mathsf{roll}\ \gamma') \mid \overline{a}[n_a] \mid \overline{b}) \setminus \{a, b, c, d\}$$
$$\xrightarrow{\mathsf{roll}\ n_a} (a_\gamma.(\overline{d}.0 \mid c.\mathsf{rolling}\ \gamma) \mid b_{\gamma'}.(\overline{c} \mid d.\mathsf{roll}\ \gamma') \mid \overline{a} \mid \overline{b}) \setminus \{a, b, c, d\}$$

$$\text{(start ROLL)}\quad \mathsf{roll}\ \gamma \overset{\mathsf{start\ roll}\ \gamma}{\rightsquigarrow} \mathsf{rolling}\ \gamma \qquad \text{(par ROLL)}\quad \frac{P_0 \overset{\mathsf{roll}\ n}{\rightsquigarrow} P_0' \quad C = \{m \mid n \leq_{P_0'|P_1} m\}}{P_0 \mid P_1 \overset{\mathsf{roll}\ n}{\rightsquigarrow} (P_0 \mid P_1)_{\star C}}$$

$$\text{(ROLL)}\quad \mathsf{rolling}\ \gamma \overset{\mathsf{roll}\ \gamma}{\rightsquigarrow} \mathsf{roll}\ \gamma \qquad \text{(act ROLL)}\quad \frac{P \overset{\mathsf{roll}\ \gamma}{\rightsquigarrow} P' \quad C = \{m \mid n \leq_{\alpha_\gamma[n].P} m\}}{\alpha_\gamma[n].P \overset{\mathsf{roll}\ n}{\rightsquigarrow} \alpha_\gamma.P'_{\star C}}$$

Table 5.2: The main rules for rollback in the operational semantics of Roll-CCSK

In addition, to ensure every rollback is associated with exactly one action, we define a *consistent process*. A consistent process is a strengthening of our notion of a reachable process, which we used in the previous section. We need this definition to prevent processes where a a roll has a tag which no action has, or the action with the tag is in parallel with the roll and therefore might not happen until after we start trying to execute the roll.

Definition 5.2.4 (Consistent process)**.** *A Roll-CCSK process P is consistent if*

1. *there exists a standard process Q with no subprocess* $\mathsf{rolling}\ \gamma$ *such that $Q \to^* P$;*
2. *any* $\mathsf{roll}\ \gamma$ *or* $\mathsf{rolling}\ \gamma$ *in P is part of a subprocess, $\alpha_\gamma.P'$ or $\alpha_\gamma[n].P'$;*
3. *any subprocess of P, $\alpha_\gamma.P'$ or $\alpha_\gamma[n].P'$ contains at most one instance of* $\mathsf{roll}\ \gamma$ *or* $\mathsf{rolling}\ \gamma$ *not part of a subprocess of P', $\alpha'_\gamma.P''$ or $\alpha'_\gamma[n].P''$;*
4. *if $A\langle \tilde{a}, \tilde{\gamma}\rangle$ is a subprocess of P defined as $A(\tilde{b}, \tilde{\delta}) = P_A$, then P_A is consistent.*

We show that consistent processes will remain consistent through their computations.

Proposition 5.2.5. *Let P be a consistent process and $P \equiv P'$. Then P' is consistent.*

Proof. The tags and definitions of process constants will not change between P and P'. □

Proposition 5.2.6. *Let P be a consistent process and $P \xrightarrow{l} P'$. Then P' is consistent.*

Proof. The tags and definitions of process constants will not change between P and P'. □

Example 5.2.7 (Recursion)**.** *Consider the process $a_\gamma.A\langle a;\gamma\rangle$ where $A(b;\delta) = b_\delta.(A\langle b;\delta\rangle \mid \mathsf{roll}\ \delta)$. This process is consistent and we show that we do not have propblems of confusing which actions are supposed to be reversed by which instance of* $\mathsf{roll}\ \gamma$*. After doing some actions and unfoldings we get*

$$a_\gamma[n_0].a_\gamma[n_1].(a_\gamma[n_2].(A\langle a;\gamma\rangle \mid \mathsf{roll}\ \gamma) \mid \mathsf{rolling}\ \gamma)$$

At this point, if we execute the outer $\mathsf{rolling}\ \gamma$ *we will get*

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{rolling}\ \gamma \overset{\mathsf{roll}\ \gamma}{\rightsquigarrow} \mathsf{roll}\ \gamma}{(a_\gamma[n_2].(A\langle a;\gamma\rangle \mid \mathsf{roll}\ \gamma) \mid \mathsf{rolling}\ \gamma) \overset{\mathsf{roll}\ \gamma}{\rightsquigarrow} (a_\gamma[n_2].(A\langle a;\gamma\rangle \mid \mathsf{roll}\ \gamma) \mid \mathsf{roll}\ \gamma)}}{a_\gamma[n_1].(a_\gamma[n_2].(A\langle a;\gamma\rangle \mid \mathsf{roll}\ \gamma) \mid \mathsf{rolling}\ \gamma) \overset{\mathsf{roll}\ n_1}{\rightsquigarrow} a_\gamma.(a_\gamma.(A\langle a;\gamma\rangle \mid \mathsf{roll}\ \gamma) \mid \mathsf{roll}\ \gamma)}}{a_\gamma[n_0].a_\gamma[n_1].(a_\gamma[n_2].(A\langle a;\gamma\rangle \mid \mathsf{roll}\ \gamma) \mid \mathsf{rolling}\ \gamma) \overset{\mathsf{roll}\ n_1}{\rightsquigarrow} a_\gamma[n_0].a_\gamma.(a_\gamma.(A\langle a;\gamma\rangle \mid \mathsf{roll}\ \gamma) \mid \mathsf{roll}\ \gamma)}$$

And we see that switching the roll from the tag to the key once we reach the first γ-tagged action means that is the action we roll back to.

	$\dfrac{P \overset{\text{roll } \gamma}{\rightsquigarrow} P' \quad \gamma \neq \gamma'}{\beta_{\gamma'}[m].P \overset{\text{roll } \gamma}{\rightsquigarrow} \beta_{\gamma'}[m].P'}$		$\dfrac{P \overset{\text{start roll } \gamma}{\rightsquigarrow} P'}{\alpha_{\gamma'}[n].P \overset{\text{start roll } \gamma}{\rightsquigarrow} \alpha_{\gamma'}[n].P'}$
(prop ROLL 1)		(prop ROLL start 1)	
(prop ROLL 2)	$\dfrac{P \overset{\text{roll } \gamma}{\rightsquigarrow} P'}{\beta_{\gamma'}.P \overset{\text{roll } \gamma}{\rightsquigarrow} \beta.\gamma'.P'}$	(prop ROLL start 2)	$\dfrac{P_0 \overset{\text{start roll } \gamma}{\rightsquigarrow} P_0' \quad \mathsf{std}(P_1)}{P_0 + P_1 \overset{\text{start roll } \gamma}{\rightsquigarrow} P_0' + P_1}$
(prop ROLL 3)	$\dfrac{P_0 \overset{\text{roll } \gamma}{\rightsquigarrow} P_0'}{P_0 + P_1 \overset{\text{roll } \gamma}{\rightsquigarrow} P_0' + P_1}$	(prop ROLL start 3)	$\dfrac{P_0 \overset{\text{start roll } \gamma}{\rightsquigarrow} P_0'}{P_0 \mid P_1 \overset{\text{start roll } \gamma}{\rightsquigarrow} P_0' \mid P_1}$
(prop ROLL 4)	$\dfrac{P \overset{\text{roll } \gamma}{\rightsquigarrow} P'}{P \setminus A \overset{\text{roll } \gamma}{\rightsquigarrow} P' \setminus A}$	(prop ROLL start 4)	$\dfrac{P \to P'}{P \setminus A \overset{\text{start roll } \gamma}{\rightsquigarrow} P' \setminus A}$
(prop ROLL 5)	$\dfrac{P \overset{\text{roll } \gamma}{\rightsquigarrow} P'}{P[f] \overset{\text{roll } \gamma}{\rightsquigarrow} P'[f]}$	(prop ROLL start 5)	$\dfrac{P \overset{\text{start roll } \gamma}{\rightsquigarrow} P'}{P[f] \overset{\text{start roll } \gamma}{\rightsquigarrow} P'[f]}$
(prop ROLL 6)	$\dfrac{P \equiv Q \overset{\text{roll } \gamma}{\rightsquigarrow} Q' \equiv P'}{P \overset{\text{roll } \gamma}{\rightsquigarrow} P'}$	(prop ROLL start 6)	$\dfrac{P \equiv Q \overset{\text{start roll } \gamma}{\rightsquigarrow} Q' \equiv P'}{P \overset{\text{start roll } \gamma}{\rightsquigarrow} P'}$
(prop ROLL Key 1)	$\dfrac{P \overset{\text{roll } n}{\rightsquigarrow} P' \quad n \neq m}{\beta_{\gamma}[m].P \overset{\text{roll } n}{\rightsquigarrow} \beta_{\gamma}[m].P'}$	(prop ROLL Key 2)	$\dfrac{P \overset{\text{roll } n}{\rightsquigarrow} P'}{\beta_{\gamma}.P \overset{\text{roll } n}{\rightsquigarrow} \beta.\gamma.P'}$
(prop ROLL Key 3)	$\dfrac{P_0 \overset{\text{roll } n}{\rightsquigarrow} P_0'}{P_0 + P_1 \overset{\text{roll } n}{\rightsquigarrow} P_0' + P_1}$	(prop ROLL Key 4)	$\dfrac{P \overset{\text{roll } n}{\rightsquigarrow} P'}{P \setminus A \overset{\text{roll } n}{\rightsquigarrow} P' \setminus A}$
(prop ROLL Key 5)	$\dfrac{P \overset{\text{roll } n}{\rightsquigarrow} P'}{P[f] \overset{\text{roll } n}{\rightsquigarrow} P'[f]}$	(prop ROLL Key 6)	$\dfrac{P \equiv Q \overset{\text{roll } n}{\rightsquigarrow} Q' \equiv P'}{P \overset{\text{roll } n}{\rightsquigarrow} P'}$

Table 5.3: The operational semantics for propagation of rolls in Roll-CCSK

We are then ready to prove Theorem 5.2.9, stating that for consistent subprocesses, any rollback can be undone by a sequence of forwards actions. For this we use Lemma 5.2.8, which states that projecting on an upwards-closed set of keys always results in a possible previous state of the process.

Lemma 5.2.8. *Given a consistent process P with no subprocess* rolling γ *and a set of keys C such that if $n \in C$ and $n \leq_P m$ then $m \in C$, we have a series of transitions $P_{\star C} \to* P$.*

Proof. We prove this by induction in the size of C in Appendix C.8. □

Theorem 5.2.9 (Loop (Soundness))**.** *Given consistent processes P_0 and P_1 containing no subprocesses* rolling γ*, such that $P_0 \overset{\text{start roll } \gamma}{\rightsquigarrow} P_0' \overset{\text{roll } n}{\rightsquigarrow} P_1$, we have $P_1 \to^* P_0$.*

Proof. Follows from Lemma 5.2.8. □

We now know that, like Roll-π [58], if an action is undone by our rollback, then any actions it caused are also reversed. We therefore know that we do not reverse too few actions in a roll. Next we need to determine that we do not reverse too many.

We will from now on use $\to_{CCSK}$ and $\rightsquigarrow_{CCSK}$ to distinguish CCSK-transitions from Roll-CCSK transitions. The last thing we need to prove about our rollback operational semantics before moving on to event structure semantics is Theorem 5.2.12, stating that (1) our rollbacks only reverse the actions caused by the action we are rolling back according to CCSK, and (2) our rollbacks are maximally permissive, meaning that any subset of reached rollbacks may be successfully executed.

Definition 5.2.10 (Transforming Roll-CCSK to CCSK)**.** *We define a function, ϕ, which translates a Roll-CCSK process into CCSK:*

$$\begin{array}{lll} \phi(0) = 0 & \phi(P_0 + P_1) = \phi(P_0) + \phi(P_1) & \phi(P_0 \mid P_1) = \phi(P_0) \mid \phi(P_1) \\ \phi(\alpha_\gamma.P) = \alpha.\phi(P) & \phi(\alpha_\gamma[n].P) = \alpha[n].\phi(P) & \phi(P \setminus A) = \phi(P) \setminus A \\ \phi(P[f]) = \phi(P)[f] & \phi(\text{roll } \gamma) = 0 & \end{array}$$

Definition 5.2.11. *Let P be a CCSK process and $T = \{m_0, m_1, \dots m_n\}$ be a set of keys. We say that $P \rightsquigarrow_T P'$ if there exists actions μ, ν and a tag m such that $P \overset{\mu[m]}{\rightsquigarrow}_{CCSK} P'$ and $\nu[m_i] \leq_P \mu[m]$ for some $m_i \in T$.*

Theorem 5.2.12 (Completeness)**.** *Let P be a consistent Roll-CCSK process with subprocesses with rolls, $\alpha_{0_{\gamma_0}}[m_0] \dots$ roll γ_0, $\alpha_{1_{\gamma_1}}[m_1] \dots$ roll γ_1, $\dots$, $\alpha_{n_{\gamma_n}}[m_n] \dots$ roll γ_n. Then for all $T \subseteq \{m_0, m_1, \dots m_n\}$, if $\phi(P) \rightsquigarrow_T^* P' \not\rightsquigarrow_T$ then there exists a Roll-CCSK process P'' such that $\phi(P'') = P'$ and $P \rightsquigarrow^* P''$*

Proof. We define P'' as the process we get by performing $\overset{\text{start roll } \gamma_i}{\rightsquigarrow}$ for every $m_i \in T$ and then $\overset{\text{roll } m_i}{\rightsquigarrow}$ for every $m_i \in T$ on P. We use Theorem 5.2.9 and Proposition 5.16 of [80] to show that this P'' fulfils the conditions described above. See Appendix C.9 for full details. □

We have now shown that not only do we not roll back more actions than we need in our roll, but if we have multiple rolls ready to be activated, we can choose to execute any subset of them. This means we avoid the problem described in the beginning of this section of the execution of one roll preventing the execution of another. With Theorems 5.2.9 and 5.2.12, we have shown that our semantics capture the behaviour of a rollback, but we would like to be able to roll back the actions one at a time in a distributed manner. For this purpose we define a new semantics, in which the roll marks the actions needing to be rolled back with the key associated with the rollback, so they can be rolled back individually. We refer to this as single-mark semantics and annotate its transitions with SM; we call the semantics described above high-level semantics, which we annotate with HL.

To mark the actions we define a new function, mark, similar to $\star$, which marks all the actions with a specific set of keys, C, with $\bullet$.

Definition 5.2.13. *Given a process P and a set of keys C, the marking $\text{mark}^\bullet_C(P)$ is defined as:*

$$\text{(start ROLL)}\quad \mathsf{roll}\ \gamma \overset{\mathsf{start\ roll}\ \gamma}{\rightsquigarrow}_{\mathrm{SM}} \mathsf{rolling}\ \gamma \qquad \text{(par ROLL)}\quad \frac{P_0 \overset{\mathsf{roll}\ n}{\rightsquigarrow}_{\mathrm{SM}} P_0' \quad C = \{m \mid n \leq_{P_0|P_1} m\}}{P_0 \mid P_1 \overset{\mathsf{roll}\ n}{\rightsquigarrow}_{\mathrm{SM}} \mathsf{mark}_C(P_0 \mid P_1)}$$

$$\text{(ROLL)}\quad \mathsf{rolling}\ \gamma \overset{\mathsf{roll}\ \gamma}{\rightsquigarrow}_{\mathrm{SM}} \mathsf{roll}\ \gamma \qquad \text{(act ROLL)}\quad \frac{P \overset{\mathsf{roll}\ \gamma}{\rightsquigarrow}_{\mathrm{SM}} P' \quad C = \{m \mid n \leq_{\alpha_\gamma[n].P} m\}}{\alpha_\gamma[n].P \overset{\mathsf{roll}\ n}{\rightsquigarrow}_{\mathrm{SM}} \alpha_\gamma[n]^\bullet.\mathsf{mark}_C(P')}$$

Table 5.4: The main rules for semi-distributed rollback of Roll-CCSK.

$$\frac{\mathsf{std}(P)}{\alpha[n]^\bullet.P \overset{\alpha[n]}{\rightsquigarrow}_{\mathrm{SM}} \alpha.P} \qquad \frac{P \overset{\mu[m]}{\rightsquigarrow}_{\mathrm{SM}} P' \quad m \neq n}{\alpha[n].P \overset{\mu[m]}{\rightsquigarrow}_{\mathrm{SM}} \alpha[n].P'} \qquad \frac{P \overset{\mu[m]}{\rightsquigarrow}_{\mathrm{SM}} P' \quad m \neq n}{\alpha[n]^\bullet.P \overset{\mu[m]}{\rightsquigarrow}_{\mathrm{SM}} \alpha[n]^\bullet.P'}$$

$$\frac{P \equiv Q \overset{\mu[n]}{\rightsquigarrow}_{\mathrm{SM}} Q' \equiv P'}{P \overset{\mu[n]}{\rightsquigarrow}_{\mathrm{SM}} P'} \qquad \frac{P_0 \overset{\mu[n]}{\rightsquigarrow}_{\mathrm{SM}} P_0' \quad \mathsf{fsh}[n](P_1)}{P_0 \mid P_1 \overset{\mu[n]}{\rightsquigarrow}_{\mathrm{SM}} P_0' \mid P_1} \qquad \frac{P_0 \overset{\alpha[n]}{\rightsquigarrow}_{\mathrm{SM}} P_0' \quad P_1 \overset{\overline{\alpha}[n]}{\rightsquigarrow}_{\mathrm{SM}} P_1'}{P_0 \mid P_1 \overset{\tau[n]}{\rightsquigarrow}_{\mathrm{SM}} P_0' \mid P_1'}$$

$$\frac{P_0 \overset{\mu[n]}{\rightsquigarrow}_{\mathrm{SM}} P_0' \quad \mathsf{std}(P_1)}{P_0 + P_1 \overset{\mu[n]}{\rightsquigarrow}_{\mathrm{SM}} P_0' + P_1} \qquad \frac{P \overset{\mu[n]}{\rightsquigarrow}_{\mathrm{SM}} P' \quad \mu, \overline{\mu} \notin A}{P \setminus A \overset{\mu[n]}{\rightsquigarrow}_{\mathrm{SM}} P' \setminus A} \qquad \frac{P \overset{\mu[n]}{\rightsquigarrow}_{\mathrm{SM}} P'}{P[f] \overset{f(\mu)[n]}{\rightsquigarrow}_{\mathrm{SM}} P'[f]}$$

Table 5.5: Reversing marked actions.

$$\begin{array}{ll}
\mathsf{mark}_C(\alpha_\gamma[m].P) = \alpha'_\gamma[m].\mathsf{mark}_C(P) \text{ if } m \notin C & \mathsf{mark}_C(0) = 0 \\
\mathsf{mark}_C(P \setminus A) = \mathsf{mark}_C(P) \setminus A & \mathsf{mark}_C(\mathsf{roll}\ \gamma) = \mathsf{roll}\ \gamma \\
\mathsf{mark}_C(\alpha_\gamma[m].P) = \alpha_\gamma[m]^\bullet.\mathsf{mark}_C(P) \text{ if } m \in C & \mathsf{mark}_C(\mathsf{rolling}\ \gamma) = \mathsf{rolling}\ \gamma \\
\mathsf{mark}_C(P_0 \mid P_1) = \mathsf{mark}_C(P_0) \mid \mathsf{mark}_C(P_1) & \mathsf{mark}_C(A\langle \tilde{b}, \tilde{\gamma} \rangle) = A\langle \tilde{b}, \tilde{\gamma} \rangle \\
\mathsf{mark}_C(P_0 + P_1) = \mathsf{mark}_C(P_0) + \mathsf{mark}_C(P_1) & \mathsf{mark}_C(\alpha_\gamma.P) = \alpha_\gamma.\mathsf{mark}_C(P) \\
\mathsf{mark}_C(P[f]) = \mathsf{mark}_C(P)[f] &
\end{array}$$

We call a process containing actions marked with $\bullet$ a marked process.

Note that this marking is not defined on already marked actions, meaning all markings must be removed before a new roll can start.

The forwards rules are still the same, though note that a forwards action cannot propagate past marked actions. The roll rules (Tables 5.4 and 5.3) are the same as in the previous semantics, with the exception of replacing instances of $P_{\star C}$ with $\mathsf{mark}_C(P)$ and letting start roll propagate past marked actions. For actually reversing the marked actions, we introduce the rules seen in Table 5.5, which invert the forwards rules, but must start with a marked action.

Example 5.2.14. *Consider the process $a_\gamma[n].b[m].\mathsf{rolling}\ \gamma \mid (\overline{b}[m] \mid \overline{a}[n])$ in which the a's and b's have communicated and the rollback has been activated. In order to actually execute the roll, we mark the actions needing to be rolled back with n:*

$$a_\gamma[n].b[m].\mathsf{rolling}\ \gamma \mid (\overline{b}[m] \mid \overline{a}[n]) \overset{\mathsf{roll}\ n}{\rightsquigarrow}_{\mathrm{SM}} a_\gamma[n]^\bullet.b[m]^\bullet.\mathsf{roll}\ \gamma \mid (\overline{b}[m]^\bullet \mid \overline{a}[n]^\bullet)$$

We can then reverse the marked actions in a causal way:

$$\begin{array}{ll}
 & a_\gamma[n]^\bullet.b[m]^\bullet.\mathsf{roll}\ \gamma \mid (\overline{b}[m]^\bullet \mid \overline{a}[n]^\bullet) \\
\xrightarrow{\tau[m]}_{\mathrm{SM}} & a_\gamma[n]^\bullet.b.\mathsf{roll}\ \gamma \mid (\overline{b} \mid \overline{a}[n]^\bullet) \\
\xrightarrow{\tau[n]}_{\mathrm{SM}} & a_\gamma.b.\mathsf{roll}\ \gamma \mid (\overline{b} \mid \overline{a})
\end{array}$$

We intend to show that these new semantics are equivalent to the previous. For this purpose we define a notion of *reverse-ignoring bisimulation* which, as the name suggests ignores all reverse transitions except rolls. This is useful for comparing our high-level and single-mark semantics, since the intention is that the only difference between the two semantics is the reverse actions which remove the keys from the marked actions.

Definition 5.2.15 (Reverse-ignoring Bisimulation)**.** *A relation $\mathcal{R}$ on processes from single-mark and high-level semantics, is a reverse-ignoring bisimulation if whenever $P\ \mathcal{R}\ Q$:*

1. *If $P \xrightarrow{\mu[n]}_{\text{SM}} P'$ then $Q \xleftarrow{\mu[n]}_{\text{HL}} Q'$ and $P'\mathcal{R}Q'$.*
2. *If $Q \xrightarrow{\mu[n]}_{\text{HL}} Q'$ then $P \xleftarrow{\mu[n]}_{\text{SM}} P'$ and $P'\mathcal{R}Q'$.*
3. *For $l \in \{\mathsf{roll}\ n \mid n \text{ is a key}\}$, if $P \overset{l}{\rightarrow}_{\text{SM}} P'$ then $Q \overset{l}{\hookleftarrow}_{\text{HL}} Q'$ and $P'\mathcal{R}Q'$.*
4. *For $l \in \{\mathsf{roll}\ n \mid n \text{ is a key}\}$, if $Q \overset{l}{\rightarrow}_{\text{HL}} Q'$ then $P \overset{l}{\hookleftarrow}_{\text{SM}} P'$ and $P'\mathcal{R}Q'$.*

Where $P \overset{l}{\hookleftarrow} P'$ if $P \overset{l_1}{\rightsquigarrow} \cdots \overset{l_i}{\rightsquigarrow} \overset{l}{\rightarrow} \overset{l'_1}{\rightsquigarrow} \cdots \overset{l'_j}{\rightsquigarrow} P'$ or $P \overset{l_1}{\rightsquigarrow} \cdots \overset{l_i}{\rightsquigarrow} \overset{l}{\rightarrow} \overset{l'_1}{\rightsquigarrow} \cdots \overset{l'_j}{\rightsquigarrow} P'$ for $\{l_1, \ldots, l_i, l'_1, \ldots, l'_j\} \cap (\{\mathsf{roll}\ n \mid n \text{ is a key}\}) = \emptyset$.

For processes P and Q, if there exists a reverse-ignoring bisimulation $\mathcal{R}$ such that $P\mathcal{R}Q$ then $P \approx_{\text{SM,HL}} Q$.

We also define a mapping, M, which removes the $\bullet$ and key from all marked actions, effectively reversing them, which we will use to describe our bisimulation relation between single-mark and high-level processes.

Definition 5.2.16 (Mapping from this semantic to high-level)**.** *Given a reachable process of the single-mark semantics, P, we define a mapping to a high-level process without markings:*

$$
\begin{array}{lll}
M(\alpha_\gamma[n].P) = \alpha_\gamma[n].M(P) & M(0) = 0 & M(P \setminus A) = M(P) \setminus A \\
M(\mathsf{roll}\ \gamma) = \mathsf{roll}\ \gamma & M(\mathsf{rolling}\ \gamma) = \mathsf{rolling}\ \gamma & M(P_0 \mid P_1) = M(P_0) \mid M(P_1) \\
M(A\langle \tilde{b}, \tilde{\gamma} \rangle) = A\langle \tilde{b}, \tilde{\gamma} \rangle & M(\alpha_\gamma[n]^\bullet.P) = \alpha_\gamma.M(P) & M(P_0 + P_1) = M(P_0) + M(P_1) \\
M((\alpha_\gamma.P) = \alpha_\gamma.M(P) & M(P[f]) = M(P)[f] &
\end{array}
$$

Using those, we get a correspondence between the two semantics, as expressed in Theorem 5.2.17.

Theorem 5.2.17. *Given a reachable process of the single-mark semantics, P, we have $P \approx_{SR,HL} M(P)$.*

Proof. We define

$$\mathcal{R} = \{(P, M(P)) \mid P \text{ is reachable } \}$$

and show that this is a reverse-ignoring bisimulation.

We do this by induction on the structure of P, and it is straightforward in all cases except $P = \alpha_\gamma[n]^\bullet.P'$. In this case P can only perform reverse non-roll actions $P \overset{\mu[m]}{\rightsquigarrow}_{\text{SM}} Q$, but $M(P)$ can perform $M(P) \xrightarrow{\alpha_\gamma[n']}_{HL} R$. Fortunately P can perform a series of transitions $P \overset{\mu_1[m_1]}{\rightsquigarrow}_{\text{SM}} \cdots \overset{\mu_i[m_i]}{\rightsquigarrow}_{\text{SM}} \overset{\alpha[n]}{\rightsquigarrow}_{\text{SM}} M(P) \xrightarrow{\alpha_\gamma[n']}_{\text{SM}} R$. □

This theorem ensures that the distributed rollback preserves the reachability of configurations yet does not make undesired configurations reachable. With our operational semantics of rollback now defined and proved to reverse the desired actions, it is time to add the rollback to our event structure semantics.

5.3 Event Structure Semantics of Roll-CCSK

Now that we have defined operational semantics of rollback, we are ready to describe a more formal method for using LREBESs to model rollback than the ad-hoc approach we used in Example 4.5.1.

To model a roll in an event structure, we have two events, one which triggers the roll, labelled $\mathsf{start\ roll}\ \gamma$, and another which starts the actual roll by allowing the events caused by the associated action to begin

reversing. When prefixing a process P with an action α_γ, we now need to ensure that any action in P, and any start roll associated with such an action, will be reversed by any roll γ in P, and that the rollback does not stop, signified by the event labelled roll γ being reversed, until those actions have all been reversed.

When composing the LREBESs of two processes, we also create a separate event for each set of causes it might have (Definition 5.3.1), similarly to how products of PESs are handled or how we define enablings when converting to an RES. This allows us to say that en event can be rolled back if it was caused by a communication with one of the events being rolled back, but not if the communication went differently. Consider the process $a_\gamma.\mathsf{roll}\ \gamma \mid \overline{a}.b \mid a_{\gamma'}.\mathsf{roll}\ \gamma'$. In this case we will want b to roll back if both (a_γ, a) and roll γ have happened, or if both $(a_{\gamma'}, a)$ and roll γ' have happened, but not if any other combination of the four events have happened, something which bundles cannot express unless b is split into multiple events. In addition, we use the sets of causes to ensure that if e is in e''s set of causes and e_{roll} can cause e to reverse, then e_{roll} can cause e' to reverse.

Definition 5.3.1. *Given an LREBES, $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$, the set of possible causes for an event $e \in E$, $\mathsf{cause}(e) = X$, contains minimal sets of events such that if $x \in X$ then:*

1. *if $x' \mapsto e$ then there exists e' such that $x' \cap x = \{e'\}$;*
2. *if $e' \in x$ then there exists $x' \in \mathsf{cause}(e')$ such that $X' \subseteq x$;*
3. *if $e_1, e_0 \in X$ then not $e_0 \sharp e_1$.*

Example 5.3.2 (Causes and parallel composition)**.** *Consider the process $P = a_\gamma.\mathsf{roll}\ \gamma \mid \overline{a}_{\gamma'}.\mathsf{roll}\ \gamma'$. This process would generate the event structure $[\![P]\!] = \langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \emptyset, \emptyset \rangle$ where:*

$$\begin{aligned} E = F = \quad & \{(\emptyset, (a, *)), (\emptyset(*, \overline{a})), (\emptyset, (a, \overline{a})), (\{(a, *)\}, (\mathsf{start\ roll}\ \gamma, *)), (\{(a, \overline{a})\}, (\mathsf{start\ roll}\ \gamma, *)), \\ & (\{(*, \overline{a})\}, (*, \mathsf{start\ roll}\ \gamma')), (\{(a, \overline{a})\}, (*, \mathsf{start\ roll}\ \gamma')), (\mathsf{roll}\ \gamma, *), (*, \mathsf{roll}\ \gamma')\} \end{aligned}$$

Here all events but the rolls get a set of causes associated with them.

The actions are not preceded by other actions, so their sets of causes are empty and they can happen at the start of the process.

The start roll *events are all caused by their preceding actions either happening on their own or synchronising, and whether they synchronise or not determines which of the* start roll *events happens.*

The roll *events do not have sets of causes, since we do not treat them as forwards actions needing to be reversed before the actions causing them are reversed.*

As a side effect of adding causes, we also need to change the definition of restriction to remove not only the actions associated with the restricted labels, but also the actions caused by them. We do this because we want the event structures generated by P and $0 \mid P$ to always be isomorphic, but if $P = (a.b) \setminus \{a\}$, we will otherwise get an event b, which, having no possible causes, disappears once we put P in parallel with anything, since this involves generating a b event for each set of possible causes.

Definition 5.3.3 (Removing labels and their dependants)**.** *Given an event structure $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$ and a set of labels $A \subseteq \mathsf{Act}$, we define $\rho(A) = X$ as the maximum subset of E such that*

1. *if $e \in X$ then $\lambda(e) \notin A$;*
2. *if $e \in X$ then there exists $x \in \mathsf{cause}(e)$ such that $x \subseteq X$.*

We are now ready to define event structure semantics of Roll-CCSK in Table 5.6. The processes roll γ and rolling γ, an unactivated and activated roll, generate the same event structure consisting of a start roll event e_s and a roll event e_r, but with the e_s in the initial state of rolling γ. A process with an action prefix, $\alpha_\gamma.P$ has to find any roll γ events rolling back to α_γ, R, and any events not part of a potential roll back to α_γ, E_{roll}. To define E_{roll}, we look for whether tags occur in P. This is not strictly necessary, as we can tell for a given roll event whether P contains its corresponding action by seeing whether it is caused by any action events (as we do when defining R). We then say that all non-roll events in P are caused by e_α and reversed by the rolls in R. And that the rolls in R are prevented from reversing by e_α. After adding causes to action events, a parallel composition, $P_0 \mid P_1$, has to figure out which rolls are associated with which of the new

events with causes. To do this, we need to find the rolls associated with either half of either the original event or one of its causes. The rest of the event structure semantics are similar to the ones in Section 5.1.

Much like we did in Proposition 5.1.13, we need to show that there exists a least upper bound of the event structures resulting from unfolding recursion. For this we first show that our action prefix, parallel composition, and tag binding are monotonic.

Lemma 5.3.4. *Given consistent processes P_0 and P_1 such that $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$, and $\mathcal{E}_0 \leq \mathcal{E}_1$, and there exists $A\langle\tilde{\alpha},\tilde{\gamma}\rangle$ in P_0 such that $A(\tilde{\beta},\tilde{\delta}) = P_A$ and $P_1 = P_0\{^{A\langle\tilde{\alpha},\tilde{\gamma}\rangle}/_{P_A\{^{\tilde{\alpha},\tilde{\gamma}}/_{\tilde{\beta},\tilde{\delta}}\}}\}$, and an action α_γ such that $\{\!|\alpha_\gamma.P_0|\!\} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$ and $\{\!|\alpha_\gamma.P_1|\!\} = \langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1\rangle$, we get $\mathcal{E}'_0 \leq \mathcal{E}'_1$.*

Proof. We prove this from Definition 5.1.7 in Appendix C.10. □

$\{\!|\mathsf{roll}\ \gamma|\!\}_l =$ $\langle(\{e_r,e_s\},\{e_r,e_s\},\mapsto,\rhd,\lambda,\mathsf{Act}),\emptyset,\emptyset\rangle$ where:
$\{e_r\} \mapsto \underline{e_r}$, $\{e_s\} \mapsto \underline{e_s}$ $\quad$ $\{e_s\} \mapsto e_r$, and $\{e_r\} \mapsto \underline{e_s}$ $\quad$ $e_s \rhd \underline{e_r}$ and $e_r \rhd e_s$
$\lambda(e) = \begin{cases} \mathsf{roll}\ \gamma & \text{if } e = e_r \\ \mathsf{start\ roll}\ \gamma & \text{if } e = e_s \end{cases}$ $\quad \mathsf{Act} = \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$

$\{\!|\mathsf{rolling}\ \gamma|\!\}_l =$ $\langle(\{e_r,e_s\},\{e_r,e_s\},\mapsto,\rhd,\lambda,\mathsf{Act}),\{e_s\},\emptyset\rangle$ where:
$\{e_r\} \mapsto \underline{e_r}$, $\{e_s\} \mapsto \underline{e_s}$ $\quad$ $\{e_s\} \mapsto e_r$, and $\{e_r\} \mapsto \underline{e_s}$ $\quad$ $e_s \rhd \underline{e_r}$ and $e_r \rhd e_s$
$\lambda(e) = \begin{cases} \mathsf{roll}\ \gamma & \text{if } e = e_r \\ \mathsf{start\ roll}\ \gamma & \text{if } e = e_s \end{cases}$ $\quad \mathsf{Act} = \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$

$\{\!|0|\!\}_l =$ $\langle(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset),\emptyset,\emptyset\rangle$

$\{\!|\alpha_\gamma.P|\!\}_l =$ $\langle(E,F,\mapsto,\rhd,\lambda,\mathsf{Act}),\mathsf{Init},k\rangle$ where:
$\{\!|P|\!\}_l = \langle(E_P,F_P,\mapsto_P,\rhd_P,\lambda_P,\mathsf{Act}_P),\mathsf{Init},k\rangle$
$E = E_P \cup \{e_\alpha\}$ where e_α fresh
$E_{\mathsf{roll}} = \left\{ e \;\middle|\; \begin{array}{l} \lambda_P(e) \in \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \text{ or} \\ \lambda_P(e) \in \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P\} \end{array} \right\}$
$F = F_P \cup \{e_\alpha\}$
$X \mapsto e$ if $X \mapsto_P e$ or $X = \{e_\alpha\}$, $e \in E_P$, and $\lambda_P(e) \neq \mathsf{roll}\ \gamma'$
We define the set of roll events rolling back to α_γ as:
$R = \{e \mid \lambda_P(e) = \mathsf{roll}\ \gamma \text{ and } e' \rhd_P \underline{e} \Rightarrow \lambda_P(e') = \mathsf{start\ roll}\ \gamma\}$
$X \mapsto \underline{e}$ if $X = \{e\}$, or $e = e_\alpha$ and $X = R$, or $e \in E_{\mathsf{roll}}$ and $X \mapsto_P \underline{e}$,
or $e \notin E_{\mathsf{roll}}$, $\{e\} \neq X' \mapsto_P \underline{e}$, and $X = X' \cup R$
$\rhd = \rhd_P \cup ((E \setminus \{e_r \mid \exists \gamma'.\lambda(e_r) \in \{\mathsf{roll}\ \gamma', \mathsf{start\ roll}\ \gamma'\}\}) \times \{\underline{e_\alpha}\}) \cup (\{e_\alpha\} \times \underline{R}) \cup (R \times (E \setminus R))$
$\mathsf{Act} = \mathsf{Act}_P \cup \{\alpha\}$
For all $e \in E$, $\lambda(e) = \begin{cases} \lambda_P(e) & \text{if } e \in E_P \\ \alpha & \text{if } e = e_\alpha \end{cases}$

$\{\!|\alpha_\gamma[m].P|\!\}_l =$ $\langle(E,F,\mapsto,\rhd,\lambda,\mathsf{Act}),\mathsf{Init},k\rangle$ where:
$\{\!|\alpha_\gamma.P|\!\}_l = \langle(E,F,\mapsto,\rhd,\lambda,\mathsf{Act}),\mathsf{Init}',k'\rangle$
$\{e_\alpha\} = \{e \in E \mid \lambda(a) = \alpha \text{ and } \nexists X \subseteq E.X \mapsto e_\alpha\}$
$\mathsf{Init} = \mathsf{Init}' \cup \{e_\alpha\}$ $\quad k(e) = \begin{cases} m & \text{if } e = e_\alpha \\ k'(e) & \text{otherwise} \end{cases}$

$\left\{\!\left|A\langle\tilde{b},\tilde{\delta}\rangle\right|\!\right\}_0 =$ $\langle(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset),\emptyset,\emptyset\rangle$

$\left\{\!\left|A\langle\tilde{b},\tilde{\delta}\rangle\right|\!\right\}_l =$ $\left\{\!\left|P_A\{^{\tilde{b},\tilde{\delta}}/_{\tilde{a},\tilde{\gamma}}\}\right|\!\right\}_{l-1}$ where $A\langle\tilde{a},\tilde{\gamma}\rangle = P_A$ and $l \geq 1$

$\{\!|P_0 + P_1|\!\}_l =$ $\langle\mathcal{E}_0 + \mathcal{E}_1, \mathsf{Init}, k\rangle$ where
For $i \in \{0,1\}$, $\{\!|P_i|\!\}_l = \langle\mathcal{E}_i, \mathsf{Init}_i, k_i\rangle$
$\mathsf{Init} = \{(j,e) \mid j \in \{0,1\} \text{ and } e \in \mathsf{Init}_j\}$

Table 5.6: LREBES-semantics of Roll-CCSK

$k((j,e)) = k_j(e)$ if $e \in \mathsf{Init}_j$

$\{\!|P_0 \mid P_1|\!\}_l =$ $\langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$ where:
$\{\!|P_i|\!\}_l = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i\rangle$ and $\mathcal{E}_i = (E_i, F_i, \mapsto_i, \rhd_i, \lambda_i, \mathsf{Act}_i)$ for $i \in \{0,1\}$
$(E_\times, F_\times, \mapsto_\times, \rhd_\times, \lambda_\times, \mathsf{Act}_\times) = \mathcal{E}_0 || \mathcal{E}_1$
$\mathsf{Init}_\times = \{(e_0,e_1) \mid e_0 \in \mathsf{Init}_0, e_1 \in \mathsf{Init}_1, k_0(e_0) = k_1(e_1)\} \cup$
$\{(*,e_1) \mid e_1 \in \mathsf{Init}_1, \nexists e_0 \in \mathsf{Init}_0. \lambda_0(e_0) = \underline{\lambda_1(e_1)}$, and $k_0(e_0) = k_1(e_1)\} \cup$
$\{(e_0,*) \mid e_0 \in \mathsf{Init}_0, \nexists e_1 \in \mathsf{Init}_1. \lambda_0(e_0) = \underline{\lambda_1(e_1)}$, and $k_0(e_0) = k_1(e_1)\}$
$E_{\mathsf{action}} = \left\{ (X,e) \middle| \begin{array}{l} e \in E_\times,\ \lambda_\times(e) \notin \{\mathsf{roll}\ \gamma \mid \gamma \text{ is a tag}\}, \\ X \in \mathsf{cause}(e), \text{ and } \forall e' \in X. \exists X' \in \mathsf{cause}(e'). X' \subseteq X \end{array} \right\}$
$E_{\mathsf{roll}} = \{e \mid e \in E_\times$ and $\lambda_\times(e) \in \{\mathsf{roll}\ \gamma \mid \gamma$ is a tag$\}\}$
$E = E_{\mathsf{action}} \cup E_{\mathsf{roll}}$
$F_{\mathsf{action}} = \{(X,e) \in E \mid e \in F_\times\} \qquad F_{\mathsf{roll}} = E_{\mathsf{roll}} \cap F_\times \qquad F = F_{\mathsf{action}} \cup F_{\mathsf{roll}}$
We define π_0 and π_1 such that for $(X,(e_0,e_1)) \in E_a$, $\pi_i((X,(e_0,e_1))) = e_i$, and
for $(e_0,e_1) \in E_r$, $\pi_i(e_0,e_1) = e_i$
$\{(X,e') \mid X' \subseteq X\} \mapsto (X',(e_0,e_1))$ if $e' \in X'$
$X \mapsto (e_0,e_1)$ if there exists X' such that $X' \mapsto_\times e$ and $X = \{e' \mid (\pi_0(e'),\pi_1(e')) \in X'\}$
$X \mapsto \underline{e}$ if $X = \{e\}$, or $e = (e_0,e_1)$ and $X = \{e' \mid (\pi_0(e'),\pi_1(e')) \in X'\}$ for some $X' \mapsto_\times \underline{e}$, or
$e = (X', e_\times)$ and $X = \bigcup \left\{ X'' \middle| \begin{array}{l} \exists i \in \{0,1\}, X_i \in E_i. X_i \mapsto \underline{\pi_i(e_\times)} \text{ or } \exists e'_\times \in X'. X_i \mapsto \underline{\pi_i(e'_\times)} \\ \text{and } e' \in X'' \text{ iff } \pi_i(e') \in X_i \end{array} \right\}$
$e \rhd e'^*$ if there exists $i \in \{0,1\}$ such that $\pi_i(e) \rhd_i \pi_i(e')^*$, or $e \neq e'$, and $e \in X \mapsto \underline{e'}$, or
$\pi_i(e) = \pi_i(e') \neq \bot$, and $e \neq e'$, $e'^* = e'$, or $e'^* \neq e'$ and $e, e' \in E_{\mathsf{roll}}$
$\mathsf{Act} = \mathsf{Act}_0 \cup \mathsf{Act}_1 \cup \{\tau\}$
$$\lambda(e) = \begin{cases} \tau & \text{if } e = (X,(e_0,e_1)) \\ \lambda_0(e_0) & \text{if } e = (X,(e_0,*)) \text{ or } e = (e_0,*) \\ \lambda_1(e_1) & \text{if } e = (X,(*,e_1)) \text{ or } e = (*,e_1) \end{cases}$$
$\mathsf{Init} = \{(X,e) \mid X \cup \{e\} \subseteq \mathsf{Init}_\times\} \cup (E_{\mathsf{roll}} \cap \mathsf{Init}_\times)$
$$k(e) = \begin{cases} k_0(e_0) & \text{if } e = (X,(e_0,*)) \\ k_1(e_1) & \text{if } e = (X,(*,e_1)) \\ k_0(e_0) & \text{if } e = (X,(e_0,e_1)) \quad \textit{– note that } k_0(e_0) = k_1(e_1) \end{cases}$$

$\{\!|P \setminus A|\!\}_l =$ $\langle \mathcal{E} \upharpoonright \rho(A \cup \overline{A}), \mathsf{Init} \cap \rho(A \cup \overline{A}), k \upharpoonright \rho(A \cup \overline{A})\rangle$ where $\{\!|P|\!\}_l = \langle \mathcal{E}, \mathsf{Init}, k\rangle$

Table 5.6: LREBES-semantics of Roll-CCSK (continued)

Lemma 5.3.5. *Given consistent processes P_0 and P_1 such that $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$, and $\mathcal{E}_0 \leq \mathcal{E}_1$, and an action α_γ such that $\left\{\!\left|\alpha_\gamma[m].P_0\right|\!\right\} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$ and $\left\{\!\left|\alpha_\gamma[m].P_1\right|\!\right\} = \langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1\rangle$, we get $\mathcal{E}'_0 \leq \mathcal{E}'_1$.*

Proof. Follows from Lemma 5.3.4 and the definitions of $\left\{\!\left|\alpha_\gamma[m].P_0\right|\!\right\}$ and $\left\{\!\left|\alpha_\gamma[m].P_1\right|\!\right\}$. □

Lemma 5.3.6. *Given consistent processes $P_0 \mid P_2$, $P_1 \mid P_2$ such that $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$, $\{\!|P_0 \mid P_2|\!\} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$, $\{\!|P_1 \mid P_2|\!\} = \langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1\rangle$, and $\mathcal{E}_0 \leq \mathcal{E}_1$, we get $\mathcal{E}'_0 \leq \mathcal{E}'_1$.*

Proof. We prove this from Definition 5.1.7 in C.11. □

We can then use these lemmas to prove that we have a least upper bound of unfoldings.

Proposition 5.3.7 (Unfolding)**.** *Given a consistent process P and a level of unfolding k, if $\{\!|P|\!\}_k = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ and $\{\!|P|\!\}_{k-1} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$, then $\mathcal{E}' \leq \mathcal{E}$, $\mathsf{Init} = \mathsf{Init}'$, and $k = k'$.*

Proof. Follows from Lemmas 5.3.4, 5.3.5, and 5.3.6 and Proposition 5.1.13. □

In Example 5.3.8 we show how our semantics deal with recursive processes with rollback. We unfold a recursion twice and show that even though the roll in every level of unfolding has the same tag, they are only associated with the most recent action a_γ and so the later rolls do not roll back the whole process. This is because of our semantics of actions, which are only caused to reverse by later rolls if those rolls are not already associated with an action.

Example 5.3.8 (Recursion). *Consider the process $P = a_\gamma.A\langle a;\gamma\rangle$ with the recursive definition $A(b;\delta) = b_\delta.(A\langle b;\delta\rangle \mid \mathsf{roll}\ \delta)$ from Example 5.2.7. Since we have recursion $\{\!|P|\!\}$ would have an infinite number of events, but we can still get a reasonable idea of what it would look like by only unfolding twice, giving us $\{\!|P|\!\}_2 = \langle(E, F, \mapsto, \rhd), \emptyset, \emptyset\rangle$. We name events after their labels and their level of unfolding so that e.g. $\mathsf{roll}\ \gamma{:}1$ is the $\mathsf{roll}\ \gamma$-labelled event originating from the first unfolding of the recursion.*

$$
\begin{array}{ll}
E = F = \{a{:}0, a{:}1, a{:}2, \mathsf{start\ roll}\ \gamma{:}1, \mathsf{start\ roll}\ \gamma{:}2, \mathsf{roll}\ \gamma{:}1, \mathsf{roll}\ \gamma{:}2\} & \\
\{a{:}i\} \mapsto a{:}j \text{ for } i < j & \mathsf{roll}\ \gamma{:}i \rhd a{:}j \text{ for } i \leq j \\
\{a{:}i\} \mapsto \mathsf{start\ roll}\ \gamma{:}j \text{ for } i \leq j & \mathsf{roll}\ \gamma{:}i \rhd \mathsf{start\ roll}\ \gamma{:}j \text{ for } i \leq j \\
\{\mathsf{start\ roll}\ \gamma{:}i\} \mapsto \mathsf{roll}\ \gamma{:}i \text{ for } i \in \{1,2\} & \mathsf{roll}\ \gamma{:}i \rhd \mathsf{roll}\ \gamma{:}j \text{ for } i \neq j \\
\{\mathsf{roll}\ \gamma{:}i\} \mapsto \underline{\mathsf{start\ roll}\ \gamma{:}i} \text{ for } i \in \{1,2\} & a{:}i \rhd \underline{a{:}j} \text{ for } i > j \\
\{\mathsf{roll}\ \gamma{:}i\} \mapsto \underline{a{:}j} \text{ for } i \leq j & a{:}i \rhd \underline{\mathsf{roll}\ \gamma{:}i} \text{ for } i \in \{0,1\} \\
\lambda(\mu{:}i) = \mu & \mathsf{start\ roll}\ \gamma{:}i \rhd \underline{\mathsf{roll}\ \gamma{:}i} \text{ for } i \in \{1,2\} \\
\mathsf{Act} = \{a, \mathsf{start\ roll}\ \gamma\} &
\end{array}
$$

Obviously this can be extended to a greater level of unfolding by allowing $i > 2$.

In this case we deal with the issue of ensuring each rollback only rolls back to its most recent a_γ by using the set $\{e_r \mid \lambda_P(e_r) = \mathsf{roll}\ \gamma \text{ and } e_r \in X \mapsto e \Rightarrow \lambda(e) = \mathsf{start\ roll}\ \gamma\}$ as the set of rollback events rolling back to α_γ in the definition of $\{\!|\alpha_\gamma.P|\!\}$. This ensures that if the rollback event has found another γ-tagged action in P then it will be causing that to reverse and therefore not be in the set.

We then show that structurally congruent processes result in isomorphic event structures, for which we use Lemmas 5.3.9 to 5.3.14.

We first show that aside from rolls, we have causal consistency between events.

Lemma 5.3.9 (Causal consistency of actions). *Given a roll-CCSK process P, where $\{\!|P|\!\} = \langle\mathcal{E}, \mathsf{Init}, k\rangle$, for any events e, e', if $e \in X \mapsto e'$ and $e' \not\rhd \underline{e}$, then there exists γ such that $\lambda(e') \in \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$.*

Proof. We prove this by structural induction on P in Appendix C.12. □

We then show that, aside from rolls, causation is transitive.

Lemma 5.3.10 (Transitive causation). *Given a process P, where $\{\!|P|\!\} = \langle\mathcal{E}, \mathsf{Init}, k\rangle$, whenever $X \mapsto e \in X' \mapsto e'$, we have $X \mapsto e'$, or there exists a γ such that $\lambda(e') \in \{\mathsf{roll}\ \gamma \mid \gamma \text{ is a tag}\}$.*

Proof. We prove this by structural induction on P in Appendix C.13. □

We then show that because we split actions into multiple events, each bundle associated with an action event only contains one event.

Lemma 5.3.11 (forwards bundles). *Given a process P such that $\{\!|P|\!\} = \langle(E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$, whenever $X \mapsto e$, either there exists e' such that $X = \{e'\}$, or there exists a γ such that $\lambda(e) = \mathsf{roll}\ \gamma$.*

Proof. We Prove this by structural induction on P in Appendix C.14. □

We then show a sort of reverse causality. If an event e' causes e to reverse, then e' prevents e. This means actions being reversed as part of a roll must wait for the roll to be completed.

Lemma 5.3.12 (Reverse inverse causality). *Given a process P such that $\{\!|P|\!\} = \langle(E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$, whenever $e' \in X \mapsto \underline{e}$ and $e \neq e'$, we get $e' \rhd e$.*

Proof. We prove this by structural induction on P in Appendix C.15. □

We then show that the reversal of an event is associated with at most one bundle. This means no action needs multiple rolls to be in progress before it can reverse.

Lemma 5.3.13 (Single backwards bundle)**.** *Given a process P such that $\{\!|P|\!\} = \langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$, for any event $e \in F$, there exists at most one bundle $X \mapsto \underline{e}$ such that $X \neq \{e\}$.*

Proof. We prove this by structural induction on P in Appendix C.16. □

The final lemma we need for structural congruence says that if e' causes an action event e then every roll that reverses e' must also reverse e.

Lemma 5.3.14 (Reverse transitivity)**.** *Given a process P such that $\{\!|P|\!\} = \langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$, whenever $e' \in X \mapsto e$, $X' \mapsto \underline{e'}$, $X' \neq \{e'\}$, and $\lambda(e) = \mu$, there must exist $X'' \supseteq X'$ such that $X'' \mapsto \underline{e}$.*

Proof. We prove this by structural induction on P in Appendix C.17. □

Having introduced these lemmas, we are ready to prove that structurally congruent processes generate isomorphic event structures. The sets of causes makes this the most complex part of the proof of operational correspondence, unlike in Section 5.1 where the proof of Proposition 5.1.15 was fairly simple.

Proposition 5.3.15 (Structural Congruence)**.** *Given consistent roll-CCSK-processes P and P', if $P \equiv P'$, $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, and $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$, then there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $f(\mathsf{Init}) = \mathsf{Init}'$ and for all $e \in \mathsf{Init}$, $k(e) = k'(f(e))$.*

Proof. We prove this by case analysis on structural congruence in Appendix C.18. We define the morphisms f and g by induction on P and P'. This is simple in most cases, but the case for $P = P_0 \mid (P_1 \mid P_2)$ and $P' = (P_0 \mid P_1) \mid P_2$ requires us to split and reassemble the sets of causes of P_0, P_1, and P_2 since the events of P are structured as $(X, (e_0, (X', (e_1, e_2))))$ while the events of P' are structured as $(Y, ((Y', (e_0, e_1)), e_2))$. □

We then prove, much like we did for our CCSK event structure semantics, that, as stated in Theorems 5.3.16 and 5.3.17, process P has a transition $P \xrightarrow{\mu} P'$ if and only if P and P' correspond to isomorphic event structures, and there exists a μ-labelled transition from the initial state of P's event structure to the initial state of P''s event structure.

Theorem 5.3.16. *Let P be a consistent Roll-CCSK process. If $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle_I$, $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$, $C_{re}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, Init is conflict-free, and there exists a transition $P \xrightarrow{\mu[m]} P'$ such that $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle_I$, then there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ and a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

Proof. We prove this by structural induction on $P \xrightarrow{\mu[m]} P'$ in Appendix C.19 using Lemmas 5.3.10, 5.3.12, and 5.3.14 and Proposition 5.3.15. We describe the morphisms f and g in each case and find e. f and g can be defined by simple induction on P, and in most cases it is straightforward to define e and prove the remaining conditions hold. Unlike in Theorem 5.1.18, in the parallel composition case, our events now have a set of causes which need to be considered when defining our isomorphisms. This makes it more difficult to prove that f and g fulfil conditions 2 and 3 of Definition 4.5.3 of REBES morphisms. □

Having shown that each forwards transition in the operational semantics corresponds to one in the generated event structure, we now show the inverse.

Theorem 5.3.17. *Let P be a consistent Roll-CCSK process. If $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$, $C_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, Init is conflict-free, and there exists a transition $\mathsf{Init} \xrightarrow{e} X$ in $C_{br}(\mathcal{E})$ such that $\lambda(e) = \mu$, then there exists a key m and a transition $P \xrightarrow{\mu[m]} P'$, such that $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$ and there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ such that $f \circ k' = k[e \mapsto m]$ and $f(X) = \mathsf{Init}'$.*

Proof. We prove this by structural induction on P in Appendix C.20 using Lemma 5.3.9. As in Theorem 5.1.19, we define f and g similarly to Theorem 5.3.16 and use a fresh m to prove that the transition $P \xrightarrow{\mu[m]} P'$ exists. □

We have now proved that the changes to the event structure semantics to accommodate rollback do not prevent the operational correspondence with the forwards semantics. We then prove the same correspondence for start roll transitions.

Proposition 5.3.18. *Let P be a consistent Roll-CCSK process. If $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$, $C_{re}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, Init is conflict-free, and there exists a transition $P \xrightarrow{\mathsf{start\ roll}\ \gamma} P'$ such that $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$, then there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ and a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ such that $\lambda(e) = \mathsf{start\ roll}\ \gamma$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

Proof. Similar to proof of Theorem 5.3.16. □

Proposition 5.3.19. *Let P be a consistent Roll-CCSK process. If $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$, $C_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, Init is conflict-free, and there exists a transition $\mathsf{Init} \xrightarrow{e} X$ in $C_{br}(\mathcal{E})$ such that $\lambda(e) = \mathsf{start\ roll}\ \gamma$, then there exists a a key m and a transition $P \xrightarrow{\mathsf{start\ roll}\ \gamma} P'$, such that $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$ and there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ such that $f \circ k' = k[e \mapsto m]$ and $f(X) = \mathsf{Init}'$.*

Proof. Similar to proof of Theorem 5.3.17. □

We finally need to prove that P can do a roll γ transition if and only if the event structure generated by P can do a roll γ-labelled event followed by reversing all the events corresponding to actions and start roll's with tags causally dependent on γ and then undoing roll γ. For this we need Lemmas 5.3.20 to 5.3.24.

Lemma 5.3.20 states that all rolls are in conflict.

Lemma 5.3.20. *Let P be a Roll-CCSK process. If $\{\!|P|\!\} = \langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$ where $e, e' \in E$, $e \neq e'$, and there exist γ, γ' such that $\lambda(e) \in \{\mathsf{roll}\ \gamma \mid \gamma \text{ is a tag}\}$ and $\lambda(e') \in \{\mathsf{roll}\ \gamma \mid \gamma \text{ is a tag}\}$, then $e \rhd e'$.*

Proof. We see from the syntax that e and e' come from parallel subprocesses, and the result follows from the parallel composition rule. □

Lemma 5.3.21 states that reversal of actions and roll initiations are only caused by rolls.

Lemma 5.3.21. *Let P be a Roll-CCSK process. If $\{\!|P|\!\} = \langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$ and $e \in E$ where $\lambda(e) \notin \{\mathsf{roll}\ \gamma \mid \gamma \text{ is a tag}\}$. Then whenever $X \mapsto \underline{e}$, either $X = \{e\}$ or for all $e' \in X$, $\lambda(e') \notin \{\mathsf{roll}\ \gamma \mid \gamma \text{ is a tag}\}$.*

Proof. We prove this by structural induction on P. Most cases are straightforward. In parallel composition we use the fact that we never have $e'' \in X'' \mapsto e'''$ where $\lambda(e'') \in \{\mathsf{roll}\ \gamma \mid \gamma \text{ is a tag}\}$. □

Lemma 5.3.22 states that if an event e' in the initial state of a process prevents another event e in the initial state from reversing, then the key of e must have caused the key of e'.

Lemma 5.3.22. *Let P be a Roll-CCSK process. If $\{\!|P|\!\} = \langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$ and $e, e' \in \mathsf{Init}$ then $e' \rhd \underline{e}$ if and only if $k(e) \leq_P k(e')$.*

Proof. We prove this by induction on P. It is trivial in all cases except $P = \alpha_{\gamma''}[n].P'$ and $P = P_0 \mid P_1$.

- Suppose $P = \alpha_{\gamma''}[n].P'$. Then if $e' \rhd \underline{e}$ either $e = e_\alpha$ or the result follows from induction and the fact that $e \in \mathsf{Init}$ means $\lambda(e) \notin \{\mathsf{roll}\ \gamma'' \mid \gamma'' \text{ is a tag}\}$. If $e = e_\alpha$ then $e' \rhd \underline{e}$ unless $\lambda(e') \in \mathsf{start\ roll}\ \gamma', \mathsf{roll}\ \gamma$ for some γ.
- Suppose $P = P_0 \mid P_1$. Then there exists $i \in \{0, 1\}$ such that either $\pi_i(e') \rhd_i \underline{\pi_i(e)}$ or $\pi_i(e') = \pi_i(e)$. If $\pi_i(e') \rhd_i \underline{\pi_i(e)}$ then the result follows from induction, and if $\pi_i(e') = \pi_i(e)$ then that contradicts $e, e' \in \mathsf{Init}$. □

We now need a function, $\mathsf{N}(e)$, to give us the key an event e labelled with an initiated rollback roll γ is rolling back to in Definition 5.3.23. This is necessary because the event is labelled roll γ but when executing the roll, the transition is labelled roll n and to describe our operational correspondence, we need to show that these are the same roll. We prove that N finds such a key in Lemma 5.3.24.

Definition 5.3.23 (N). *Let P be a Roll-CCSK process with $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ and $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$. We define a partial function N on events such that for an event $e \in E$, $\mathsf{N}(e) = \mathsf{roll}\ n$ if (1) $\lambda(e) = \mathsf{roll}\ \gamma$, (2) $\mathsf{Init} \xrightarrow{e}$, (3) for any key m, we have $m \geq_P n$ if and only if there exists an event $e' \in \mathsf{Init}$ such that $k(e') = m$ and $X \subseteq E$ such that $X \mapsto \underline{e'}$ and $e \in X$. Otherwise $\mathsf{N}(e)$ is undefined.*

Lemma 5.3.24. *Let P be a consistent forwards-reachable roll-CCSK process such that $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ and $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$, then:*

1. *Let $e \in E$ be an event such that $\lambda(e) = \mathsf{roll}\ \gamma$ and $\mathsf{Init} \xrightarrow{e}$. Then $\mathsf{N}(e)$ is defined and unique.*
2. *Let $P \xrightarrow{\mathsf{roll}\ n}$. Then there exists a unique $e \in E$ such that $\mathsf{N}(e) = \mathsf{roll}\ n$.*

Proof.

1. By Theorems 5.3.16 and 5.3.17 any key m occurs in P if and only if it occurs in $k(\mathsf{Init})$ and given $e, e' \in \mathsf{Init}$, $e \in X \mapsto e'$ if and only if $k(e) <_P k(e')$. The rest follows from Lemma 5.3.14.
2. Since P is consistent and forwards-reachable, exactly one $\alpha_\gamma[n]$ must occur somewhere in P and by Theorems 5.3.16 and 5.3.17, we have $e' \in E$ such that $\lambda(e') = \alpha$ and $k(e') = n$. Then we say that e is the event such that $\lambda(e) = \mathsf{roll}\ \gamma$ and by similar logic to the first part, $\mathsf{N}(e) = \lambda(e) = \mathsf{roll}\ n$. To prove uniqueness, we only need to show that in the subprocess $\alpha_\gamma[n].P'$ of P, no other such e' exists. This follows from the definition of R in $\left\{\!\left|\alpha_\gamma[n].P'\right|\!\right\}$ in Table 5.6 and the consistency of P. □

With these lemmas and definitions introduced, we will now prove Theorems 5.3.25 and 5.3.26. These state that a process P can do $\mathsf{roll}\ \gamma$ if and only if the REBES generated by P can do a $\mathsf{roll}\ \gamma$ event, followed by undoing all the events corresponding to actions P rolled back, and then reverse the $\mathsf{roll}\ \gamma$ event.

Theorem 5.3.25. *Let P be a Roll-CCSK process such that $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$. Whenever $P \overset{\mathsf{roll}\ n}{\rightsquigarrow} P'$, there exist e and $e_0, e_1, \ldots e_n$ such that $\mathsf{Init} \xrightarrow{\{e\}} X_0 \xrightarrow{\{\underline{e_0}\}} X_1 \cdots \xrightarrow{\{\underline{e_n}\}} X_{n+1} \xrightarrow{\{\underline{e}\}} X_{\mathsf{done}}$, $\mathsf{N}(e) = \mathsf{roll}\ n$, $\{e_0, e_1, \ldots e_n\} = \{e' \mid n \leq_P k(e')\}$, $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}, k\rangle$, and there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $f(X_{\mathsf{done}}) = \mathsf{Init}'$.*

Proof. We prove this by induction on the derivation of $P \overset{\mathsf{roll}\ n}{\rightsquigarrow} P'$ in Appendix C.21. We define f and g inductively on P as in previous theorems and prove that they satisfy the conditions using Lemmas 5.3.9, 5.3.13, 5.3.14, 5.3.20, 5.3.21, and 5.3.22, Proposition 5.3.15, and Theorem 5.3.16. □

Having shown that each roll transition in the operational semantics corresponds to a sequence of transitions in the generated event structure, we now show the inverse.

Theorem 5.3.26. *Let P be a consistent Roll-CCSK process such that $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$. Whenever $\mathsf{Init} \xrightarrow{\{e_r\}} X_0 \xrightarrow{\{\underline{e_0}\}} X_1 \cdots \xrightarrow{\{\underline{e_n}\}} X_{n+1} \xrightarrow{\{\underline{e_r}\}} X_{\mathsf{done}}$, we have a transition $P \overset{\mathsf{roll}\ n}{\rightsquigarrow} P'$, such that $\mathsf{N}(e) = \mathsf{roll}\ n$, $\{e_0, e_1, \ldots e_n\} = \{e' \mid n \leq_P k(e')\}$, $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}, k\rangle$, and there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $f(X_{\mathsf{done}}) = \mathsf{Init}'$.*

Proof. Follows from Theorem 5.3.16 and Proposition 5.3.15 □

We have now proved an operational correspondence between the operational semantics presented in Section 5.2 and the denotational event structure semantics presented in this section. In particular, we have shown that using non-causal event structures lets us model a control operator. Since any usable reversible programming language needs a way to control reversibility, and rollback is a common way to do this, having true concurrency semantics of rollback is important for reasoning about reversible programming languages. In addition, there is nothing about our approach to modelling rollback that is specific to CCSK, so we have reason to believe a similar approach could be used to add rollback to other denotational reversible event structure semantics and potentially event structures generated from different sources. This would also enable us to add rollback directly to the event structure semantics without defining operational rollback semantics, and proving Theorems 5.2.9 and 5.2.12 directly on the event structures.

5.4 Conclusion

We have described denotational true concurrency semantics of CCSK [80]. Unlike previous efforts to describe truly concurrent semantics of a reversible process calculus [4,34], we have generated both the event structure and the initial state directly from the process, rather than needing to first undo previous actions to get the original process and from there the event structure, and then redo the actions to get the initial state. This was made easier by CCSK using static reversibility, since we did not have to combine events generated separately from a memory and a process. Our event structure semantics has shown that the same process at different points in its execution will generate isomorphic event structures with different initial states (Theorems 5.1.18 and 5.1.19). This is a key difference between our semantics and previous event structure semantics of forward-only process calculi [21, 22, 31, 32, 64, 66, 94] where the event structure will get smaller after a transition.

By extending CCSK with the rollback operator of [58] and proving it works as in Roll-π (Theorems 5.2.9 and 5.2.12), we have created Roll-CCSK, the first statically reversible calculus with controlled reversibility. We defined both high-level semantics where the entire roll is executed at once and distributed semantics where the keys are marked for rollback and then are reversed one by one. A similar approach of marking keys could likely be used to add rollback to the statically reversible π-calculus [67]. As well as defining operational semantics of Roll-CCSK, we have used the *non-causal* reversible extended bundle event structures to define the event structure semantics of Roll-CCSK. Using event structures with non-causal reversibility allows us to treat rolls as normal events where the process $a_\gamma.\mathsf{roll}\ \gamma$ has the event a_γ cause the event $\mathsf{roll}\ \gamma$, and at the same time a_γ prevents $\mathsf{roll}\ \gamma$ from reversing. These semantics also create the isomorphic event structures with different initial states for different points in the execution of the same process (Theorems 5.3.16, 5.3.17, 5.3.25, and 5.3.26).

Another common way to control reversibility is by commit or irreversible actions, introduced for reversible CCS in [36]. These are used to denote a safe state which the process cannot reverse past. As such they function as a dual of the rollback, which ensures the process will reverse when hitting a fail state. Irreversible actions are simple to add to event structure semantics of CCSK by making the corresponding events irreversible, but would be more challenging to add to Roll-CCSK due to potentially having events which are irreversible but also required to reverse in order to finish a roll. We chose to focus on rollback since, as pointed out in [58], rollback gives the programmer more control than commit and more closely models system recovery techniques.

The key result of this chapter is that we are able to define semantics of the first controlled statically reversible process calculus and describe true concurrency semantics of a reversible process with and without a control operator.

Chapter 6

Event structures for the reversible early internal Pi-calculus

The π-calculus [74, 75] is a widely used process calculus, which models communications between processes using input and output actions, and allows the passing of communication links. There are three versions of the operational semantics of the π-calculus, called *reduction*, *late*, and *early*. The early version allows a process to receive (free) names it already knows from the environment, while the others do not. All existing reversible versions of the π-calculus use reduction [59, 89] or late [33, 67] semantics. However the early semantics of the (forward-only) π-calculus is more widely used than the late, partly because it has a sound correspondence with contextual congruences [53, 76].

We define πIH and πIK, the first reversible early π-calculi, and give them semantics in terms of reversible event structures. The new calculi are a reversible form of the internal π-calculus, or πI-calculus [85], which is a subset of the π-calculus where every link sent by an output is bound (private), yielding greater symmetry between inputs and outputs. It has been shown that the asynchronous π-calculus can be encoded in the asynchronous form of the πI-calculus [17].

Although πI-calculus is less expressive than the full π-calculus, not allowing processes to send the same name more than once or send a name after receiving it, it can still model kinds of the behaviours the π-calculus was designed for. Consider Example 6.0.1 adapted from [88], which can be modelled by the πI-calculus. In a reversible setting, instead of having a choice of whether they want to pay the price, the buyer can at any point decide to reverse their actions to before the transaction began. The seller can do the same if e.g. they do not have the requested product or do not ship to the given address. We describe this example in the syntax of the πI-calculus in Example 6.2.2.

Example 6.0.1. *A seller waits to receive a channel on which a buyer wants to communicate, which it will then use to receive the name of the product the buyer wants to buy. The seller then replies with the price of the product and the buyer replies with the address the product is to be sent to.*

Our two calculi have different approaches to reversibility. Our πIK is inspired by CCSK and the statically reversible π-calculus of [67], which use communication keys to denote past actions. To keep track of link causation, keys are used in a number of different ways in [67]. In our case we can handle link causation by using keys purely to annotate the action which was performed using the key, and any names which were substituted during that action.

We base πIH on the work of Hildebrandt *et al.* [52], which used extrusion histories and locations to define a stable non-interleaving early operational semantics for the π-calculus. These extrusion histories consist of two sets, $(\overline{H}, \underline{H})$ where $\overline{H}$ contains the locations and restricted names of restrictions which have been opened, and $\underline{H}$ contains the locations and received names of input actions which have received names from the environment. Locations here consist of the path taken through parallel compositions to reach the action and what the subprocess at the end of this path looks like before and after the action is takes. These extrusion histories and locations allow the authors to define causal semantics of the early π-calculus, where a transition $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u,D]{\alpha} (\overline{H'}, \underline{H'}) \vdash P'$ is labelled with an action α a location u where α took place, and

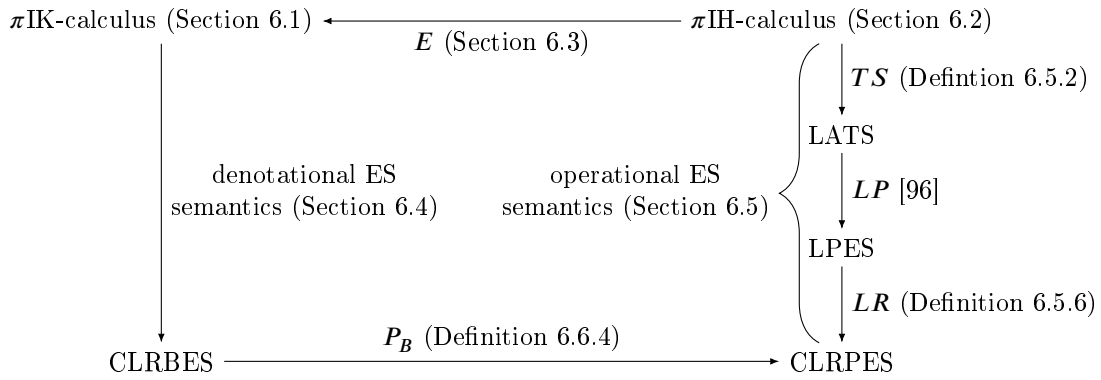


Figure 6.1: The reversible πI-calculi and event structures and mappings between them. The mapping TS generates a labelled transition system, LP maps from there to a prime event structure, which LR maps to a causal reversible prime event structure. The mapping P_B is a functor from reversible bundle to reversible prime event structures.

a set of causes $D \subseteq \overline{H}$ consisting of the extrusions which removed restrictions on the non-output names in α. In other words, D contains the actions which the transaction is link dependent on, which is important when defining a reversible calculus.

Example 6.0.2 shows the execution of a process using stable non-interleaving early operational semantics for the π-calculus. While the extrusion histories contain enough information to determine the causal relationships between actions, they clearly do not contain enough information to reverse the actions, since they do not record non-extrusion actions such as communication. To define πIH, we extend the extrusion histories so that they contain enough information to reverse the πI-calculus, storing not only extrusions but also communications and inputs and outputs of initially free names. Allowing processes to evolve while moving past actions to a history separate from the process is called dynamic reversibility. In comparison, the πIK-calculus only needs to use keys to annotate the action which were performed using the key, and any names which were substituted during that action. Although causation in the πI-calculus is simpler that in the π-calculus, causation in πIH and πIK is more complex than previous reversible π-calculus variants in one way: we introduce what we believe to be the first *early* semantics of a reversible process calculus. In early semantics, an input can receive a free name of the process from the environment. This means that if a previous output is what made this name free, then that output caused the input receiving the free name.

Example 6.0.2. *The π-calculus process $a(x).\left((\nu b)\left(\overline{x}\langle b\rangle\right)|x(y)\right)|(\nu c)\left(\overline{a}\langle c\rangle\right)$ consists of two parallel processes. The right process sends the bound name c on channel a, which allows the left process to receive the same c on a. The left process then splits in two parallel processes, of which the left sends the bound name b to the right on channel c. In the semantics defined by [52], this execution looks as follows:*

$$\left(\emptyset,\emptyset\right) \vdash a(x).\left((\nu b)\left(\overline{x}\langle b\rangle\right)|x(y)\right)|(\nu c)\left(\overline{a}\langle c\rangle\right)$$

$$\xrightarrow[1[\overline{a}\langle c\rangle][0],\emptyset]{\overline{a}\langle c\rangle}$$

$$\left(\left\{\left(c,1[\overline{a}\langle c\rangle][0]\right)\right\},\emptyset\right) \vdash a(x).\left((\nu b)\left(\overline{x}\langle b\rangle\right)|x(y)\right)|0$$

$$\xrightarrow[0[a(x).((\nu b)(\overline{x}\langle b\rangle)|x(y))][((\nu b)(\overline{c}\langle b\rangle)|c(y))],\{(c,1[\overline{a}\langle c\rangle][0])\}}{a(c)}$$

$$\left(\left\{\left(c,1[\overline{a}\langle c\rangle][0]\right)\right\},\left\{\left(c,0[a(x).\left((\nu b)\left(\overline{x}\langle b\rangle\right)|x(y)\right)][\left((\nu b)\left(\overline{c}\langle b\rangle\right)|c(y)\right)]\right)\right\}\right) \vdash \left((\nu b)\left(\overline{c}\langle b\rangle\right)|c(y)\right)|0$$

$$\xrightarrow[0\langle 0[(\overline{c}\langle b\rangle)][0],1[c(y)][0]\rangle,\emptyset]{\tau}$$

$$\left(\left\{\left(c,1[\overline{a}\langle c\rangle][0]\right)\right\},\left\{\left(c,0[a(x).\left((\nu b)\left(\overline{x}\langle b\rangle\right)|x(y)\right)][\left((\nu b)\left(\overline{c}\langle b\rangle\right)|c(y)\right)]\right)\right\}\right) \vdash (\nu b)\left(0|0\right)|0$$

Here, $a(x)$ is dependent on $\overline{a}\langle c\rangle$ making c a free name, and therefore has the set of causes $\left\{\left(c,1[\overline{a}\langle c\rangle][0]\right)\right\}$.

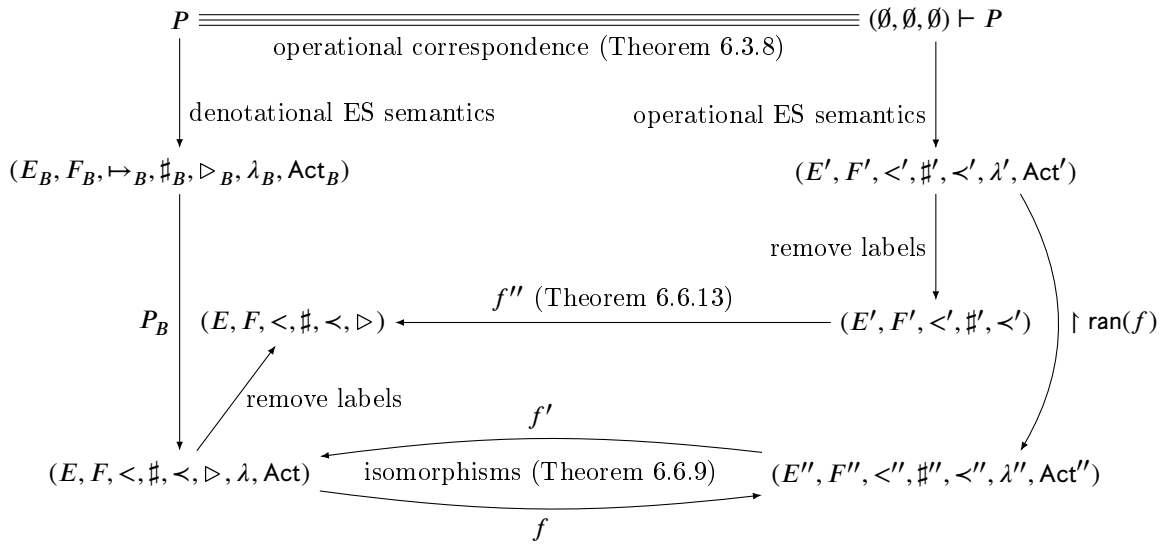


Figure 6.2: A reversible process, P, with no past actions, expressed in the two calculi and the relationship between them and the event structures generated by applying the two semantics to P. The functor P_B is a mapping from reversible bundle to reversible prime event structures. we have three mophisms f, f', and f'' such that $f' = f'' \restriction \mathsf{ran}(f)$.

Although our two reversible variants of the πI-calculus have very different syntax and originate from different ideas, we show an operational correspondence between them in Theorem 6.3.8. We do this despite the extrusion histories containing more information than the keys, since they remember what bound names were before being substituted. The mapping from πIH to πIK bears some resemblance to the one presented from RCCS to CCSK in [68], though with some important differences. Crucially, πIH uses centralised extrusion histories more similar to $\rho\pi$ [60] while RCCS uses distributed memories. Additionally, unlike CCS, πI has substitution as part of its transitions and memories are handled differently by πIK and πIH, and our mapping must consider this. Our encoding is the first result comparing statically and dynamically reversible π-calculi, and the first to compare a statically reversible calculus to a dynamically reversible calculus with centralised memories.

We define event structure semantics of our two reversible πI-calculi. We define the event structure semantics of πIK-calculus denotationally, similarly to Section 5.1, using reversible bundle event structures. We do this by defining the event structure inductively on the syntax of processes. Our event structure semantics of πIH-calculus are operationally defined and derived from the labelled asynchronous transition system semantics of [52], which we can turn into labelled reversible prime event structures [82], using mappings from [96] and Definition 4.2.12. This means each event is a set of possible traces in the process, all of which end with the same action and vary only in the order in which independent actions are performed and whether they include any actions, which are independent of the final action. Since the semantics of our process calculi are causally reversible, so are the event structures they generate. The mappings between our calculi and event structures are shown in Figure 6.1.

As we did for CCSK, we use reversible bundle event structures to describe denotational semantics of πIK-calculus. While it is possible to model the πIK-calculus using reversible prime event structures, using bundle event structures not only gives us fewer events, it also lays the foundation for expanding to adding rollback to the πIK-calculus, similarly to what we did for CCSK in Section 5.2, which cannot be done using reversible prime event structures.

Example 6.0.3. *The πI-calculus process $a(x).\overline{b}(c)$ can be modelled by a bundle event structure where one $\overline{b}(c)$ can be caused by any possible input $a(x)$ might receive, or by a prime event structure where each possible input $a(x)$ might receive causes its own copy of the $\overline{b}(c)$ event.*

The primary difficulty in modelling the π-calculus using event structures or similar frameworks is that

the π-calculus has both *structural* causation, e.g. in $a(b).c(d)$ we get an $a(b)$ event that causes an $c(d)$ event, and *link* causation, e.g. as described previously in $(\nu b)(\overline{a}\langle b\rangle|b(x))$. By limiting ourselves to the πI-calculus we get rid of some of the link causation, such as the previous example, but, since we are modelling early semantics, not all. For example, in the process $a(x)|\overline{b}(c)$, the event $a(c)$ will be caused by $\overline{b}(c)$ making c a free name. In the denotational event structure semantics this is dealt with in the rule for parallel composition, where we create a causal relationship between each output action, and any other action using the name being sent. In the operational semantics this is handled by the fact that any trace including $a(c)$ will inevitably have $\overline{b}(c)$ preceding it.

In addition to an operational correspondence between the two versions of reversible πI-calculus, we show two results about the correspondence between the event structure semantics. For any process with no past actions, we have two inverse morphisms between the event structure generated by the denotational semantics and a subset of the events in the event structure generated by the operational semantics (Theorem 6.6.9). The morphism from the subset of the events in the event structure generated by the operational semantics to the event structure generated by the denotational semantics can also be expanded, if one ignores labels, to map all the events of the event structure generated by the operational semantics to events from event structure generated by the denotational semantics (Theorem 6.6.13). This is illustrated by Figure 6.2.

The reason the operational semantics generate events with labels which do not exist in the denotational semantics is that the operational semantics include actions requiring α-conversion, whereas the denotational semantics assume that the process has already been α-converted so that all free and bound names are different, and only use α-conversion when expanding a replication. This means that in a simple process $\overline{b}(a)$, the denotational semantics will generate one event, labelled $\overline{b}(a)$, while the operational semantics will generate an infinite set of events, with the labels $\{\overline{b}(x) \mid x \text{ is a name}\}$, and the inverse morphisms discussed above will be between the events labelled $\overline{b}(a)$ in the two event structures, while the expanded morphism ignoring labels will map every event in the second event structure to $\overline{b}(a)$.

Section 6.2 describes πIH. Section 6.1 describes πIK. Section 6.3 describes the mapping from πIH to πIK. Section 6.4 gives denotational event structure semantics of πIK. Section 6.5 gives operational event structure semantics of πIH. and Section 6.6 shows a correspondence between the two event structure semantics.

6.1 πIK-Calculus

In this section we introduce πIK, our statically reversible variant of the πI-calculus. The πI-calculus is a restricted variant of the π-calculus wherein output binds the name being sent, so in $\overline{a}(b).P$ name b is bound, corresponding to the π-calculus process $(\nu b)\overline{a}\langle b\rangle.P$. We base πIK's approach to reversibility on previous statically reversible calculi πK [67] and CCSK [80]. Both of these use *communication keys* to denote past actions and which other actions they have interacted with, so $a(x)|\overline{a}(b) \xrightarrow{\tau[n]} a(b)[n]|\overline{a}(b)[n]$ means a communication with the key n has taken place between the two actions. We apply this idea to define early semantics of πIK, which has the following syntax:

$$P ::= \alpha.P \mid \alpha[n].P \mid P_0 + P_1 \mid P_0|P_1 \mid (\nu x)P \quad \alpha ::= \overline{a}(b) \mid a(b) \mid !P$$

The primary difference between applying communication keys to CCS and the πI-calculus is the need to deal with substitution. We need to keep track of not only which actions have communicated with each other, but also which names were substituted when. We do this by giving the substituted names a key, $a_{[n]}$, but otherwise treating them the same as those without the key, except when undoing the input associated with n.

Example 6.1.1 (Substitution)**.** *The process $a(x).\overline{x}(c)|\overline{a}(b)$ can perform a communication of a with key n:*

$$a(x).\overline{x}(c)|\overline{a}(b) \xrightarrow{\tau[n]} (\nu b)(a(b)[n].\overline{b_{[n]}}(c)|\overline{a}(b)[n])$$

This assigns the key n to both parts of the communication, just as in CCSK, and additionally assigns n to all the names that were substituted during this communication. After performing the substitution, the process does not remember the variable used in the input. Therefore, when reversing, the process simply picks a new

$$\frac{\mathsf{std}(P) \quad P' = P[x := b_{[n]}]}{a(x).P \xrightarrow{a(b)[n]} a(b)[n].P'} \qquad \frac{\mathsf{std}(P)}{\overline{a}(b).P \xrightarrow{\overline{a}(b)[n]} \overline{a}(b)[n].P}$$

$$\frac{P \xrightarrow{\mu[m]} P' \quad m \neq n \quad \text{if } \mu = \overline{a}(x) \text{ then } x \notin n(\alpha)}{\alpha[n].P \xrightarrow{\mu[m]} \alpha[n].P'}$$

$$\frac{P_0 \xrightarrow{\mu[n]} P_0' \quad \mathsf{fsh}[n](P_1) \quad \text{if } \mu = \overline{a}(b) \text{ then } b \notin \mathsf{fn}(P_1)}{P_0|P_1 \xrightarrow{\mu[n]} P_0'|P_1} \qquad \frac{P_0 \xrightarrow{\mu[n]} P_0' \quad \mathsf{std}(P_1)}{P_0 + P_1 \xrightarrow{\mu[n]} P_0' + P_1}$$

$$\frac{P_0 \xrightarrow{a(b)[n]} P_0' \quad P_1 \xrightarrow{\overline{a}(b)[n]} P_1' \quad b \notin \mathsf{fn}(P_0)}{P_0|P_1 \xrightarrow{\tau[n]} (\nu b)(P_0'|P_1')} \qquad \frac{\mathsf{std}(P)}{\overline{a}(b).P \xrightarrow{\overline{a}(b)[n]} \overline{a}(b)[n].P}$$

$$\frac{P \xrightarrow{\mu[m]} P' \quad a \notin n(\mu)}{(\nu a)P \xrightarrow{\mu[m]} (\nu a)P'} \qquad \frac{P \equiv Q \xrightarrow{\mu[n]} Q' \equiv P'}{P \xrightarrow{\mu[n]} P'}$$

Table 6.1: πIK-calculus forward semantics

$$\frac{\mathsf{std}(P) \quad x \notin n(P) \quad P' = P[b_{[m]} := x]}{a(b)[m].P \overset{a(b)[m]}{\rightsquigarrow} a(x).P'} \qquad \frac{\mathsf{std}(P)}{\overline{a}(b)[n].P \overset{\overline{a}(b)[n]}{\rightsquigarrow} \overline{a}(b).P} \qquad \frac{P \overset{\mu[m]}{\rightsquigarrow} P' \quad m \neq n}{\alpha[n].P \overset{\mu[m]}{\rightsquigarrow} \alpha[n].P'}$$

$$\frac{P_0 \overset{\mu[n]}{\rightsquigarrow} P_0' \quad \mathsf{fsh}[n](P_1) \quad \text{if } \mu = \overline{a}(b) \text{ then } b \notin \mathsf{fn}(P_1)}{P_0|P_1 \overset{\mu[n]}{\rightsquigarrow} P_0'|P_1} \qquad \frac{P_0 \overset{a(b)[n]}{\rightsquigarrow} P_0' \quad P_1 \overset{\overline{a}(b)[n]}{\rightsquigarrow} P_1'}{(\nu b)(P_0|P_1) \overset{\tau[n]}{\rightsquigarrow} P_0'|P_1'}$$

$$\frac{P_0 \overset{\mu[n]}{\rightsquigarrow} P_0' \quad \mathsf{std}(P_1)}{P_0 + P_1 \overset{\mu[n]}{\rightsquigarrow} P_0' + P_1} \qquad \frac{P \overset{\mu[m]}{\rightsquigarrow} P' \quad b \notin n(\mu)}{(\nu b)P \overset{\mu[m]}{\rightsquigarrow} (\nu b)P'} \qquad \frac{P \equiv Q \overset{\mu[n]}{\rightsquigarrow} Q' \equiv P'}{P \overset{\mu[n]}{\rightsquigarrow} P'}$$

Table 6.2: πIK reverse semantics

one and substitute that for any occurrences of $b_{[n]}$*:*

$$(\nu b)(a(b)[n].\overline{b_{[n]}}(c)|\overline{a}(b)[n]) \overset{\tau[n]}{\rightsquigarrow} a(y).\overline{y}(c)|\overline{a}(b)$$

Table 6.1 shows the forward semantics of the πIK-calculus. The reverse semantics can be seen in Table 6.2. We use α to range over input and output actions and μ over input, output, and τ. We use $\rightsquigarrow$ to denote reverse actions, $\mathsf{std}(P)$ to denote that P is a *standard process*, meaning it does not contain any past actions, $\mathsf{keys}(P)$ to denote the keys mentioned in P, and $\mathsf{fsh}[n](P)$ to denote that a key n is fresh for P. The semantics use structural congruence as defined in Table 6.3. Our semantics very much resemble those of CCSK, with the exceptions of substitution and ensuring that any name being output does not appear elsewhere in the process.

$P\|0 \equiv P$	$P_0\|P_1 \equiv P_1\|P_0$	$P_0\|(P_1\|P_2) \equiv (P_0\|P_1)\|P_2$
$P + 0 \equiv P$	$P_0 + P_1 \equiv P_1 + P_0$	$P_0 + (P_1 + P_2) \equiv (P_0 + P_1) + P_2$
$!P \equiv !P\|P$	$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$	$(\nu a)(P_0\|P_1) \equiv ((\nu a)P_0\|P_1)$ if $a \notin n(P_1)$

Table 6.3: Structural congruence

The reason we need to consider substitution despite dealing with πI-calculus is that we are describing

early semantics. In early semantics, unlike late, an input can receive a name from the environment that is also a free name of the process.

Example 6.1.2. *The process $a(x).\overline{b}(c).x(y)$ can, since b is free, perform a transition*

$$a(x).\overline{b}(c).x(y) \xrightarrow{a(b)[n]} a(b)[n].\overline{b}(c).b_{[n]}(y)$$

In this case, when reversing the input, we would not want to substitute the b doing the output.

We will show that our calculus reverses in a causal manner from any reachable process. We use $\rightarrowtail$ to denote that an action may be forwards or reverse. We show that forwards and reverse transitions are inverse of one another in Proposition 6.1.5.

Lemma 6.1.3. *Given a forwards-reachable process P, any subprocess of P, P' is also forwards-reachable.*

Proof. By induction on P. □

Lemma 6.1.4. *Given a forwards-reachable process P, if $P \xrightarrow{\overline{a}(x)[n]}$ then there cannot exist a past output action $\overline{b}(x)[m]$ anywhere in P.*

Proof. This would require $\overline{b}(x)[m]$ to either prefix, be in parallel with, or be an alternative choice to $\overline{a}(x)$ in P. The first two cases are impossible due to the "if $\mu = \overline{a}(x)$ then $x \notin n(\alpha)$" and "if $\mu = \overline{a}(x)$ then $x \notin \mathsf{fn}(P_1)$" requirements in the rules for propagating $\overline{a}(x)[n]$ past past actions and parallel composition, and the last case is prevented by requiring alternative paths to be standard if we want to propagate an action past the choice. □

Proposition 6.1.5 is a standard result for reversible calculi starting from the first reversible process calculus [35]. It states that any transition must have an inverse.

Proposition 6.1.5 (Loop)**.**

1. *Given a process P, if $P \xrightarrow{\mu[n]} Q$ then $Q \overset{\mu[n]}{\rightsquigarrow} P$.*
2. *Given a forwards-reachable process P, if $P \overset{\mu[n]}{\rightsquigarrow} Q$ then $Q \xrightarrow{\mu[n]} P$.*

Proof. We prove both of these by induction of the transitions in Appendix D.1. □

We also have a diamond confluence property for reverse transitions. This means we can reverse any independent actions in whichever order we want and always get the same result.

Proposition 6.1.6 (Reverse diamond)**.** *Given forwards-reachable processes P, Q, and R, if $P \overset{\mu[m]}{\rightsquigarrow} Q$ and $P \overset{\mu'[n]}{\rightsquigarrow} R$ and $m \neq n$, then there exists a process S such that $Q \overset{\mu'[n]}{\rightsquigarrow} S$ and $R \overset{\mu[m]}{\rightsquigarrow} S$.*

Proof. We prove this by structural induction on P in Appendix D.2. □

We also show that whenever we have a reverse transition for a key, every reverse transition with that key has the same label and structurally congruent end processes. This means we cannot e.g. reverse one part of a communication without reversing the other, or get two different results from reversing the same action.

Proposition 6.1.7. *Given forwards-reachable processes P, Q, and R, if $P \overset{\mu[m]}{\rightsquigarrow} Q$ and $P \overset{\mu'[m]}{\rightsquigarrow} R$ then $\mu = \mu'$ and $R \equiv Q$.*

Proof. We prove this by structural induction on P in Appendix D.3. □

We also have what Danos and Krivine [35] refer to as a parabolic trace, meaning a trace wherein all reverse transitions happen before any forward transitions, whenever we have a trace between two processes.

Lemma 6.1.8. *Given a forwards-reachable process P, if $P \overset{\mu[m]}{\rightsquigarrow} Q$ then Q is forwards-reachable.*

Proof. Since P is forwards-reachable, by Proposition 6.1.5, we have a process P' such that $P \rightsquigarrow^* P'$ and $\mathsf{std}(P')$. By repeated use of Proposition 6.1.6, we get $Q \rightsquigarrow^* P'$. Therefore, by Proposition 6.1.5, Q is forwards-reachable. □

Theorem 6.1.9 (Parabola)**.** *Given processes P and Q, such that $P \rightarrowtail^* Q$, there exists a process R such that $P \rightsquigarrow^* R \rightarrow^* Q$.*

Proof. We say that $P \overset{\mu_0[m_0]}{\rightarrowtail} P_0 \ldots \overset{\mu_n[m_n]}{\rightarrowtail} P_n = Q$ and perform induction on the length of the trace, the number of pairs $\xrightarrow{\mu_i[m_i]}\overset{\mu_{i+1}[m_{i+1}]}{\rightsquigarrow}$ in the trace, and the location of the first such pair.

If no such pair exists then R must exist.

Otherwise, we say that $\xrightarrow{\mu_i[m_i]}\overset{\mu_{i+1}[m_{i+1}]}{\rightsquigarrow}$ is the first such pair in the trace. We have two cases, either $m_i = m_{i+1}$ or not.

If $m_i = m_{i+1}$ then by Propositions 6.1.5 and 6.1.7, $P_{i-1} = P_{i+1}$, and we therefore have a trace $P \overset{\mu_0[m_0]}{\rightarrowtail} P_0 \ldots \overset{\mu_{i-1}[m_{i-1}]}{\rightarrowtail} P_{i-1} \overset{\mu_{i+2}[m_{i+2}]}{\rightarrowtail} \ldots \overset{\mu_n[m_n]}{\rightarrowtail} P_n = Q$.

If $m_i \neq m_{i+1}$ then by Lemma 6.1.8, P_i is forwards-reachable, and therefore by Propositions 6.1.5, $P_i \overset{\mu_i[m_i]}{\rightsquigarrow} P_{i-1}$, and by Proposition 6.1.6 we have a trace $P \overset{\mu_0[m_0]}{\rightarrowtail} P_0 \ldots P_{i-1} \overset{\mu_{i+1}P_i'[m_{i+1}]}{\rightsquigarrow} \xrightarrow{\mu_i[m_i]} P_{i+1} \ldots \overset{\mu_n[m_n]}{\rightarrowtail} P_n = Q$ □

Corollary 6.1.10. *A process P is reachable if and only if it is forwards-reachable.*

This means all reachable processes are forwards-reachable, and we can therefore take any previous results about forwards-reachable processes and apply them to all reachable processes. We also get a forwards diamond confluence (Proposition 6.1.13), similar to the reverse diamond in Proposition 6.1.6. We can view actions as being independent if the resulting processes are not structurally congruent but can still eventually reach the same state.

Lemma 6.1.11. *Given a process P, if $P \xrightarrow{\mu[m]}$ then $m \notin \mathsf{keys}(P)$.*

Proof. Follows from the semantics. □

Lemma 6.1.12. *Given processes P, Q, if $P \rightarrow^* Q$ then $\mathsf{keys}(P) \subseteq \mathsf{keys}(Q)$.*

Proof. Follows from the semantics. □

Proposition 6.1.13 (Forward diamond [80])**.**

1. *Given reachable processes P, Q, and R, if $P \xrightarrow{\mu[m]} Q$, $P \xrightarrow{\mu'[n]} R$, $m = n$, and there exists a process T such that $Q \rightarrow^* T$ and $R \rightarrow^* T$, then $\mu = \mu'$ and $R \equiv Q$.*

2. *Given reachable processes P, Q, and R, if $P \xrightarrow{\mu[m]} Q$, $P \xrightarrow{\mu'[n]} R$, and there exists a process T such that $Q \rightarrow^* T$ and $R \rightarrow^* T$, then there exists a process S such that $Q \xrightarrow{\mu'[n]} S$ and $R \xrightarrow{\mu'[m]} S$ and $S \rightarrow^* T$.*

Proof. We prove both simultaneously by structural induction on P in Appendix D.4. □

We have now proved the most important results for a causally reversible system hold for πIK as they did for its predecessor CCSK. Next we move on to show that we can also define early semantics of a dynamically reversible calculus.

6.2 πIH-Calculus

Stable non-interleaving, early operational semantics of the π-calculus were defined in [52], using locations and extrusion histories to keep track of link causation. We will in this section use a similar approach to define a reversible variant of the πI-calculus, πIH, using the locations and histories to keep track of not just causation, but also past actions. The syntax of πIH processes is:

$$P ::= \sum_{i\in I} \alpha_i.P_i \;\mid\; P_0|P_1 \;\mid\; (\nu x)P \qquad \alpha ::= \overline{a}(b) \;\mid\; a(b)$$

The forward semantics of πIH can be seen in Table 6.4 and the reverse semantics in Table 6.5. We associate each transition with an action $\mu ::= \alpha \mid \tau$ and a location u (Definition 6.2.1), describing where the action came from and what changes are made to the process as a result of the action. We store these location and action pairs in extrusion and communication histories associated with processes, so $(\overline{H}, \underline{H}, H) \vdash P$ means that if (μ, u) is an action and location pair in the output history $\overline{H}$ then μ is an output action, which P previously performed at location u. Similarly $\underline{H}$ contains pairs of input actions and locations and H contains triples of two communicating actions and the location associated with their communication. We use $\mathbf{H}$ as shorthand for $(\overline{H}, \underline{H}, H)$.

Definition 6.2.1 (Location [52])**.** *A location u of an action μ is of one of the following forms:*

1. *$l[P][P']$ if μ is an input or output, where $l \in \{0,1\}^*$ describes the path taken through parallel compositions to get to μ's origin, P is the subprocess reached by following the path before μ has been performed, and P' is the result of performing μ in P.*
2. *$l\langle 0l_0[P_0][P'_0], 1l_1[P_1][P'_1]\rangle$ if $\mu = \tau$, where $l0l_0[P_0][P'_0]$ and $l1l_1[P_1][P'_1]$ are the locations of the two actions communicating.*

The path l may be empty if the action was performed at the outer level of the process before any parallel composition.

We can use this syntax to describe the transaction between a buyer and seller in Example 6.0.1.

Example 6.2.2. *We assume that a is a publicly known channel and describe the two involved processes:*

$$\mathbf{seller} = a(x).x(y).\overline{x}(price).x(z)$$

$$\mathbf{buyer} = \overline{a}(ch).\overline{ch}(prod).ch(x).\overline{ch}(addr)$$

*Here the buyer initiates communication by sending a channel ch which the rest of the transaction will use to communicate. The buyer then informs the seller which product, $prod$, they want to buy and the seller responds with a price $price$. Finally the buyer tells the seller which address, $addr$, to ship the product. To execute the transaction, we put these processes in parallel and give them an extrusion history, $(\emptyset,\emptyset,\emptyset) \vdash$ **buyer** | **seller**.*

Once the first two communications have taken place, we have the following process:

$$(\emptyset,\emptyset,H) \vdash (\nu\ ch)(\nu\ prod)(\overline{ch}(price).ch(z) \mid ch(x).\overline{ch}(addr))$$

with the communication history

$$H = \left\{ \begin{array}{l} ((a(x),\overline{a}(ch))\langle 0[a(x).x(y).\overline{x}(price).x(z)][ch(y).\overline{ch}(price).ch(z)], \\ \qquad 1[\overline{a}(ch).\overline{ch}(prod).ch(x).\overline{ch}(addr)][\overline{ch}(prod).ch(x).\overline{ch}(addr)]\rangle), \\ ((ch(y),\overline{ch}(prod))\langle 0[ch(y).\overline{ch}(price).ch(z)][\overline{ch}(price).ch(z)], \\ \qquad 1[\overline{ch}(prod).ch(x).\overline{ch}(addr)][ch(x).\overline{ch}(addr)]\rangle) \end{array} \right\}$$

At this point, the seller may realise they do not have the requested product, and they can consequently decide to reverse the most recent communication. This is done through the reverse transition:

$$\xrightarrow{\tau} \langle 0[ch(y).\overline{ch}(price).ch(z)][\overline{ch}(price).ch(z)], 1[\overline{ch}(prod).ch(x).\overline{ch}(addr)][ch(x).\overline{ch}(addr)]\rangle$$

Resulting in the process:

$$(\emptyset,\emptyset,\{((a(x),\overline{a}(ch))\langle 0[a(x).x(y).\overline{x}(price).x(z)][ch(y).\overline{ch}(price).ch(z)],\\ 1[\overline{a}(ch).\overline{ch}(prod).ch(x).\overline{ch}(addr)][\overline{ch}(prod).ch(x).\overline{ch}(addr)]\rangle\})\\ \vdash(\nu\ ch)(ch(y).\overline{ch}(price).ch(z)\mid \overline{ch}(prod)ch(x).\overline{ch}(addr))$$

The buyer then has the opportunity to send a different product if they want to buy something else or otherwise reverse the beginning of the transaction. This shows how reversible calculi avoid needing to explicitly describe how to handle cancelling an ongoing transition, which simplifies writing processes.

When defining the semantics of πIH-calculus, we use the operations on extrusion histories from Definition 6.2.3. These (1) add a branch to the path in every location, (2) isolate the extrusions whose locations begin with a specific branch, (3) isolate the extrusions whose locations begin with a specific branch and then remove the first branch from the locations, and (4) add a pair to the history it belongs in.

Definition 6.2.3 (Operations on extrusion histories [52])**.** *Given an extrusion history $(\overline{H},\underline{H},H)$, for $H^*\in\{\overline{H},\underline{H},H\}$ we have the following operations for $i\in\{0,1\}$:*

1. $iH^*=\{(\mu,iu)\mid(\mu,u)\in H^*\}$;
2. $[i]H^*=\{(\mu,iu)\mid(\mu,iu)\in H^*\}$;
3. $[\check{i}]H^*=\{(\mu,u)\mid(\mu,iu)\in H^*\}$;
4. $$\mathbf{H}+(\mu,u)=\begin{cases}(\overline{H}\cup\{L\},\underline{H},H) & \text{if } (\mu,u)=(\overline{a}(n),u)\\ (\overline{H},\underline{H}\cup\{L\},H) & \text{if } (\mu,u)=(a(x),u)\\ (\overline{H},\underline{H},H\cup\{L\}) & \text{if } (\mu,u)=((a(x),\overline{a}(n)),l\langle u_0,u_1\rangle)\end{cases}.$$

The forwards semantics of πIH have six rules. In [OUT] the action is an output, the location is the process before and after doing the output, and they are added to the output history. The equivalent reverse rule, [OUT^{-1}], similarly removes the pair from the history and transforms the process from the second part of the location back to the first. The input rule [IN] works similarly, but performs a substitution on the received name and adds the pair to the input history instead. In [PAR$_i$] we isolate the parts of the histories whose locations start with i and use those to perform an action in P_i, getting $\mathbf{H}_i'\vdash P_i'$. It then replaces the part of the histories parts of the histories whose locations start with i with $\mathbf{H}_i'$ when propagating the action through the parallel. A communication in [COM$_i$] adds memory of the communication to the history. The rules [SCOPE] and [STR] are standard and self-explanatory.

The reverse rules use the extrusion histories to find a location $l[P][P']$ such that the current state of the subprocess at l is P', and change it to P.

In these semantics structural congruence, consisting only of α-conversion, $(\nu\ a)(\nu b)P\equiv(\nu\ b)(\nu\ a)P$, and $!P\equiv!P|P$, is primarily used to create and remove extra copies of a replicated process when reversing the action that happened before the replication. Since we use locations in our extrusion histories, we try to avoid using structural congruence any more than necessary. However, not using it for parallel composition would mean that we would need some other way of preventing traces such as $\mathbf{H}\vdash!P\xrightarrow[u]{\mu}\overset{\mu}{\underset{u}{\rightsquigarrow}}\mathbf{H}\vdash!P|P$, which allows a process to reach a state it could not reach via a parabolic trace. Using structural congruence for replication does not cause any problems for the locations, as we can tell past actions originating in each copy of P apart by the path in their location, with actions from the ith copy having a path of i 0s followed by a 1.We show in Example 6.2.4 how actions and locations are stored in the histories.

Example 6.2.4. *Consider the process $a(x).(\overline{x}(b)|x(y))|\overline{a}(c)$. If we start with empty histories, each transition adds actions and locations:*

$$\frac{u = [\sum_{i\in I} \alpha_i.P_i][P_j] \quad \alpha_j = \overline{a}(n) \quad j \in I}{\mathbf{H} \vdash \sum_{i\in I} \alpha_i.P_i \xrightarrow[u]{\alpha_j} (\overline{H} \cup \{(\overline{a}(n), u)\}, \underline{H}, H) \vdash P_j} \text{ [OUT]}$$

$$\frac{u = [\sum_{i\in I} \alpha_i.P_i][P_j] \quad P'_j = P_j[x := n] \quad \alpha_j = a(x) \quad j \in I \quad (\overline{b}(n), u') \notin \overline{H}}{\mathbf{H} \vdash \sum_{i\in I} \alpha_i.P_i \xrightarrow[u]{a(n)} (\overline{H}, \underline{H} \cup \{(a(n), u)\}, H) \vdash P'_j} \text{ [IN]}$$

$$\frac{([\check{i}]\overline{H}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i \xrightarrow[u]{\mu} \mathbf{H}'_i \vdash P'_i \quad P'_{1-i} = P_{1-i} \quad \text{if } \mu = \overline{a}(n) \text{ then } n \notin \mathsf{fn}(P_{1-i} \text{ and } (\overline{b}(n), u') \notin \overline{H})}{\mathbf{H} \vdash P_0|P_1 \xrightarrow[iu]{\mu} ((\overline{H} \setminus [i]\overline{H}) \cup i\overline{H'_i}, (\underline{H} \setminus [i]\underline{H}) \cup i\underline{H'_i}, (H \setminus [i]H) \cup iH'_i) \vdash P'_0|P'_1} \text{ [PAR}_i\text{]}$$

$$\frac{\begin{array}{c}([\check{i}]\overline{H}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i \xrightarrow[v_i]{\alpha_i} \mathbf{H}'_i \vdash P'_i \quad \alpha_i = \overline{a}(n) \quad \alpha_j = a(n) \\ ([\check{j}]\overline{H}, [\check{j}]\underline{H}, [\check{j}]H) \vdash P_j \xrightarrow[v_j]{\alpha_i} \mathbf{H}'_j \vdash P'_j \quad j = 1 - i \quad n \notin \mathsf{fn}(P_j)\end{array}}{\mathbf{H} \vdash P_0|P_1 \xrightarrow[(0v_0, 1v_1)]{\tau} (\overline{H}, \underline{H}, H \cup \{((\alpha_0, \alpha_1), \langle 0v_0, 1v_1\rangle)\}) \vdash (\nu n)(P'_0|P'_1)} \text{ [COM}_i\text{]}$$

$$\frac{\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P' \quad x \notin n(\mu)}{\mathbf{H} \vdash (\nu x)P \xrightarrow[u]{\mu} \mathbf{H}' \vdash (\nu x)P'} \text{ [SCOPE]} \qquad \frac{P \equiv P' \quad \mathbf{H} \vdash P' \xrightarrow[u]{\mu} \mathbf{H}' \vdash Q' \quad Q' \equiv Q}{\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash Q} \text{ [STR]}$$

Table 6.4: Semantics of πIH (forwards rules)

$(\emptyset, \emptyset, \emptyset) \vdash a(x).(\overline{x}(b)|x(y))|\overline{a}(c)$

$\xrightarrow[1[\overline{a}(c)][0]]{\overline{a}(c)}$

$(\{(\overline{a}(c), 1[\overline{a}(c)][0])\}, \emptyset, \emptyset) \vdash a(x).(\overline{x}(b)|x(y))|0$

$\xrightarrow[0[a(x).(\overline{x}(b)|x(y))][(\overline{a}(b)|a(y))]]{a(c)}$

$(\{(\overline{a}(c), 1[\overline{a}(c)][0])\}, \{(a(c), 0[a(x).(\overline{x}(b)|x(y))][(\overline{a}(b)|a(y))])\}, \emptyset) \vdash (\overline{a}(b)|a(y))|0$

$\xrightarrow[0\langle 0[\overline{a}(b)][0], 1[a(y)][0]\rangle]{\tau}$

$(\{(\overline{a}(c), 1[\overline{a}(c)][0])\}, \{(a(c), 0[a(x).(\overline{x}(b)|x(y))][(\overline{a}(b)|a(y))])\}, \{((\overline{a}(b), a(b)), 0\langle 0[\overline{a}(b)][0], 1[a(y)][0]\rangle)\}) \vdash (\nu b)(0|0)|0$

The first transition adds an action and location to the output history, which allows the input to receive the same name, c. The input transition is than added to the input history, and finally the synchronisation history stores both of the actions involved in the final transition and their locations.

We then show a correspondence between forward and reverse transitions.

Lemma 6.2.5. *Let P be a process. If there exists an extrusion history $\mathbf{H}$ such that $\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash P'$ then there exists L such that $\mathbf{H} = \mathbf{H}' + L$, and for any extrusion history $\mathbf{H}''$ not containing L, $\mathbf{H}'' + L \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}'' \vdash P'$.*

Proof. [SCOPE^{-1}] and [PAR$_i^{-1}$] simply propagate the changes to extrusion histories, and [COM$_i^{-1}$], [IN^{-1}], and [OUT^{-1}] remove exactly one extrusion from the histories, which is the only one they depend on. □

Proposition 6.2.6 (Loop)**.**

1. *Given a forwards-reachable πIH-calculus process P and an extrusion history $\mathbf{H}$, if $\mathbf{H} \vdash P \xrightarrow[u]{\alpha} \mathbf{H}' \vdash Q$, then $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.*

$$\frac{u = \langle j \rangle[\sum_{i \in I} \alpha_i.P_i][P_j] \quad \alpha_j = \overline{a}(n) \quad j \in I \quad (\overline{a}(n), u) \in \overline{H}}{\mathbf{H} \vdash P_j \overset{\alpha_j}{\underset{u}{\rightsquigarrow}} (\overline{H} \setminus \left\{(\overline{a}(n), u)\right\}, \underline{H}, H) \vdash \sum_{i \in I} \alpha_i.P_i} \; [\mathrm{OUT}^{-1}]$$

$$\frac{u = \langle j \rangle[\sum_{i \in I} \alpha_i.P_i][P_j] \quad P'_j = P_j[x := n] \quad \alpha_j = a(x) \quad j \in I \quad (a(n), u) \in \underline{H}}{\mathbf{H} \vdash P'_j \overset{a(n)}{\underset{u}{\rightsquigarrow}} (\overline{H}, \underline{H} \setminus \{(a(n), u)\}, H) \vdash \sum_{i \in I} \alpha_i.P_i} \; [\mathrm{IN}^{-1}]$$

$$\frac{\begin{array}{l}([\check{i}]\overline{H}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H}'_i \vdash P'_i \quad P'_{1-i} = P_{1-i} \\ \text{if } \alpha = \overline{a}(n) \text{ then } n \notin \mathsf{fn}(P_{1-i})\end{array}}{\mathbf{H} \vdash P_0 | P_1 \overset{\alpha}{\underset{iu}{\rightsquigarrow}} ((\overline{H} \setminus [i]\overline{H}) \cup i\overline{H'_i}, (\underline{H} \setminus [i]\underline{H}) \cup i\underline{H'_i}, (H \setminus [i]H) \cup iH'_i) \vdash P'_0 | P'_1} \; [\mathrm{PAR}_i^{-1}]$$

$$\frac{\begin{array}{l}([\check{i}]\overline{H} \cup \{(\overline{a}(n), v_i)\}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i \overset{\overline{a}(n)}{\underset{v_i}{\rightsquigarrow}} \mathbf{H}'_i \vdash P'_i \\ ([\check{j}]\overline{H}, [\check{j}]\underline{H} \cup \{(a(n), v_j)\}, [\check{j}]H) \vdash P_j \overset{a(n)}{\underset{v_j}{\rightsquigarrow}} \mathbf{H}'_j \vdash P'_j \\ j = 1 - i \quad n \notin \mathsf{fn}(P_j) \quad ((a(n), \overline{a}(n)), \langle 0v_0, 1v_1 \rangle) \in H\end{array}}{\mathbf{H} \vdash (\nu n)(P_0 | P_1) \overset{\tau}{\underset{\langle 0v_0, 1v_1 \rangle}{\rightsquigarrow}} (\overline{H}, \underline{H}, H \setminus \left\{((a(n), \overline{a}(n)), \langle 0v_0, 1v_1 \rangle)\right\}) \vdash P'_0 | P'_1} \; [\mathrm{COM}_i^{-1}]$$

$$\frac{\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash P' \quad x \notin n(\alpha)}{\mathbf{H} \vdash (\nu x)P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash (\nu x)P'} \; [\mathrm{SCOPE}^{-1}] \qquad \frac{P \equiv P' \quad \mathbf{H} \vdash P' \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash Q' \quad Q' \equiv Q}{\mathbf{H} \vdash P \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash Q} \; [\mathrm{STR}^{-1}]$$

Table 6.5: Semantics of reversible πIH-calculus (reverse rules)

2. *Given a forwards-reachable πIH-calculus process P and an extrusion history $\mathbf{H}$, if $\mathbf{H} \vdash P \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash Q$, then $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\longrightarrow}} \mathbf{H} \vdash P$.*

Proof. We prove both by induction on the transitions in Appendix D.5. □

To get causal semantics of πIH, we add a set of causes, D, to each transition. D associates the non-output names of an action with the locations where they were extruded, and thereby tells us which actions extruded those names necessary for the current action. If the non-output names in an action were always free then D is empty.

Definition 6.2.7 (Causal semantics [52])**.** *The early causal semantics consist of transitions of the form $\mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H}' \vdash P'$ where $\mathbf{H} \vdash P \xrightarrow[u]{\alpha} \mathbf{H}' \vdash P'$ and*

1. *$(n,u) \in D \Rightarrow \exists a.\ (\overline{a}(n), u) \in \overline{H}$;*
2. *if $(n,u),(n,u') \in D$ then $u = u'$;*
3. *$(n,u) \in D$ if and only if $(\overline{a}(n), u') \in \overline{H}$ for some a, u' and $n \in \mathsf{no}(\alpha)$ where $\mathsf{no}(\alpha)$ is the set of non-output names in α, defined by $\mathsf{no}(\overline{a}(b)) = \{a\} \setminus \{b\}$, $\mathsf{no}(a(b)) = \{a, b\}$ and $\mathsf{no}(\tau) = \emptyset$.*

This lets us define a notion of independence on locations and causes. Our definition of independence differs from the notions of concurrency used by other reversible process calculi, in that it is not just defined on coinitial [35] or composable [33] transitions. Having the definition apply to any two transitions becomes important in our event structure semantics in Section 6.5.

Definition 6.2.8 (Path)**.** *Given a location u, its set of paths is defined as*

$$\mathsf{loc}(u) = \begin{cases} \{l\} & \text{if } u = l[P][P'] \\ \{ll_0, ll_1\} & \text{if } u = l\langle l_0[P_0][P_0'], l_1[P_1][P_0]\rangle \end{cases}$$

Definition 6.2.9 (Independence [52])**.** *Two locations, u_0 and u_1, are independent, $u_0 I u_1$, if for all $l_0 \in \mathsf{loc}(u_0)$ and $l_1 \in \mathsf{loc}(u_1)$, there exist l, l_0', l_1' such that either $l_0 = l0l_0'$ and $l_1 = l1l_1'$ or $l_0 = l1l_0'$ and $l_1 = l0l_1'$.*

Two transitions $\overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}}$ and $\overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}}$ are independent if u_0 and u_1 are independent, and there does not exist n such that $D_i(n) = u_{1-i}$.

As we did in the previous section, we prove the square property, though in this case we use our definition of independence of transitions. Unlike Rπ [33], we are not keeping track of which restriction was opened by each output action. We therefore get transitions to P' with the same labels as the original, rather than equivalent labels.

Proposition 6.2.10 (Square [35])**.** *If $\mathbf{H} \vdash P \overset{\mu_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_0 \vdash P_0$ and $\mathbf{H} \vdash P \overset{\mu_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_1 \vdash P_1$ are independent transitions then there exists $\mathbf{H}' \vdash P'$ such that $\mathbf{H}_0 \vdash P_0 \overset{\mu_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}' \vdash P'$ and $\mathbf{H}_1 \vdash P_1 \overset{\mu_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}' \vdash P'$.*

Proof. We prove this in Appendix D.6. The proof is similar to Theorem 14 of [52]. □

Unlike in the previous section, since we have an explicit notion of independence, we can define trace equivalence. We can then use trace equivalence to get an alternative description of the parabolic property we want our reversible calculus to have. We use t to refer to traces of the form $\mathbf{H}_0 \vdash P_0 \overset{\mu_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_1 \vdash P_1 \ldots \overset{\mu_{n-1}}{\underset{u_{n-1},D_{n-1}}{\rightarrowtail}} \mathbf{H}_n \vdash P_n$, $\underline{\mathsf{t}}$ to refer to the reversal of t, which according to Proposition 6.2.6 is a trace if and only if t is, $\mathsf{t};\mathsf{t}'$ to compose traces, and ε to refer to the empty trace.

Definition 6.2.11 (Causal equivalence [35]). *We define trace equivalence $\sim$ as the least equivalence relation closed under composition such that:*

$$\begin{array}{rcl} \mathsf{t};\underline{\mathsf{t}} & \sim & \varepsilon \\ \underline{\mathsf{t}};\mathsf{t} & \sim & \varepsilon \\ \mathbf{H} \vdash P \overset{\mu_0}{\underset{u_0,D_0}{\rightarrowtail}} \overset{\mu_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}' \vdash P' & \sim & \mathbf{H} \vdash P \overset{\mu_1}{\underset{u_1,D_1}{\rightarrowtail}} \overset{\mu_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}' \vdash P' \text{ if } \overset{\mu_1}{\underset{u_1,D_1}{\rightarrowtail}} \text{ and } \overset{\mu_0}{\underset{u_0,D_0}{\rightarrowtail}} \\ & & \text{are independent} \end{array}$$

Lemma 6.2.12. *Given a forwards-reachable process $\mathbf{H} \vdash P$ with transitions $\mathbf{H} \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}} \mathbf{H}_0 \vdash P_0$ and $\mathbf{H} \vdash P \overset{\alpha_1}{\underset{u_1,D_1}{\rightsquigarrow}} \mathbf{H}_1 \vdash P_1$, either $\alpha_0 = \alpha_1$, $u_0 = u_1$, $D_0 = D_1$, and $\mathbf{H}_0 \vdash P_0 \equiv \mathbf{H}_1 \vdash P_1$, or $\overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}}$ and $\overset{\alpha_1}{\underset{u_1,D_1}{\rightsquigarrow}}$ are independent.*

Proof. This follows from Definition 6.2.9. See Appendix D.7 for details. □

Lemma 6.2.13. *Given a process, $\mathbf{H} \vdash P$, we cannot have an infinite reverse trace $\mathbf{H} \vdash P \twoheadrightarrow^*$.*

Proof. Each reverse transition removes an element from $\mathbf{H}$. Since $\mathbf{H}$ is finite, so is every reverse trace. □

Theorem 6.2.14 (Parabola [35]). *Let t be a trace, then there exists a forward trace t_f and a backward trace t_b such that $\mathsf{t} \sim \mathsf{t}_b;\mathsf{t}_f$.*

Proof. Follows from Proposition 6.2.10 and Lemma 6.2.12 according to [63]. □

As for πIK, this implies that all reachable processes are forwards-reachable, an important property of causally-reversible systems.

Theorem 6.2.15 (Trace equivalence [35]). *Let t and t' be cofinal and coinitial traces. Then $\mathsf{t} \sim \mathsf{t}'$.*

Proof. Follows from Theorem 6.2.14 and Lemma 6.2.13 according to [63]. □

Theorems 6.2.14 and 6.2.15 and not the only results from [63] we can apply to the πIH calculus. We gain several interesting properties by proving some simple results, both of which follow easily from our definition of independence. First we recall their definition of an event. An event is an equivalence class of transitions with the same label which form two sides of a commuting diamond. In πIH, due to independence being defined on labels, this is all transitions with the same label.

Definition 6.2.16 (Events [63]). *Let $\sim$ be the smallest equivalence relation on transitions satisfying: if $\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_Q \vdash Q$, $\mathbf{H}_P \vdash P \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_R \vdash R$, $\mathbf{H}_Q \vdash Q \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_S \vdash S$, $\mathbf{H}_R \vdash R \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_S \vdash S$, $\overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}}$ and $\overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}}$ are independent, and*

- *$\mathbf{H}_Q \vdash Q \neq \mathbf{H}_R \vdash R$ if $\overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}}$ and $\overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}}$ are either both forwards or both backwards;*
- *$\mathbf{H}_P \vdash P \neq \mathbf{H}_S \vdash S$ otherwise;*

then $\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_Q \vdash Q \sim \mathbf{H}_R \vdash R \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_S \vdash S$. The equivalence classes of forward transitions, written $[\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q]_\sim$, are events. The equivalence classes of backwards transitions, written $[\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}} \mathbf{H}_Q \vdash Q]_\sim$ are reverse events.

Coinitial propagation of independence states that independence can be propagated around a commuting diamond.

Lemma 6.2.17 (Coinitial propagation of independence [63]). *Given a diamond consisting of transitions* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *and* $\mathbf{H}_P \vdash P \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_R \vdash R$ *and* $\mathbf{H}_Q \vdash Q \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_S \vdash S$ *and* $\mathbf{H}_R \vdash R \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_S \vdash S$ *with* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *and* $\mathbf{H}_P \vdash P \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_R \vdash R$ *independent, we get* $\mathbf{H}_Q \vdash Q \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_S \vdash S$ *and* $\mathbf{H}_Q \vdash Q \overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}} \mathbf{H}_P \vdash P$ *are independent.*

Proof. Follows from independence being defined on labels of transitions. □

Independence respect events if two transitions being in the same event means they are independent of the same other transitions.

Lemma 6.2.18 (Independence respects events [63]). *Whenever* $\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_Q \vdash Q \sim \mathbf{H}_{P'} \vdash P' \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$ *and* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$ *and* $\mathbf{H}_R \vdash R \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_S \vdash S$ *are independent, we get* $\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_Q \vdash Q$ *and* $\mathbf{H}_R \vdash R \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_S \vdash S$ *are independent.*

Proof. Follows from independence being defined on labels of transitions. □

These lemmas along with Proposition 6.2.10 and Lemmas 6.2.12 and 6.2.13 give πIH a number of key properties from [63], including causal safety (Proposition 6.2.20) and causal liveness (Proposition 6.2.21). Causal safety and liveness were defined by [63] to strengthen previous results for reversible calculi. Causal safety means an action cannot be reversed if it has caused another action and causal liveness that if an action has caused no other actions then it can be reversed. To formally define these properties, we first define how to count occurrences in a path.

Definition 6.2.19 ([63]). *Let* t *be a trace and* e *be an event. Let* $\sharp\,(t, e)$ *be the number of occurrences of transitions* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *in* t *such that* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q \in e$*, minus the number of occurrences of transitions* $\mathbf{H}_R \vdash R \overset{\alpha_1}{\underset{u_1,D_1}{\rightsquigarrow}} \mathbf{H}_S \vdash S$ *such that* $\mathbf{H}_S \vdash S \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_R \vdash R \in e$.

Proposition 6.2.20 (Causal safety [63]). *Whenever we have a transition,* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *and a trace* $t = \mathbf{H}_Q \vdash Q \rightarrowtail^* \mathbf{H}_R \vdash R$ *such that* $\sharp\,(t, [\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q]_\sim) = 0$*, and a transition* $\mathbf{H}_S \vdash S \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_R \vdash R$ *with* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q \sim \mathbf{H}_S \vdash S \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_R \vdash R$*, we get that all* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$ *such that* $\sharp\,(t, [\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q']_\sim) > 0$*, are independent of* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$.

Proof. Follows from Proposition 6.2.10 and Lemmas 6.2.12, 6.2.13, 6.2.17, and 6.2.18. □

Proposition 6.2.21 (Causal liveness [63]). *Whenever we have a transition,* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *and a trace* $t = \mathbf{H}_Q \vdash Q \rightarrowtail^* \mathbf{H}_R \vdash R$ *such that* $\sharp\,(t, [\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q]_\sim) = 0$*, and all* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$ *such that* $\sharp\,(t, [\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q']_\sim) > 0$*, are independent of* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$*, we get a transition* $\mathbf{H}_S \vdash S \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_R \vdash R$ *with* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q \sim \mathbf{H}_S \vdash S \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_R \vdash R$.

Proof. Follows from Proposition 6.2.10 and Lemmas 6.2.12, 6.2.13, 6.2.17, and 6.2.18. □

The last result required by [63] to get all the properties defined in their paper is that independence of events is coinitial. However, our definition of independence allows a process to have independent transitions without having a state in which both can be performed. Consider the process $(\emptyset,\emptyset,\emptyset) \vdash \overline{a}(b).(\overline{c}(d) \mid 0)+\overline{e}(f).(0 \mid \overline{g}(h))$. This process will eventually be able to do $\xrightarrow[0[\overline{c}(d)][0],\emptyset]{\overline{c}(d)}$ or $\xrightarrow[1[\overline{g}(h)][0],\emptyset]{\overline{g}(h)}$, which are independent transitions, but no state of P exists where both can be performed. We therefore cannot say that independence of events is coinitial in πIH. Allowing actions on opposite sides of a choice to be independent sets us apart from most traditional definitions of independence or concurrency, such as [24]. However, since these independent events on opposite sides of a choice are caused by conflicting events due to guarded choice, and are therefore not coinitial or consecutive, this does not cause us any problems.

Independence of events being coinitial is used in [63] to prove that reversing preserves independence.

Lemma 6.2.22 (Reversing preserves independence)**.**

- *Given independent transitions* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *and* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$*, we also have independence between* $\mathbf{H}_Q \vdash Q \overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}} \mathbf{H}_P \vdash P$ *and* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$*.*
- *Given independent transitions* $\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}} \mathbf{H}_Q \vdash Q$ *and* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$*, we also have independence between* $\mathbf{H}_Q \vdash Q \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_P \vdash P$ *and* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$*.*

Proof. Follows from independence being defined on labels of transitions. □

We now have a number of important results for reversible process calculi, all straightforwardly given by our simple definition of causal semantics. This makes it likely that if we were to expand πIH to reverse the full early semantics of the π-calculus described by [52], we would still get these results quite easily.

Since we based our calculus on the π-calculus presented in [52], we would like to show a correspondence between the two. Our calculus is less expressive because it is a πI-calculus rather than a π-calculus, but contains more information in the histories because it is able to reverse. We show that by removing the extra information needed to reverse from our extrusion histories, we can get a fragment of [52].

Definition 6.2.23 (Mapping a πIH to a π-calculus process with an extrusion history)**.** *Given πIH process $\mathbf{H} \vdash P$, the corresponding π-calculus process with extrusion history, $phi_I(\mathbf{H} \vdash P)$, is defined as:*

$$\phi_I(\mathbf{H} \vdash P) = (\phi_I(\overline{H}), \phi_I(\underline{H})) \vdash \phi_I(P) \qquad \phi_I(\{(\overline{a}(b), u)\} \cup \overline{H}) = \{(b, u)\} \cup \phi_I(\overline{H})$$
$$\phi_I(\{(a(b), u)\} \cup \underline{H}) = \{(b, u)\} \cup \phi_I(\underline{H}) \qquad \phi_I(P_0 | P_1) = \phi_I(P_0) | \phi_I(P_1)$$
$$\phi_I(\sum_{i \in J} P_i) = \sum_{i \in J} \phi_I(P_i) \qquad \phi_I((\nu a)P) = (\nu a)\phi_I(P)$$
$$\phi_I(\overline{a}(b).P) = (\nu b)\overline{a}\langle b\rangle.\phi_I(P) \qquad \phi_I(a(x).P) = \underline{a}(x).\phi_I(P)$$
$$\phi_I(!P) = !\phi_I(P)$$

Proposition 6.2.24. *Given a πIH process $\mathbf{H} \vdash P$:*

- *if there exists a transition $\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$, then there exists a transition $\phi_I(\mathbf{H} \vdash P) \xrightarrow[u]{\mu} \phi_I(\mathbf{H}'' \vdash P'')$ for some $\mathbf{H}'' \vdash P'' \equiv \mathbf{H}' \vdash P'$ in the semantics of [52];*
- *if $\phi_I(\mathbf{H} \vdash P) \xrightarrow[u]{\mu} \phi_I(\mathbf{H}' \vdash P')$, then $\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$.*

Proof. The only difference between the semantics is outputs being treated as bound names, and therefore put in extrusion histories automatically, adding a history of communications, which ϕ_I removes, and replacing the (Rep) rule with structural congruence. □

We now know that our forwards semantics are the early semantics of the internal π-calculus and that our reverse semantics causally reverse our forwards semantics, which are the two results we want for a reversible πI-calculus. As we now have both a statically and a dynamically reversible πI-calculus, the next step is to compare the two.

6.3 Mapping Between πIK-Calculus and πIH-Calculus

We will now define a mapping from πIH to πIK and show that we have an operational correspondence in Theorem 6.3.8. The extrusion histories store more information than the keys, as they keep track of which names were substituted, as illustrated by Example 6.3.1. This means we lose some information in our mapping, but not information we need.

Example 6.3.1. *Consider the processes* $(\emptyset, \{(a(b), [a(x)][0])\}, \emptyset) \vdash 0$ *and* $a(b)[n]$. *These are the result of* $a(x)$ *receiving* b *in the two different semantics. We can see that the extrusion history remembers that the input name was* x *before* b *was received, but the keys do not remember, and when reversing the action could use any name as the input name. This does not make a great deal of difference, as after reversing* $a(b)$, *the process with the extrusion history can also* α*-convert* x *to any name.*

Since we intend to define a mapping from processes with extrusion histories to processes with keys, we first describe how to add keys to substituted names in a process in Definition 6.3.2. The function S takes a process, P_1, in which we will be adding keys to the previously substituted names, the key we want to add, $[n]$, the name x which has been substituted and we want to add keys to, and P_2, a previous state of P_1 where the substitution has not yet taken place. We use P_2 to distinguish instances of x created by the substitution we are currently marking from instances of x which existed before the substitution took place.

Definition 6.3.2 (Substituting in πIK-process)**.** *Given a* πIK *process* P_1, *a* πI*-calculus process without keys* P_2, *a key* n, *and a name* x, *we can add the key* n *to any names which* x *has been substituted with, by applying* $S(P_1, P_2, [n], x)$, *defined as:*

1. $S(0, 0, [n], x) = 0$;
2. $S\left(\sum_{i\in I} P_{i1}, \sum_{i\in I} P_{i2}, [n], x\right) = \sum_{i\in I} S\left(P_{i1}, P_{i2}, [n], x\right)$;
3. $S\left(P_1|Q_1, P_2|Q_2, [n], x\right) = S\left(P_1, P_2, [n], x\right)|S\left(Q_1, Q_2, [n], x\right)$;
4. $S\left((\nu a)P_1, (\nu b)P_2, [n], x\right) = P_1'$ *where:*
   if $x = b$ *then* $P_1' = (\nu a)P_1$ *and otherwise* $P_1' = (\nu a)S\left(P_1, P_2, [n], x\right)$.
5. $S\left(\alpha_1.P_1, \alpha_2.P_2, [n], x\right) = \alpha_1'.P_1'$ *where:*
   if $\alpha_2 \in \{x(c), \overline{x}(c)\}$ *then* $\alpha_1' = \alpha_{1_{[n]}}$ *and otherwise* $\alpha_1' = \alpha_1$;
   if $\alpha_2 \in \{c(x), \overline{c}(x)\}$ *then* $P_1' = P_1$ *and otherwise* $P_1' = S\left(P_1, P_2, [n], x\right)$.
6. $S\left(\alpha_1[m].P_1, \alpha_2.P_2, [n], x\right) = \alpha_1'[m].P_1'$ *where:*
   if $\alpha_2 \in \{x(c), \overline{x}(c)\}$ *then* $\alpha_1' = \alpha_{1_{[n]}}$ *and otherwise* $\alpha_1' = \alpha_1$;
   if $\alpha_2 \in \{c(x), \overline{c}(x)\}$ *then* $P_1' = P_1$ *and otherwise* $P_1' = S\left(P_1, P_2, [n], x\right)$.
7. $S\left(!P_1, !P_2, [n], x\right) = !S\left(P_1, P_2, [n], x\right)$;
8. $S\left(P_1|P_1', !P_2, [n], x\right) = S\left(P_1, !P_2, [n], x\right)|S\left(P_1', P_2, [n], x\right)$;
9. $S\left(!P_1, P_2|P_2', [n], x\right) = S\left(!P_1, P_2, [n], x\right)|S\left(P_1, P_2', [n], x\right)$.

where $a(b)_{[n]} = a_{[n]}(b)$ *and* $\overline{a}(b)_{[n]} = \overline{a_{[n]}}(b)$

We also define the root of a πIK process as removing all keys from the process.

Definition 6.3.3 (Root)**.** *We say that a* πIK *process,* P, *has a root,* $\mathsf{rt}(P)$, *defined as:*

$$\mathsf{rt}(0) = 0 \qquad \mathsf{rt}(!P) = \mathsf{rt}(P) \qquad \mathsf{rt}(P_0|P_1) = \mathsf{rt}(P_0)|\mathsf{rt}(P_1) \qquad \mathsf{rt}(\alpha.P) = \alpha.\mathsf{rt}(P)$$
$$\mathsf{rt}(P_0 + P_1) = \mathsf{rt}(P_0) + \mathsf{rt}(P_1) \qquad \mathsf{rt}(\alpha[m].P) = \alpha.\mathsf{rt}(P) \qquad \mathsf{rt}((\nu x)P) = (\nu x)\mathsf{rt}(P)$$

In Lemma 6.3.4 we demonstrate that S does indeed annotate any name, which was was substituted for x_1, with n.

Lemma 6.3.4. *Given a standard πIK-calculus process P, a πIK-calculus process P', a series of substitutions $[x_1 := a_1][x_2 := a_2] \dots [x_k := a_k]$, such that $\mathsf{rt}(P') \equiv P[x_1 := a_1][x_2 := a_2] \dots [x_k := a_k]$ using the definition of $\equiv$ from Section 6.2, and a key $[n]$, we get $S(P', P, [n], x_1) = P''$ for some P'' such that $\mathsf{rt}(P'') \equiv P[x_1 := a_{1_{[n]}}][x_2 := a_2] \dots [x_k := a_k]$.*

Proof. We prove this by structural induction of P in Appendix D.8. □

Being able to annotate our names with keys, we can define a mapping, E, from extrusion histories to keys in Definition 6.3.6, thereby creating a πIK process which has performed the same actions as the πIH process. This is done by E iterating over the extrusions, having one process which builds πIk-process, and another that keeps track of which state of the original πIH process has been reached. When turning an extrusion into a keyed action, we use the locations as key and also give each extrusion an extra copy of its location to use for determining where the action came from. This way we can use one copy to iteratively go through the process, removing splits from the path as we go through them, while still having another intact copy of the location to use as the final key. If E encounters a parallel composition (case 2), it splits its extrusion histories in three. One part, $\mathbf{H}_{\mathsf{shared}}$ contains the locations which have an empty path, and therefore belong to actions from before the processes split. Another part contains the locations beginning with 0, and goes to the first part of the process. And finally the third part contains the locations beginning with 1, and goes to the second part of the process. When turning an input memory from the history into a past input action in the process (case 4), we use S (Definition 6.3.2) to add keys to the substituted names. When E encounters a restriction (case 5), it iteratively moves a memory that can be used inside the restriction inside. It does this iteratively until there are no such memories left in the extrusion histories. We apply E to a process in Example 6.3.7.

Definition 6.3.5. *The function* lcopy *gives each member of an extrusion history an extra copy of its location:*

$$\mathsf{lcopy}(H^*) = \{(\mu, u, u) \mid (\mu, u) \in H^*\}$$
$$\mathsf{lcopy}(\overline{H}, \underline{H}, H) = (\mathsf{lcopy}(\overline{H}), \mathsf{lcopy}(\underline{H}), \mathsf{lcopy}(H))$$

Definition 6.3.6. *Given a πIH process, $\mathbf{H} \vdash P$, we can create an equivalent πIK process, $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) = P'$ defined as*

1. $E((\emptyset, \emptyset, \emptyset) \vdash P, P') = P$

2. $E(\mathbf{H} \vdash P_0 | P_1, P_0' | P_1') = E(\mathbf{H}_{\mathsf{shared}} \vdash P_0'' | P_1'', P_0''' | P_1''')$ *where:*

   $\mathbf{H}_{\mathsf{shared}} = (\{(\alpha, u, u') \mid (\alpha, u, u') \in \overline{H}$ *and* $u \neq iu''\}, \{(\alpha, u, u') \mid (\alpha, u, u') \in \underline{H}$ *and* $u \neq iu''\}, \emptyset)$

   $P_0'' = E((\overline{H_0}, \underline{H_0}, H_0) \vdash P_0, P_0')$ *where:*

   $\overline{H_0} = \{(\overline{a}(b), u_0, u_0') \mid (\overline{a}(b), 0u_0, u_0') \in \overline{H}$ *or* $((\overline{a}(b), \alpha_1), \langle 0u_0, 1u_1\rangle, u_0') \in H\}$

   $\underline{H_0} = \{(a(b), u_0, u_0') \mid (a(b), 0u_0, u_0') \in \underline{H}$ *or* $((a(b), \alpha_1), \langle 0u_0, 1u_1\rangle, u_0') \in H\}$

   $H_0 = \{((\alpha, \alpha'), u, u') \mid ((\alpha, \alpha'), 0u, u') \in H\}$

   $P_1'' = E((\overline{H_1}, \underline{H_1}, H_1) \vdash P_1, P_1'))$ *where:*

   $\overline{H_1} = \{(\overline{a}(b), u_1, u_1') \mid (\overline{a}(b), 1u_1, u_1') \in \overline{H}$ *or* $((\alpha_0, \overline{a}(b)), \langle 0u_0, 1u_1\rangle, u_1') \in H\}$

   $\underline{H_1} = \{(a(b), u_1, u_1') \mid (a(b), 1u_1, u_1') \in \underline{H}$ *or* $((\alpha_0, a(b)), \langle 0u_0, 1u_1\rangle, u_1') \in H\}$

   $H_1 = \{((\alpha, \alpha'), u, u') \mid ((\alpha, \alpha'), 1u, u') \in H\}$

   $\mathbf{H_i} \vdash P_i' \overset{\alpha_{i,0}}{\underset{u_{i,0}}{\rightsquigarrow}} \dots \overset{\alpha_{i,n}}{\underset{u_{i,n}}{\rightsquigarrow}} (\emptyset, \emptyset, \emptyset) \vdash P_i'''$ *for* $i \in \{0, 1\}$

3. $E((\overline{H} \cup \{(\overline{a}(b), [Q][P'], u)\}, \underline{H}, H) \vdash P, P') = E(\mathbf{H} \vdash \overline{a}(b)\,[u]\,.P + \sum_{i \in I \setminus \{j\}} \alpha_i.P_i, Q)$

   if $Q = \sum_{i \in I} \alpha_i.P_i$, $\overline{a}(b) = \alpha_j$, *and* $P' = P_j$

4. $E((\overline{H}, \underline{H} \cup \{(a(b), [Q][P'], u)\}, H) \vdash P, P') = E(\mathbf{H} \vdash a(b)\,[u]\,.S(P, P_j, [u], x) + \sum_{i \in I \setminus \{j\}} \alpha_i.P_i, Q)$

   if $Q = \sum_{i \in I} \alpha_i.P_i$, $a(x) = \alpha_j$, *and* $P' = P_j[x := b]$

5. $E(\mathbf{H} \vdash (\nu x)P, (\nu x)P') = E(\mathbf{H} - (\alpha, u, u') \vdash P'', (\nu x)Q')$ *where* $P'' = (\nu x)E((\emptyset, \emptyset, \emptyset) + (\alpha, u, u') \vdash P, P')$

   if $(\alpha, u, u') \in \overline{H} \cup \underline{H}$ *and* $(\emptyset, \emptyset, \emptyset) + (\alpha, u, u) \vdash P \overset{\alpha}{\underset{u}{\rightarrow}} (\emptyset, \emptyset, \emptyset) \vdash Q'$

6. $E(\mathbf{H} \vdash !P, !P') = E(\mathbf{H} \vdash !P | P, !P' | P')$ *if there exists* $(\alpha, u, u') \in \overline{H} \cup \underline{H} \cup H$ *such that* $u \neq [Q][Q']$.

Example 6.3.7. *We will now apply E to the process*

$$(\{(\overline{b}(c), u_2)\}, \emptyset, \{((b(a), \overline{b}(a)), \langle 0u_0, 1u_1 \rangle)\}) \vdash a(x) \mid 0$$

with locations $u_0 = [b(y).y(x)][a(x)]$, $u_1 = [\overline{b}(a)][0]$, *and* $u_2 = [\overline{b}(c).(b(y).y(x) \mid \overline{b}(a)][b(y).y(x) \mid \overline{b}(a)]$. *We perform*

$$E(\mathsf{lcopy}((\{(\overline{b}(c), u_2)\}, \emptyset, \{((b(a), \overline{b}(a)), \langle 0u_0, 1u_1 \rangle)\})) \vdash a(x) \mid 0, a(x) \mid 0)$$

Since we are at a parallel, we use Case 2 of Definition 6.3.6 to split the extrusion histories into three to get $E((\{(\overline{b}(c), u_2, u_2)\}, \emptyset, \emptyset) \vdash P_0 \mid P_1, b(y).y(x) \mid \overline{b}(a))$ *where* $P_0 = E((\emptyset, \{(b(a), u_0, \langle 0u_0, 1u_1 \rangle)\}, \emptyset) \vdash a(x), a(x))$ *and* $P_1 = E((\{(\overline{b}(a), u_1, \langle 0u_0, 1u_1 \rangle)\}, \emptyset, \emptyset) \vdash 0, 0)$.

To find P_0, *we look at* u_0, *and find that it has* $a(x)$ *as its result, meaning we can apply Case 4. This gives us* $E((\emptyset, \emptyset, \emptyset) \vdash b(a)[\langle 0u_0, 1u_1 \rangle].S(a(x), y(x), [\langle 0u_0, 1u_1 \rangle], y), b(y).y(x))$ *And by applying Case 5 of Definition 6.3.2,* $S(a(x), y(x), [\langle 0u_0, 1u_1 \rangle], y) = a_{[\langle 0u_0, 1u_1 \rangle]}(x)$. *Since we have no more extrusions to add, we apply Case 1 to get our process* $P_0 = b(a)[\langle 0u_0, 1u_1 \rangle].a_{[\langle 0u_0, 1u_1 \rangle]}(x)$.

To find P_1, *we similarly look at* u_1 *and find that we can apply Case 3. This gives us* $P_1 = \overline{b}(a)[\langle 0u_0, 1u_1 \rangle].0$.

We can then apply Case 3 to $E((\{(\overline{b}(c), u_2, u_2)\}, \emptyset, \emptyset) \vdash P_0 \mid P_1, b(y).y(x) \mid \overline{b}(a))$. *This gives us our final process,*

$$\overline{b}(c)[u_2].b(a)[\langle 0u_0, 1u_1 \rangle].a_{[\langle 0u_0, 1u_1 \rangle]}(x) \mid \overline{b}(a)[\langle 0u_0, 1u_1 \rangle].0$$

We can then show, in Theorem 6.3.8, that we have an operational correspondence between our two calculi and E preserves transitions. Item 1 states that every transition in πIH corresponds to one in πIk process generated by E, and Item 2 vice versa.

Theorem 6.3.8. *Given a reachable πIH process, $\mathbf{H} \vdash P$, and an action, μ,*

1. *if there exists a location u such that* $\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightarrow}} \mathbf{H}' \vdash P'$ *then* $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \xrightarrow{\mu[u]} E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$;

2. *if* $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \xrightarrow{\mu[n]} Q$ *then there exists a location, u, and a πIH process,* $\mathbf{H}' \vdash P'$, *such that* $\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightarrow}} \mathbf{H}' \vdash P'$ *and* $P'' \equiv E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$.

Proof. We show both of these by induction on the size of $\overline{H} \cup \underline{H} \cup H$ and the structure of P in Appendix D.9. We use Lemma 6.3.4 to prove that the correct annotations on substitutions are performed in the input case, and Propositions 6.1.5 and 6.2.6 to transfer the result for forward transitions to reverse transitions. □

A correspondence between CCSK and RCCS was used to transfer results between the two calculi in [57]. Our correspondence is not as strong as theirs, since we cannot get the location (or set of causes) of a πIH transition from a πIK transition. We can however use our correspondence to transfer results from πIK to πIH. For example, the forward diamond property of Proposition 6.1.13, implies Proposition 6.3.9.

Proposition 6.3.9 (πIH forward diamond). *Given reachable processes* $\mathbf{H}_P \vdash P$, $\mathbf{H}_Q \vdash Q$, *and* $\mathbf{H}_R \vdash R$, *if* $\mathbf{H}_P \vdash P \xrightarrow[u_0]{\mu_0} \mathbf{H}_Q \vdash Q$, $\mathbf{H}_P \vdash P \xrightarrow[u_1]{\mu_1} \mathbf{H}_R \vdash R$, *and there exists a process* $\mathbf{H}_T \vdash T$ *such that* $\mathbf{H}_Q \vdash Q \to^* \mathbf{H}_T \vdash T$ *and* $\mathbf{H}_R \vdash R \to^* \mathbf{H}_T \vdash T$, *then there exists processes* $\mathbf{H}_S \vdash S$ *and* $\mathbf{H}'_S \vdash S'$ *such that* $\mathbf{H}_Q \vdash Q \xrightarrow[u_1]{\mu_1} \mathbf{H}_S \vdash S$ *and* $\mathbf{H}_R \vdash R \xrightarrow[u_0]{\mu_0} \mathbf{H}'_S \vdash S'$ *and* $E(\mathsf{lcopy}(\mathbf{H}_S) \vdash S, S) \equiv E(\mathsf{lcopy}(\mathbf{H}'_S) \vdash S', S')$ *and* $\mathbf{H}_S \vdash S \to^* \mathbf{H}_T \vdash T$ *and* $\mathbf{H}'_S \vdash S' \to^* \mathbf{H}_T \vdash T$.

Proof. By Theorem 6.3.8, $E(\mathsf{lcopy}(\mathbf{H}_P) \vdash P, P) \xrightarrow{\mu_0[m_0]} E(\mathsf{lcopy}(\mathbf{H}_Q) \vdash Q, Q)$ and $E(\mathsf{lcopy}(\mathbf{H}_P) \vdash P, P) \xrightarrow{\mu_1[m_1]} E(\mathsf{lcopy}(\mathbf{H}_R) \vdash R, R)$ and $E(\mathsf{lcopy}(\mathbf{H}_Q) \vdash Q, Q) \to^* E(\mathsf{lcopy}(\mathbf{H}_T) \vdash T, T)$ and $E(\mathsf{lcopy}(\mathbf{H}_R) \vdash R, R) \to^* E(\mathsf{lcopy}(\mathbf{H}_T) \vdash T, T)$. By Proposition 6.1.13, we get an S' such that $E(\mathsf{lcopy}(\mathbf{H}_Q) \vdash Q, Q) \xrightarrow{\mu_1[m_1]} S'$ and $E(\mathsf{lcopy}(\mathbf{H}_R) \vdash R, R) \xrightarrow{\mu_0[m_0]} S'$ and by Theorem 6.3.8 we get $\mathbf{H}_Q \vdash Q \xrightarrow[u'_1]{\mu_1} \mathbf{H}_S \vdash S$ and $\mathbf{H}_R \vdash R \xrightarrow[u'_0]{\mu_0} \mathbf{H}'_S \vdash S'$ such that $E(\mathsf{lcopy}(\mathbf{H}_S) \vdash S, S) \equiv E(\mathsf{lcopy}(\mathbf{H}'_S) \vdash S', S')$, and by Lemma 6.2.12, $u_0 = u'_0$ and $u_1 = u'_1$. And by Theorem 6.3.8 and Proposition 6.1.13, $\mathbf{H}_S \vdash S \to^* \mathbf{H}_T \vdash T$ and $\mathbf{H}'_S \vdash S' \to^* \mathbf{H}_T \vdash T$. □

We now have the first result showing a correspondence between a statically and dynamically reversible π-calculus, and have shown that this result is useful in proving that at least some of the properties of one calculus also hold for the other. Although our encoding is not between full π-calculi, it still tackles a number of the issues that would come with mapping between those, such as the memory of which names have been substituted being stored more directly in the statically reversible calculus. This is also the first time a calculus with a centralised memory has been encoded in a statically reversible calculus, since [57, 68] used RCCS, which has distributed memories as their chosen dynamically reversible calculus.

6.4 Denotational Event Structure Semantics of πIK-Calculus

In this section we define event structure semantics of πIK-calculus using a similar denotational approach to what we did for CCSK in Section 5.1 and Roll-CCSK in Section 5.3.

As we want to ensure that all free and bound names in our process are distinct, we modify our syntax for replication, assigning each replication an infinite set, $\mathbf{x}$, of names to substitute into the place of bound names in each created copy of the process, so that

$$!_{\mathbf{x}}P \equiv !_{\mathbf{x}\setminus\{x_0,\ldots,x_k\}}P|P\{^{x_0,\ldots,x_k}/_{a_0,\ldots,a_k}\} \text{ if } \{x_0,\ldots,x_k\} \subseteq \mathbf{x} \text{ and } \mathsf{bn}(P) = \{a_0,\ldots,a_k\}$$

Before proceeding to the semantics we also define the standard bound names of a process P, $\mathsf{sbn}(P)$, meaning the names that would be bound in P if every action was reversed, in Definition 6.4.1.

Definition 6.4.1. *The standard bound names of a process* P, $\mathsf{sbn}(P)$, *are defined as:*

$$\begin{array}{ll}
\mathsf{sbn}(a(x).P') = \{x\} \cup \mathsf{sbn}(P') & \mathsf{sbn}(a(x)[m].P') = \{x\} \cup \mathsf{sbn}(P') \\
\mathsf{sbn}(\overline{a}(x).P') = \{x\} \cup \mathsf{sbn}(P') & \mathsf{sbn}(\overline{a}(x)[m].P') = \{x\} \cup \mathsf{sbn}(P') \\
\mathsf{sbn}(P_0|P_1) = \mathsf{sbn}(P_0) \cup \mathsf{sbn}(P_1) & \mathsf{sbn}(P_0 + P_1) = \mathsf{sbn}(P_0) \cup \mathsf{sbn}(P_1) \\
\mathsf{sbn}(\nu x)P' = \{x\} \cup \mathsf{sbn}(P') & \mathsf{sbn}(!_{\mathbf{x}}P) = \mathbf{x}
\end{array}$$

We can now define the event structure semantics in Table 6.6. We do this using rules of the form $\{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ where l is the level of unfolding of replication, $\mathcal{E}$ is an LRBES, Init is the initial configuration, $\mathcal{N} \supseteq n(P)$ is a set of names, which any input in the process could receive, and $k : \mathsf{Init} \to \mathcal{K}$ is a function assigning communication keys to the past actions, which we use in parallel composition to determine which synchronisations of past actions to put in Init. We define $\{\!|P|\!\}_{\mathcal{N}} = \sup_{l\in\mathbb{N}} \{\!|P|\!\}_{(\mathcal{N},l)}$

Most of the cases in Table 6.6 are straightforward uses of the RBES operators defined in Section 4.4, 5.1 and 5.3. The input creates a case for each name in $\mathcal{N}$ and a choice between the cases. We have two cases for restriction, one for restriction originating from a past communication and another for restriction originating from the original process. The parallel composition is more complex as it needs to consider link causation caused by the early semantics. Each event labelled with an input of a name in standard bound names gets

a bundle consisting of the event labelled with the output on that name. And each output event is prevented from reversing by the input names receiving that name.

Note that the only difference between a future and a past action is that the event corresponding to a past action is put in the initial state and given a communication key.

Example 6.4.2. *Consider the process* $a(b)[n] \mid \overline{a}(b)[n]$. *Our event structure semantics generate a LRBES* $\{\!|a(x)[n] \mid \overline{a}(b[n])|\!\}_{\{a,b,x\}} = \langle (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$ *where:*

$$
\begin{array}{ll}
E = F = \{a(b), a(a), a(x), \overline{a}(b), \tau\} & \lambda(e) = \ e \\
\{\overline{a}(b)\} \mapsto a(b) & \mathsf{Act} = \ \{a(b), a(a), a(x), \overline{a}(b), \tau\} \\
a(b) \sharp a(a),\ a(b) \sharp a(x),\ a(a) \sharp a(x), & \mathsf{Init} = \ \{\tau\} \\
a(b) \sharp \tau,\ a(a) \sharp \tau,\ a(x) \sharp \tau,\ \overline{a}(b) \sharp \tau & k(\tau) = \ n \\
a(b) \rhd \overline{a}(b) &
\end{array}
$$

From this we see that (1) receiving b is causally dependent on sending b, (2) all the possible inputs on a are in conflict with one another, (3) the synchronisation between the input and the output is in conflict with either happening on their own, and (4) since the two past actions have the same key, the initial state contains their synchronisation.

As we did for CCSK in Proposition 5.1.13, we show that there exists a least upper bound of our unfolding in Proposition 6.4.7. This again requires us to prove that each of our operations is monotonic in Lemmas 6.4.3 to 6.4.6.

Lemma 6.4.3. *Let* P *be a consistent process and* $\mathcal{N} \supseteq n(P)$ *be a set of names such that* $\{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $\{\!|P|\!\}_{(\mathcal{N},l+1)} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$, *and* $\mathcal{E}_0 \leq \mathcal{E}_1$. *Then for any process* P', *where* $\{\!|P'|\!\}_{(\mathcal{N},l)} = \{\!|P'|\!\}_{(\mathcal{N},l)+1}$ *and* $n(P') \subseteq \mathcal{N}$, *if* $\{\!|P + P'|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$ *and* $\{\!|P + P'|\!\}_{(\mathcal{N},l+1)} = \langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1\rangle$, *then* $\mathcal{E}'_0 \leq \mathcal{E}'_1$.

Proof. Follows from the semantics of choice in Table 6.6. □

Lemma 6.4.4. *Let* P *be a consistent process and* $\mathcal{N} \supseteq n(P)$ *be a set of names such that* $\{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $\{\!|P|\!\}_{(\mathcal{N},l+1)} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$, *and* $\mathcal{E}_0 \leq \mathcal{E}_1$. *Then for any action* α, *if* $\{\!|\alpha.P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$ *and* $\{\!|\alpha.P|\!\}_{(\mathcal{N},l+1)} = \langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1\rangle$, *then* $\mathcal{E}'_0 \leq \mathcal{E}'_1$.

Proof. Follows from the semantics of action prefix in Table 6.6. □

Lemma 6.4.5. *Let* P *be a consistent process and* $\mathcal{N} \supseteq n(P)$ *be a set of names such that* $\{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $\{\!|P|\!\}_{(\mathcal{N},l+1)} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$, *and* $\mathcal{E}_0 \leq \mathcal{E}_1$. *Then for any process* P' *with* $n(P') \subseteq \mathcal{N}$, *if* $\{\!|P|P'|\!\}(\mathcal{N}, l) = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$ *and* $\{\!|P|P'|\!\}(\mathcal{N}, l+1) = \langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1\rangle$, *then* $\mathcal{E}'_0 \leq \mathcal{E}'_1$.

Proof. Let $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$. Then $\mathcal{E}_0 \times \mathcal{E}_2 \leq \mathcal{E}_1 \times \mathcal{E}_2$ and for any $(e, *), (*, e') \in E'_0$, there exist names $x \in no(\lambda'_0((e, *)))$ and a such that $\lambda'_0((*, e')) = \overline{a}(x)$ if and only if $x \in no(\lambda'_1((e, *)))$ and $\lambda'_1((*, e')) = \overline{a}(x)$. Additionally, the standard bound names of P_0 and P_1. are the same. □

Lemma 6.4.6. *Let* P_0 *and* P_1 *be consistent processes and* $\mathcal{N} \supseteq n(P_0) \cup n(P_1)$ *be a set of names such that* $\{\!|P_0|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $\{\!|P_1|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$, *and* $\mathcal{E}_0 \leq \mathcal{E}_1$. *Then for any name* n, *if* $\{\!|(\nu n)P_0|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$ *and* $\{\!|(\nu n)P_1|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1\rangle$, *then* $\mathcal{E}'_0 \leq \mathcal{E}'_1$.

Proof. Follows from the semantics of restriction in Table 6.6. □

Using these lemmas, we can show that $\{\!|P|\!\} = \sup_{l \in \mathcal{N}} \{\!|P|\!\}_{(\mathcal{N},l)}$ exists for any P because we can order event structures by the level of unfolding of P they were generated by.

Proposition 6.4.7 (Unfolding)**.** *Given a process* P, *a set of names* $\mathcal{N} \supseteq n(P)$, *and a level of unfolding* l, *if* $\{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ *and* $\{\!|P|\!\}_{(\mathcal{N},l-1)} = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$, *then* $\mathcal{E}' \leq \mathcal{E}$, $\mathsf{Init} = \mathsf{Init}'$, *and* $k = k'$.

Proof. Follows from Lemmas 6.4.3 to 6.4.6, and $\mathcal{E} \times (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \cong \mathcal{E}$. □

$$
\begin{array}{ll}
\{\!|0|\!\}_{(\mathcal{N},l)} = & \langle(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset),\emptyset,\emptyset\rangle \\
\{\!|P_0+P_1|\!\}_{(\mathcal{N},l)} = & \langle \mathcal{E}_0+\mathcal{E}_1, (\{0\}\times \mathsf{Init}_0)\cup(\{1\}\times \mathsf{Init}_1), k((i,e)) = k_i(e)\rangle \text{ where} \\
& \{\!|P_i|\!\} = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i\rangle \text{ for } i\in\{0,1\} \\
\left\{\!\left|\overline{a}(n).P\right|\!\right\}_{(\mathcal{N},l)} = & \langle \overline{a}(n)(e).\mathcal{E}_P, \mathsf{Init}_P, k_P\rangle \text{ for some fresh } e\notin E \text{ where} \\
& \{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_P, \mathsf{Init}_P, k_P\rangle \\
\{\!|a(x).P|\!\}_{(\mathcal{N},l)} = & \langle \sum\limits_{n\in(\mathcal{N}\setminus \mathsf{sbn}(P))} a(n)(e_n).\mathcal{E}_{P_n}, \bigcup\limits_{n\in(\mathcal{N}\setminus \mathsf{sbn}(P))} \{n\}\times \mathsf{Init}_{P_n}, (n,e)\mapsto k_{P_n}(e)\rangle \\
& \text{for some fresh } e_n\notin E_n \text{ where } \{\!|P[x:=n]|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_{P_n}, \mathsf{Init}_{P_n}, k_{P_n}\rangle \\
\left\{\!\left|\overline{a}(n)[m].P\right|\!\right\}_{(\mathcal{N},l)} = & \langle \overline{a}(n)(e).\mathcal{E}_P, \mathsf{Init}_P\cup\{e\}, k_P[e\mapsto m]\rangle \text{ for some fresh } e\notin E \text{ where} \\
& \{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_P, \mathsf{Init}_P, k_P\rangle \text{ and} \\
\{\!|a(b)[m].P|\!\}_{(\mathcal{N},l)} = & \langle \sum\limits_{n\in(\mathcal{N}\setminus \mathsf{sbn}(P))} a(n)(e_n).\mathcal{E}_{P_n}, (\bigcup\limits_{n\in(\mathcal{N}\setminus \mathsf{sbn}(P))} \{n\}\times \mathsf{Init}_{P_n})\cup\{(b,e_b)\}, k\rangle \\
& \text{for some fresh } e_n\notin E_n \text{ where } \{\!|P[b_{[m]}:=n]|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_{P_n}, \mathsf{Init}_{P_n}, k_{P_n}\rangle \\
& k((n,e)) = \begin{cases} m & \text{if } e=e_b \text{ and } n=b \\ k_{P_n}(e) & \text{otherwise}\end{cases} \\
\{\!|(\nu a)P|\!\}_{(\mathcal{N},l)} = & \langle \mathcal{E}\upharpoonright E_\alpha, \mathsf{Init}\cap E_\alpha, k\upharpoonright E_\alpha\rangle \text{ where:} \\
& \{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}, \mathsf{Init}, k\rangle \qquad E_\alpha = \rho(\{\alpha \mid a\in n(\alpha)\}) \\
& \text{if whenever there exist past actions } b(a)[m] \text{ and } \overline{b}(a)[m] \text{ in } P \text{ then} \\
& \text{they are guarded by a restriction } (\nu a) \text{ in } P \\
\{\!|(\nu a)P|\!\}_{(\mathcal{N},l)} = & \langle \mathcal{E}, \mathsf{Init}, k\rangle \text{ where } \{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}, \mathsf{Init}, k\rangle \text{ and} \\
& \text{if there exist past actions } b(a)[m] \text{ and } \overline{b}(a)[m] \text{ in } P \text{ which} \\
& \text{are not guarded by a restriction } (\nu a) \text{ in } P \\
\{\!|P_0|P_1|\!\}_{(\mathcal{N},l)} = & \langle (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act})\upharpoonright\{e\mid\lambda(e)\neq 0\}, \mathsf{Init}, k\rangle \text{ where} \\
& \text{for } i\in\{0,1\},\ \{\!|P_i|\!\}_l = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i\rangle \\
& (E_0,F_0,\mapsto_0,\sharp_0,\rhd_0)\times(E_0,F_0,\mapsto_0,\sharp_0,\rhd_0) = (E,F,\mapsto',\sharp,\rhd') \\
& \pi_0,\ \pi_1 \text{ are functions such that } \pi_i(e_0,e_1) = e_i \text{ and} \\
& \lambda(e) = \begin{cases} \lambda_0(e_0) & \text{if } e=(e_0,*) \\ \lambda_1(e_1) & \text{if } e=(*,e_1) \\ \tau & \text{if } e=(e_0,e_1) \text{ and for } i\in\{0,1\},\ \lambda_i(e_i)=a(x) \text{ and } \lambda_{1-i}(e_{1-i})=\overline{a}(x) \\ 0 & \text{otherwise}\end{cases} \\
& \mathsf{Act} = \{\tau\}\cup\mathsf{Act}_0\cup\mathsf{Act}_1 \\
& \mathsf{Init} = \{(e_0,*)|e_0\in\mathsf{Init}_0 \text{ and } \nexists e_1\in\mathsf{Init}_1.k_1(e_1)=k_0(e_0)\}\cup \\
& \{(*,e_1)|e_1\in\mathsf{Init}_1 \text{ and } \nexists e_0\in\mathsf{Init}_0.k_1(e_1)=k_0(e_0)\}\cup \\
& \{(e_0,e_1)|e_0\in\mathsf{Init}_0 \text{ and } e_1\in\mathsf{Init}_1 \text{ and } k_1(e_1)=k_0(e_0)\} \\
& X\mapsto e \text{ if } X\mapsto' e \text{ or there exists } x\in no(\lambda(e)) \text{ such that} \\
& X=\{e'\mid\exists a.\lambda(e')=\overline{a}(x)\} \text{ and } x\in\mathsf{sbn}(P) \\
& e\rhd \underline{e'} \text{ if either } e\rhd'\underline{e'} \text{ or there exists } x\in no(\lambda(e)) \text{ such that } \exists a.\lambda(e')=\overline{a}(x) \\
& k(e) = \begin{cases} k_0(e_0) & \text{if } e=(e_0,*) \\ k_1(e_1) & \text{if } e=(*,e_1) \\ k_0(e_0) & \text{if } e=(e_0,e_1)\end{cases} \\
\{\!|!_{\mathbf{x}}P|\!\}_{(\mathcal{N},0)} = & \langle(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset),\emptyset,\emptyset\rangle \\
\{\!|!_{\mathbf{x}}P|\!\}_{(\mathcal{N},l)} = & \left\{\!\left|!_{\mathbf{x}\setminus\{x_0,\ldots,x_k\}}P|P\{^{x_0,\ldots,x_k}/_{a_0,\ldots,a_k}\}\right|\!\right\}_{(\mathcal{N},l-1)} \text{ if } \{x_0,\ldots,x_k\}\subseteq\mathbf{x} \text{ and } \mathsf{bn}(P)=\{a_0,\ldots,a_k\}
\end{array}
$$

Table 6.6: Denotational event structure semantics of πIK

Proposition 6.4.8. *Let P be a forwards-reachable process wherein all bound and free names are different and $\mathcal{N} \supseteq n(P)$ be a set of names. If $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$ then $\mathcal{E}$ is causal.*

Proof. Follows from the event structure semantics and structural induction of P. □

Proposition 6.4.9 shows that structurally congruent processes generate isomorphic event structures.

Proposition 6.4.9 (Structural Congruence)**.** *Given processes P and P' and a set of names $\mathcal{N} \supseteq n(P) \cup n(P')$, if $P \equiv P'$, $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$, and $\{\!|P'|\!\}_{\mathcal{N}} = \langle \mathcal{E}', \mathsf{Init}', k' \rangle$, then there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $f(\mathsf{Init}) = \mathsf{Init}'$ and for all $e \in \mathsf{Init}$, $k(e) = k'(f(e))$.*

Proof. We prove this by case analysis on the structural congruence rules in Appendix D.10. □

We show in Theorems 6.4.11 and 6.4.12 that given a process P with a conflict-free initial state, including any reachable process, there exists a transition $P \xrightarrow{\mu[m]} P'$ if and only if the event structure corresponding to P is isomorphic to the event structure corresponding to P' and an event labelled μ is available in P's initial state, and P''s initial state is P's initial state with this event added.

Lemma 6.4.10. *Given a forwards-reachable process P and a set of names $\mathcal{N} \supseteq n(P)$ with $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$. Then $\mathsf{Init} = \emptyset$ if and only if $\mathsf{std}(P)$.*

Proof. The only rules in the event structure semantics which add events to an empty initial set involve past actions, and the only rule removing events from the initial state requires the event to be behind a restriction not created by a communication. □

Theorem 6.4.11. *Let P be a forwards-reachable process wherein all bound and free names are different and let $\mathcal{N} \supseteq n(P)$ be a set of names. If (1) $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$ where $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, and Init is conflict-free, and (2) there exists a transition $P \xrightarrow{\mu[m]} P'$ such that $\{\!|P'|\!\}_{\mathcal{N}} = \langle \mathcal{E}', \mathsf{Init}', k' \rangle$, then there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ and a transition in $\mathcal{C}_{br}(\mathcal{E})$, $\mathsf{Init} \xrightarrow{\{e\}} X$, such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

Proof. We prove the theorem by induction on $P \xrightarrow{\mu[m]} P'$ in Appendix D.11 using Proposition 6.4.9. This proof is similar to Theorem 5.1.18, but the input case uses the coproduct of input on all names in $\mathcal{N}$ to ensure that whatever name is received in μ has a corresponding input event. □

Theorem 6.4.12. *Let P be a forwards-reachable process wherein all bound and free names are different and let $\mathcal{N} \supseteq n(P)$ be a set of names. If (1) $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$ where $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, and (2) there exists a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ in $\mathcal{C}_{br}(\mathcal{E})$, then there exists a transition $P \xrightarrow{\mu[m]} P'$ such that $\{\!|P'|\!\}_{\mathcal{N}} = \langle \mathcal{E}', \mathsf{Init}', k' \rangle$ and an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

Proof. We prove this by structural induction on P in Appendix D.12. This proof is similar to Theorem 5.1.19 but the parallel composition case has to ensure that no input events of names bound elsewhere can happen. □

Corollary 6.4.13. *Let P be a consistent process and $\mathcal{N} \supseteq n(P)$ be a set of names such that $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$. Then Init is forwards-reachable in $\mathcal{E}$ if and only if there exists a standard process Q such that $Q \to^* P$.*

Proof. Follows from Lemma 6.4.10 and Theorems 6.4.12 and 6.4.11. □

We then prove the same operational correspondence on reverse transitions, with the event being removed rather than added to the initial state.

Theorem 6.4.14. *If $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $\mathcal{C}_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \to)$,$\mathsf{Init}$ is conflict-free, and there exists a transition $P \overset{\mu[m]}{\rightsquigarrow} P'$ such that $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k' \rangle$, then there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ and a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

Proof. Implied by Proposition 6.4.8, Theorem 6.4.11, and Corollary 6.4.13. □

Theorem 6.4.15. *If* $[\![P]\!] = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $\mathcal{C}_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, Init *is conflict-free, and there exists a transition* $\mathsf{Init} \xrightarrow{e} X$ *in* $\mathcal{C}_{br}(\mathcal{E})$*, then there exists a a key* m *and a transition* $P \overset{\lambda(e)[m]}{\rightsquigarrow} P'$*, such that* $[\![P']\!] = \langle \mathcal{E}', \mathsf{Init}', k'\rangle$ *and there exist isomorphisms* $f : \mathcal{E} \to \mathcal{E}'$ *and* $g : \mathcal{E}' \to \mathcal{E}$ *such that* $f \circ k' = k[e \mapsto m]$ *and* $f(X) = \mathsf{Init}'$*.*

Proof. Implied by Proposition 6.4.8, Theorem 6.4.12, and Corollary 6.4.13. □

As we did for CCSK in Section 5.1, we have now defined denotational event structure semantics and proved operational correspondence. We can see that the two primary differences between the event structure semantics of πIK and CCSK are that when modelling input πIK needs to create a choice between every possible input and substitution, and in the parallel composition, πIK ensures that these substituted input events are caused by any parallel output events which make the received name free. Interestingly, both of these differences are caused by the semantics being early, which leads us to believe that had we done a late semantics, we would be able to translate a CCSK process to the late πIK-calculus and get functionally the same event structure semantics as the original process.

6.5 Operational Event Structure Semantics of πIH-Calculus

In this section, we define event structure semantics of the πIH calculus. Unlike the denotational event structure semantics we have previously defined, these operational semantics generate event structures from the transitions rather than the syntax.

We generate a labelled asynchronous transition system (LATS) (Definition 6.5.1) based on the causal semantics of πIH.

Definition 6.5.1 (Labelled asynchronous transition system [15])**.** *An Labelled asynchronous transition system is a tuple* $(S, i, E, \mathcal{T}, I, lab, \mathcal{A})$*, where*

- $(S, i, E, \mathcal{T})$ *is a transition system with the set of states* S*, the initial state* i*, the set of events* E*, and the transition relation* $\mathcal{T} \subseteq S \times E \times S$*;*
- $lab : E \to \mathcal{A}$ *is the labelling function from the set of events to the set of actions;*
- $I \subseteq E \times E$ *is the irreflexive, symmetric independence relation satisfying:*
  - $e \in E \Rightarrow \exists s, s' \in S.(s, e, s') \in \mathcal{T}$*;*
  - $(s, e, s') \in \mathcal{T} \wedge (s, e, s'') \in \mathcal{T} \Rightarrow s' = s''$*;*
  - $e_0 I e_1 \wedge \{(s, e_0, s_0), (s, e_1, s_1)\} \subseteq \mathcal{T} \Rightarrow \exists s_2.\{(s_0, e_1, s_2), (s_1, e_0, s_2)\} \subseteq \mathcal{T}$*;*
  - $e_0 I e_1 \wedge \{(s, e_0, s_0), (s_0, e_1, s_2)\} \subseteq \mathcal{T} \Rightarrow \exists s_1.\{(s, e_1, s_1), (s_1, e_0, s_2)\} \subseteq \mathcal{T}$*.*

Definition 6.5.2 (Generated transition system [52])**.** *A* πIH *process generates a transition system* $TS = (S, i, E, \mathcal{T}, I, lab, \mathcal{A})$ *defined as follows:*

- *The set of states,* S *contains equivalence classes with respect to* $\equiv$ *as defined in Section 6.2 of pairs of histories and processes:*
  $S = \left\{ \mathbb{P} \mid \forall \mathbf{H} \vdash P, \mathbf{H}' \vdash P' \in \mathbb{P} : \mathbf{H} = \mathbf{H}' \text{ and } P \equiv P' \right\}$*;*
- *the set of events,* E*, consists of triples of actions, locations, and causes:*
  $E = \left\{ (\alpha, u, D) \middle| \exists \mathbf{H}, P, \mathbf{H}', P' : \mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H}' \vdash P' \right\}$
  which are labelled with their action, α*;*
- $\mathcal{T} \subseteq S \times E \times S$ *are the transitions from Definition 6.2.7.*
  $\mathcal{T} = \left\{ \mathbb{P}, (\alpha, u, D), \mathbb{P}') \middle| \exists \mathbf{H} \vdash P \in \mathbb{P}, \mathbf{H}' \vdash P' \in \mathbb{P}' : \mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H} \vdash P' \right\}$*;*

- *events* (α_0, u_o, D_0) *and* (α_1, u_1, D_1) *are independent if* $\overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}}$ *and* $\overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}}$ *are independent according to Definition 6.2.9.*

We use action, location, *and* causes *respectively to extract the action, location, and causes from events.*

For a particular process P*, we select the initial state* $i = \{(\emptyset, \emptyset, \emptyset) \vdash P\}_{\equiv}$*, and restrict the transition system to the part reachable from this state to get* $TS(P)$*.*

Theorem 6.5.3. *TS is an LATS.*

Proof. I is clearly irreflexive and symmetric, based on Definition 6.2.9.

We prove the three requirements on the definition of an LATS:

1. If $(\alpha, u, D) \in E$, then there exist processes $\mathbf{H} \vdash P$ and $\mathbf{H}' \vdash P'$ such that $\mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H}' \vdash P'$ by definition.

2. If $\mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H}' \vdash P'$ and $\mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H}'' \vdash P''$ then we can use similar arguments to Theorem 4.2 of [52] to show that $P' \equiv P''$ and $\mathbf{H}' = \mathbf{H}''$. Unlike [52], we do not need to take into account a (Rep) rule, eliminating that case, but instead a [STR] rule, which we do by only requiring $P' \equiv P''$, rather than $P' = P''$.

3. If $\mathbf{H} \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_0 \vdash P_0$, $\mathbf{H} \vdash P \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_1 \vdash P_1$ and (α_0, u_0, D_0) and (α_1, u_1, D_1) are independent then by Proposition 6.2.10, we get $\mathbf{H}_0 \vdash P_0 \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_2 \vdash P_2$ and $\mathbf{H}_1 \vdash P_1 \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_2 \vdash P_2$ for some $\mathbf{H}_2 \vdash P_2$.

4. If $\mathbf{H} \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_0 \vdash P_0$, $\mathbf{H}_0 \vdash P_0 \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_2 \vdash P_2$ and (α_0, u_0, D_0) and (α_1, u_1, D_1) are independent then we can use similar arguments to Theorem 4.2 of [52] to show that $\mathbf{H} \vdash P \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_1 \vdash P_1$ and $\mathbf{H}_1 \vdash P_1 \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_2 \vdash P_2$ for some $\mathbf{H}_1 \vdash P_1$.

□

Proving that the forwards semantics generate an LATS is the key steps in defining the operational event structure semantics. Since πIH uses causal reversibility, we can from here use existing mappings to get a causally reversible event structure.

In Definition 6.5.2, we showed how to generate an LATS using our extrusion history-based semantics. In [96], a mapping from LATSs to labelled prime event structures (LPESs) via trace languages was defined, which we can use this to generate event structures from πIH processes. An example of this applied to a πIH-calculus process can be seen in Example 6.5.5. Since events are sets of traces, independent actions with conflicting causes become conflicting events.

Definition 6.5.4 (From LATS to LPES [96])**.** *Given an LATS* $\mathcal{T} = (S, i, E, \mathcal{T}, I, lab, \mathcal{A})$*, we can generate a labelled prime event structure* $LP(\mathcal{T}) = (E', <, \sharp, \lambda, \mathcal{A})$ *where:*

- E' *is the equivalence classes of non-empty forward-only traces of* $\mathcal{T}$ *with respect to* $\approx$*, where* $\approx$ *is the smallest equivalence relation on non-empty traces such that* $ta \approx tba$ *if* bIa *and* $sa \approx ta$ *if* $s \eqsim t$*.*

  Where $\eqsim$ *is the smallest equivalence relation such that* $sabt \eqsim sbat$ *if* aIb*.*

- *Given* $e, e' \in E'$*,* $e < e'$ *if there exists a trace* $s \in e$ *such that for all* $t \in e'$*, there exists* $t' \eqsim t$ *such that* s *is a prefix of* t'*.*

- *Given* $e, e' \in E'$*,* $e \sharp e'$ *if there exist* $e_0 \leq e$ *and* $e'_0 \leq e'$ *such that there exist traces* $t_0 a \in e_0$ *and* $t_0 b \in e'_0$ *such that* $a \neq b$ *and* $\neg(aIb)$*.*

- *For* $e \in E'$*, if* $t.a \in e$ *then* $\lambda(e) = lab(a)$*.*

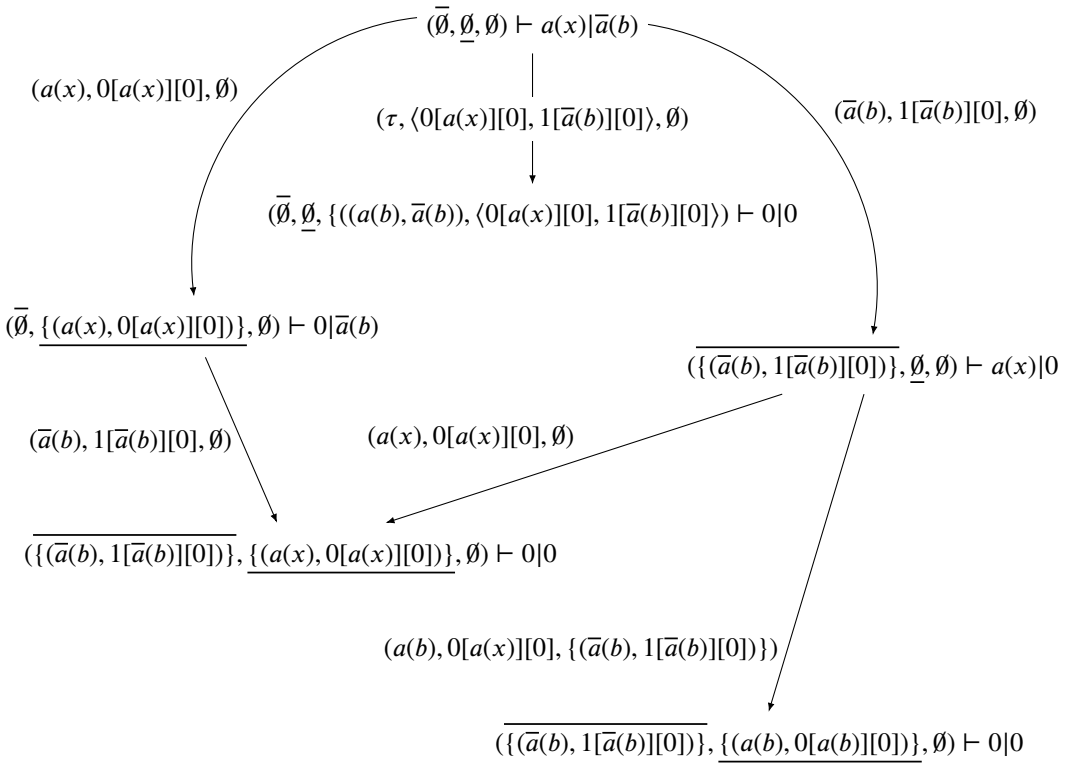


Figure 6.3: The LATS generated in Example 6.5.5

Example 6.5.5. *Consider the process* $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash a(x)|\overline{a}(b)$. *This generates the LATS seen in Figure 6.3 (where we leave out traces using* α*-conversion and only consider one possible free name as input, as otherwise there would infinitely many events and states), with the independence relation consisting of the pair* $(a(x), 0[a(x)][0], \emptyset) I (\overline{a}(b), 1[\overline{a}(b)][0], \emptyset)$.

This gives us the events:

1. $\{(a(x), 0[a(x)][0], \emptyset);\ (\overline{a}(b), 1[\overline{a}(b)][0], \emptyset), (a(x), 0[a(x)][0], \emptyset)\}$
2. $\{(\overline{a}(b), 1[\overline{a}(b)][0], \emptyset);\ (a(x), 0[a(x)][0], \emptyset), (\overline{a}(b), 1[\overline{a}(b)][0], \emptyset)\}$
3. $\{(\tau, \langle 0[a(x)][0], 1[\overline{a}(b)][0]\rangle, \emptyset)\}$
4. *and* $\{(\overline{a}(b), 1[\overline{a}(b)][0], \emptyset), (a(b), 0[a(x)][0], \{(\overline{a}(b), 1[\overline{a}(b)][0])\})\}$

With $2 < 4$, $1 \sharp 3$, *and* $2 \sharp 3$.

Because 4 *is the only event which does not contain a trace of length 1, it is also the only event with a cause. Events* 1 *and* 2 *are in conflict with* 3 *because* $0[a(x)][0]$ *and* $1[\overline{a}(b)][0]$ *are not independent of* $\langle 0[a(x)][0], 1[\overline{a}(b)][0]\rangle$.

In the denotational event structure semantics 1 *would be an event labelled* $a(x)$, 2 *would be an event labelled* $\overline{a}(b)$, 3 *would be an event labelled* τ, *and* 4 *would be an event labelled* $a(b)$.

Finally, since we are modelling a reversible calculus, we need to turn our LPES into a causal labelled reversible prime event structure (CLRPES).

Definition 6.5.6 (From LPES to CLRPES [50])**.** *Given an LPES,* $(E, <, \sharp, \lambda, \mathsf{Act})$, *generated from a process, we define* $LR((E, <, \sharp, \lambda, \mathsf{Act})) = (E, F, <, \sharp, \prec, \rhd, \lambda, \mathsf{Act})$ *where* $P'_p((E, <, \sharp)) = (E, F, <, \sharp, \prec, \rhd)$.

We now have a mapping in three steps to go from a πIH-calculus process to a CLRPES as shown in Figure 6.1. And from the previous section, we have denotational event structure semantics of πIK. It is therefore time to compare the result of using the two semantics on corresponding processes.

6.6 Correspondence Between Denotational and Operational Event Structure Semantics

With event structure semantics defined for two variants of reversible πI-calculus, we will now compare them. Example 6.6.1 shows the difference between how our processes are represented by LRBESs and LRPESs, and illustrates the difficulty of mapping LRPESs to LRBESs in a way, that gives us the intended result. We therefore create a function to turn the CLRBESs into CLRPESs (Definition 6.6.4). We do this by splitting each event into multiple events, based on the different sets of *possible causes*.

Example 6.6.1 (Turning a LRPES into a LRBES). *The LRPES generated by the process* $(\emptyset,\emptyset,\emptyset) \vdash \overline{a}(b).b(x)|a(y)$ *has two events labelled* $b(x)$*, one which is caused by an event labelled* $\overline{a}(b)$*, and another caused by an event labelled* τ*, representing the communication on* a*. The obvious way to represent this in an LRBES would be to keep two events labelled* $b(x)$ *and say* $\{\overline{a}(b)\} \mapsto b(x)_1$ *and* $\{\tau\} \mapsto b(x)_2$*.*

However, if we look at $\{\!|\overline{a}(b).b(x)|a(y)|\!\}_{\{a,b,x,y\}}$*, we get one event labelled* $b(x)$*, with* $\{\overline{a}(b),\tau\} \mapsto b(x)$*.*

These RBESs have the same configurations and transitions (up to labels), but are not isomorphic, and determining which events and bundles need to be merged from a larger RPES would be difficult.

We define the possible causes of an event in a CRBES in Definition 6.6.2, similar to definition 5.3.1 of possible causes in an LREBES, and give an alternative equivalent definition in Lemma 6.6.3. This definition requires that X is finite and $e \notin X$ for any RPES event (X, e) means we give each event a finite set of causes, and the event does not cause itself. Some processes can generate cycles of bundles, such as $a(x).\overline{b}(c)|d(y).\overline{e}(f)$, where $a(f)$ is a possible cause of $\overline{b}(c)$, which is a possible cause of $d(c)$, which is a possible cause of $\overline{e}(f)$, which is a possible cause of $a(f)$. These kinds of cycles are allowed in LRBESs, but when transforming then into LRPESs, we only take the instance of $a(f)$, which was not transitively caused by $d(c)$, but instead by d receiving some other name.

Definition 6.6.2 (Possible Cause). *Given an LRBES,* $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ *and an event* $e \in E$*,* $X \subseteq E$ *is a possible cause of* e *if*

- $e \notin X$*,* X *is finite, whenever* $X' \mapsto e$ *we have* $X' \cap X \neq \emptyset$*;*
- *for any* $e', e'' \in \{e\} \cup X$*, we have* $e' \sharp e''$ *(*$X \cup \{e\}$ *is conflict-free);*
- *for all* $e' \in X$*, there exists* $X'' \subseteq X$*, such that* X'' *is a possible cause of* e'*;*
- *there does not exist any* $X''' \subset X$*, such that* X''' *is a possible cause of* e*.*

Lemma 6.6.3. *Given a CLRBES,* $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$*,* $X \subseteq E$ *is a possible cause of* $e \in E$ *if and only if in* $C_{br}(\mathcal{E})$*,*

1. X *is a reachable and finite configuration;*
2. $X \xrightarrow{e}$*;*
3. *for all reachable configurations* $X'' \subset X$*,* $X'' \not\xrightarrow{e}$*.*

Proof. We prove this by induction in the size of X in Appendix D.13. □

Having defined possible causes, we can use them to define a mapping from the CLRBESs generated by our denotational semantics to LRPESs which can more easily be compared with the LRPESs generated by our operational semantics.

Definition 6.6.4 (CLRBES to LRPES). *Given a CLRBES* $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$*, we can create an LRPES* $p_B(\mathcal{E}) = (E', F', <, \sharp', \prec, \rhd', \lambda', \mathsf{Act}) \upharpoonright \{e \mid e' < e \text{ and } e'' < e \Rightarrow e' \sharp e''\}$ *where:*

- $E' = \{(X, e) \mid X$ *is a possible cause of* $e\}$*;*
- $F' = \{(X, e) \mid (X, e) \in E'$ *and* $e \in F\}$*;*

- $(X,e) < (X',e')$ *if* $X \cup \{e\} \subseteq X'$;
- $(X,e)\ \sharp''\ (X',e')$ *if* $e \sharp e'$ *or* $e = e'$ *and* $X \neq X'$;
- $\sharp'$ *is* $\sharp''$ *closed under conflict heredity and symmetry;*
- $\prec = \emptyset \times \emptyset$;
- $(X,e) \rhd' \underline{(X,e')}$ *if* $(X',e') < (X,e)$;
- $\lambda'((X,e)) = \lambda(e)$.

Definition 6.6.5. *We define a functor* P_B *such that:*

- $P_B(\mathcal{E}) = p_B(\mathcal{E})$;
- $P_B(f) = f'$ *where* $f'((X,e)) = (f(X), f(e))$.

Proposition 6.6.6. P_B *is a functor.*

Proof. This is similar to previous functors. See Appendix D.14 for details. □

Lemma 6.6.7 (P_B preserves transitions)**.** *Given a CLRBES,* $\mathcal{E}$*, and a reachable configuration,* Y*, of* $\mathbf{C}_{br}(\mathcal{E})$*,* $Y \xrightarrow{e}$ *if and only if there exists a reachable configuration of* $\mathbf{C}_{pr}(P_B(\mathcal{E}))$*,* $Y' = \{(X',e') \mid X' \cup \{e'\} \subseteq Y\}$ *and* $X \subseteq Y$ *such that* $Y' \xrightarrow{(X,e)}$.

Proof. We prove this by induction in the size of Y and Y' in Appendix D.15. □

Now that both our event structure semantics can generate LRBESs, we can show that the LRPES generated by the denotational semantics is isomorphic with the sub-LRPES of the LRPES generated by the operational semantics, which does not include any actions generated by α-converting apart from when unfolding replication.

In Example 6.6.8 we show the event structures generated by the two different semantics for the process $a(x)|\overline{a}(b).\overline{b}(c)$. In the operational semantics we only describe the events which include a trace not requiring α-conversion. We see that the converting the RBES generated by the denotational semantics to an RPES makes the possible cause act similarly to the trace.

Example 6.6.8. *Consider the process,* $a(x)|\overline{a}(b).\overline{b}(c)$*. We would like to show the correspondence between the LRPESs generated by this process using the two methods presented. Using the denotational event structure semantics of Section 6.4, we get* $\left\{\!\!\left\{ a(x)|\overline{a}(b).\overline{b}(c) \right\}\!\!\right\}_{a,b,c,x} = \langle (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act}), \emptyset, \emptyset \rangle$ *shown in Figure 6.4, where, since there is only one event per label, we assume the events have the same name and label for simplicity:*

$E = F = \{a(x), a(a), a(b), a(c), \tau, \overline{a}(b), \overline{b}(c)\}$
$\{\overline{a}(b)\} \mapsto a(b),\ \{\overline{a}(b), \tau\} \mapsto \overline{b}(c),\ \{\overline{b}(c)\} \mapsto a(c)$
$a(x)\ \sharp\ a(a),\ a(x)\ \sharp\ a(b),\ a(x)\ \sharp\ a(c),\ a(a)\ \sharp\ a(b),\ a(a)\ \sharp\ a(c),\ a(b)\ \sharp\ a(c),\ a(x)\ \sharp\ \tau,\ .$
$a(a)\ \sharp\ \tau,\ a(b)\ \sharp\ \tau,\ a(c)\ \sharp\ \tau,\ \overline{a}(b)\ \sharp\ \tau$
$a(b) \rhd \underline{\overline{a}(b)},\ \overline{b}(c) \rhd \underline{\tau},\ \overline{b}(c) \rhd \underline{\overline{a}(b)},\ a(c) \rhd \underline{\overline{b}(c)}$

We then turn this LRBES into an LRPES, $P_B((E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})) = (E_P, F_P, <_P, \sharp_P, \prec_P, \rhd_P, \lambda_P, \mathsf{Act}_P)$*:*

$E_P = F_P = \{(\emptyset, a(x)), (\emptyset, a(a)), (\{\overline{a}(b)\}, a(b)), (\{\overline{b}(c), \overline{a}(b)\}, a(c)),$
$(\emptyset, \overline{a}(b)), (\emptyset, \tau), (\{\overline{a}(b)\}, \overline{b}(c)), (\{\tau\}, \overline{b}(c))\}$

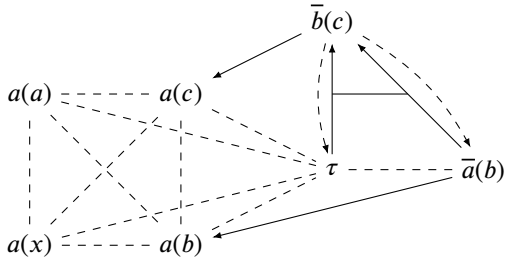


Figure 6.4: The LRBES seen in Example 6.6.8

$$
\begin{array}{rlrl}
\{(\emptyset,\overline{a}(b))\} & <_P (\{\overline{a}(b)\},a(b)), & (\emptyset,\overline{a}(b)) & <_P (\{\overline{a}(b)\},\overline{b}(c)),\\
(\emptyset,\tau) & <_P (\{\tau\},\overline{b}(c)), & (\{\overline{a}(b)\},\overline{b}(c)) & <_P (\{\overline{b}(c),\overline{a}(b)\},a(c)),\\
(\emptyset,\overline{a}(b)) & <_P (\{\overline{b}(c),\overline{a}(b)\},a(c)) & &\\
(\emptyset,a(x)) & \sharp_P (\emptyset,a(a)), & (\emptyset,a(x)) & \sharp_P (\{\overline{a}(b)\},a(b)),\\
(\emptyset,a(x)) & \sharp_P (\{\overline{b}(c),\overline{a}(b)\},a(c)), & (\emptyset,a(a)) & \sharp_P (\{\overline{a}(b)\},a(b)),\\
(\emptyset,a(a)) & \sharp_P (\{\overline{b}(c),\overline{a}(b)\},a(c)), & (\{\overline{a}(b)\},a(b)) & \sharp_P (\{\overline{b}(c),\overline{a}(b)\},a(c)),\\
(\emptyset,a(x)) & \sharp_P (\emptyset,\tau), & (\emptyset,a(a)) & \sharp_P (\emptyset,\tau),\\
(\{\overline{a}(b)\},a(b)) & \sharp_P (\emptyset,\tau), & (\{\overline{b}(c),\overline{a}(b)\},a(c)) & \sharp_P (\emptyset,\tau),\\
(\emptyset,\overline{a}(b)) & \sharp_P (\emptyset,\tau), & (\emptyset,a(x)) & \sharp_P (\{\tau\},\overline{b}(c)),\\
(\emptyset,a(a)) & \sharp_P (\{\tau\},\overline{b}(c)), & (\{\overline{a}(b)\},a(b)) & \sharp_P (\{\tau\},\overline{b}(c)),\\
(\{\overline{b}(c),\overline{a}(b)\},a(c)) & \sharp_P (\{\tau\},\overline{b}(c)), & (\emptyset,\overline{a}(b)) & \sharp_P (\{\tau\},\overline{b}(c)),\\
(\{\overline{a}(b)\},\overline{b}(c)) & \sharp_P (\emptyset,\tau), & (\{\overline{a}(b)\},\overline{b}(c)) & \sharp_P (\{\tau\},\overline{b}(c))\\
(\{\overline{a}(b)\},a(b)) & \rhd_P \underline{(\emptyset,\overline{a}(b))}, & (\{\overline{a}(b)\},\overline{b}(c)) & \rhd_P \underline{(\emptyset,\overline{a}(b))},\\
(\{\tau\},\overline{b}(c)) & \rhd_P \underline{(\emptyset,\tau)}, & (\{\overline{b}(c),\overline{a}(b)\},a(c)) & \rhd_P \underline{(\emptyset,\overline{a}(b))},\\
(\{\overline{b}(c),\overline{a}(b)\},a(c)) & \rhd_P \underline{(\{\overline{a}(b)\},\overline{b}(c))} & &\\
\lambda_P((X,e)) = \lambda(e) & & &
\end{array}
$$

Note that there are now two $\overline{b}(c)$ events, but only one $a(c)$ event, as $a(c)$ is in conflict with τ, and therefore cannot be caused by it, so we do not get $(\{\overline{b}(c),\tau\},a(c))$.

We also generate another LRPES by applying the operational semantics of Section 6.5 to $(\emptyset,\emptyset,\emptyset) \vdash a(x)|\overline{a}(b).\overline{b}(c)$ to get a PES. Similarly to Example 6.5.5 we exclude traces requiring α-conversion and any inputs of names not already present in the process in order to get a finite number of events. We apply $\mathbf{L}$ to turn it into a CLRPES, as seen in Definition 6.5.6, giving us $(E',F',<',\sharp',\prec',\rhd',\lambda',\mathsf{Act}')$ with $E'=F'$ consisting of:

$$
\begin{array}{l}
e'_{a(x)} = \left[\left(a(x),0[a(x)][0],\emptyset\right)\right]_{\approx}\\
e'_{a(a)} = \left[\left(a(a),0[a(x)][0],\emptyset\right)\right]_{\approx}\\
e'_{a(b)} = \left[\left(\overline{a}(b),1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)],\emptyset\right),\left(a(b),0[a(x)][0],\{(\overline{a}(b),1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)])\}\right)\right]_{\approx}\\
e'_{a(c)} = \left[\left(\overline{a}(b),1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)],\emptyset\right),\left(\overline{b}(c),1[\overline{b}(c)][0],\{(\overline{a}(b),1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)])\}\right),\right.\\
\left.\left(a(c),0[a(x)][0],\{\overline{b}(c),1[\overline{b}(c)][0]\}\right)\right]_{\approx}\\
e'_{\tau} = \left[\left(\tau,\langle 0[a(x)][0],1[\overline{a}(b)][0]\rangle,\emptyset\right)\right]_{\approx}\\
e'_{\overline{a}(b)} = \left[\left(\overline{a}(b),1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)],\emptyset\right)\right]_{\approx}\\
e'_{\underline{b(c)}_{\overline{a}(b)}} = \left[\left(\overline{a}(b),1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)],\emptyset\right),\left(\overline{b}(c),1[\overline{b}(c)][0],\{(\overline{a}(b),1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)])\}\right)\right]_{\approx}\\
e'_{\underline{b(c)}_{\tau}} = \left[\left(\tau,\langle 0[a(x)][0],1[\overline{a}(b)][0]\rangle,\emptyset\right),\left(\overline{b}(c),1[\overline{b}(c)][0],\emptyset\right)\right]_{\approx}
\end{array}
$$

Where

$$
\begin{array}{lll}
e'_{\overline{a}(b)} <' e'_{a(b)}, & e'_{\overline{a}(b)} <' e'_{\overline{b(c)}_{\overline{a}(b)}}, & e'_{\tau} <' e'_{\overline{b(c)}_{\tau}}, \\
e'_{\overline{b(c)}_{\overline{a}(b)}} <' e'_{a(c)}, & e'_{\tau} <' e'_{a(c)} &
\end{array}
$$

$$
\begin{array}{lll}
e'_{a(x)} \sharp' e'_{a(a)}, & e'_{a(x)} \sharp' e'_{a(b)}, & e'_{a(x)} \sharp' e'_{a(c)}, \\
e'_{a(a)} \sharp' e'_{a(b)}, & e'_{a(a)} \sharp' e'_{a(c)}, & e'_{a(b)} \sharp' e'_{a(c)}, \\
e'_{a(x)} \sharp' e'_{\tau}, & e'_{a(a)} \sharp' e'_{\tau}, & e'_{a(b)} \sharp' e'_{\tau}, \\
e'_{a(c)} \sharp' e'_{\tau}, & e'_{a(c)} \sharp' e'_{\overline{b(c)}_{\overline{a}(b)}}, & e'_{\tau} \sharp e'_{\overline{a}(b)}, \\
e'_{a(x)} \sharp' e'_{\overline{b(c)}_{\tau}}, & e'_{a(a)} \sharp' e'_{\overline{b(c)}_{\tau}}, & e'_{a(b)} \sharp' e'_{\overline{b(c)}_{\tau}}, \\
e'_{a(c)} \sharp' e'_{\overline{b(c)}_{\tau}}, & e'_{\tau} \sharp' e'_{\overline{b(c)}_{\tau}}, & e'_{\overline{b(c)}_{\overline{a}(b)}} \sharp' e'_{\overline{b(c)}_{\tau}}
\end{array}
$$

$$
\begin{array}{lll}
e'_{a(b)} \rhd' \underline{e'_{\overline{a}(b)}}, & e'_{\overline{b(c)}_{\overline{a}(b)}} \rhd' \underline{e'_{\overline{a}(b)}}, & e'_{\overline{b(c)}_{\tau}} \rhd' \underline{e'_{\overline{b(c)}_{\tau}}}, \\
e'_{a(c)} \rhd' \underline{e'_{\overline{b(c)}_{\overline{a}(b)}}}, & e'_{\tau} \rhd' \underline{e'_{\overline{b(c)}_{\overline{a}(b)}}} &
\end{array}
$$

$$
\lambda(e') = \begin{cases} \mu & \text{if } e' = e'_{\mu} \text{ and } \mu \in \{a(x), a(a), a(b), a(c), \tau, \overline{a}(b)\} \\ \overline{b}(c) & \text{if } e' \in \{e'_{\overline{b(c)}_{\overline{a}(b)}}, e'_{\overline{b(c)}_{\tau}}\} \end{cases}
$$

$\mathsf{Act}' = \{a(x), a(a), a(b), a(c), \tau, \overline{a}(b), \overline{b}(c)\}$

We can define a mapping between these two event structures,

$$
f((X, e) = \begin{cases} e'_{e} & \text{if } e \in \{a(x), a(a), a(b), a(c), \tau, \overline{a}(b)\} \\ e'_{\overline{b(c)}_{\overline{a}(b)}} & \text{if } e = \overline{b}(c) \text{ and } \overline{a}(b) \in X \\ e'_{\overline{b(c)}_{\tau}} & \text{if } e = \overline{b}(c) \text{ and } \tau \in X \end{cases}
$$

and f and f^{-1} are both morphisms, since the event structures only differ in the names of the events.

However, as noted above, the real event structure generated by the operational semantics would also include events corresponding to traces in which α-conversion is performed, such as $[(\tau, \langle 0[a(y)][0], 1[\overline{a}(d)][0]\rangle, \emptyset)]_{\approx}$, which corresponds to the same action as e'_{τ}, but the communication is only performed after x has been α-converted to y and b to d. The denotational event structure semantics do not generate any event for this possibility, as they assume all names are distinct and the only α-conversion done is one necessitated by wanting to keep all names distinct after a replication.

We now show isomorphism between the two event structure semantics.

Theorem 6.6.9. *Given a standard πIK-calculus process P with guarded choice, distinct names and $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ for some $\mathcal{N} \supseteq n(P)$ and $\mathcal{E}'$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash P$, we can define two LRPES morphisms, $f : P_B(\mathcal{E}) \to \mathcal{E}'$ and $f' : \mathcal{E}' \to P_B(\mathcal{E})$, such that $f \circ f' = 1_{P_B(\mathcal{E})}$ and if $f'(e')$ is defined, then $f(f'(e')) = e'$.*

Proof. We prove the theorem by induction on the structure and level of nesting of replication in P in Appendix D.16 by defining the morphisms for each case. The most complex case is the parallel composition, where to define f we need to use the set of causes given to each event in the denotational semantics to create a trace of actions which corresponds to an event in the operational semantics, and vice versa to define g we need to find a set of causes from an equivalence class of traces. □

We now know that the actions included in both event structure semantics are isomorphic, which strengthens the operational correspondence between πIK and πIH found in section 6.3, but we would like to go one step further and include the events not found in the denotational event structure semantics.

The operational event structure semantics generate additional events corresponding to the actions the process can take if α-converted, and the input actions receiving names not in $\mathcal{N}$. We would like to show that

the way these events interact corresponds to the way the events generated by the denotational semantics interact, although the events will have different labels. In order to get a correspondence between events generated by α-converting the process and events generated without α-converting the process, we wish to define substitution on traces.

We specifically want to define substitution of a name x for a in trace t, at location u in which a is bound, but not continuing after a set of locations, U, at which new xs are bound. To do this we look at each triple (μ, u', D) in t in order, and determine whether it is in a location where the substitution should take place, that is u' is within the gap from u to U as defined in Definition 6.6.10. If this is the case, then we substitute free occurrences of x for a in μ and the processes of u'. If x is bound in μ, then we add u' to U before continuing. If a is bound in μ then we substitute a for some new name, d. If μ is an input of x and u' is not a location where the substitution should take place, then we use D to determine whether the x being received is the same as the one being substituted or a different one. If μ is τ, then we use the processes in u' to figure out which channel communicated which name, and then treat it as separate input and output. The full definition can be seen in Definition 6.6.11.

Definition 6.6.10 (location order)**.** *We define an ordering on locations, $\prec$, as follows:*

- $l[P][Q] \prec l'[P'][Q']$ *if* $l' = ll''$ *or* $l' = l$ *and* $Q \rightarrow^* P'$*;*
- $l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle \prec l'[P'][Q']$ *if* $ll_i[P_i][Q_i] \prec l'[P'][Q']$ *for some* $i \in \{0,1\}$*;*
- $l[P][Q] \prec l'\langle l'_0[P'_0][Q'_0], l'_1[P'_1][Q'_1]\rangle$ *if* $l[P][Q] \prec l'l'_i[P'_i][Q'_i]$ *for some* $i \in \{0,1\}$*;*
- $l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle \prec l'\langle l'_0[P'_0][Q'_0], l'_1[P'_1][Q'_1]\rangle$ *if* $ll_i[P_i][Q_i] \prec l'l'_j[P'_j][Q'_j]$ *for some* $i, j \in \{0,1\}$*.*

Given a start location u and a set of end locations U, we say that u' is within the gap from u to U, $\mathsf{within}_{u-U}(u')$*, if $u \prec u'$ and there does not exist $u'' \in U$ such that $u'' \prec u'$.*

Definition 6.6.11. *Given a trace t, a location u, a set of locations, U, and names a, x, t with x substituted for a starting at location u but not continuing after locations in U, $t[x := a]_{(u-U)}$ is defined as :*

1. $((\alpha, l[P][Q], D)t)[x := a]_{(u-U)} = (\alpha', l[P'][Q'], D')t'$ *where:*

   (a) $\alpha' = \alpha''[x := a]$ *if* $\mathsf{within}_{u-U}(l[P][Q])$ *and otherwise* $\alpha' = \alpha''$
   where if $\alpha = \overline{x}(n)$, $\alpha[x := a] = \overline{a}(n)$ *and if* $\alpha = x(n)$, $\alpha[x := a] = a(n)$ *and otherwise* $\alpha[x := a] = \alpha$.
   And $\alpha'' = b(a)$ *if* $\alpha = b(x)$ *and* $(x, u') \in D$, $\alpha'' = b(d)$ *if* $\alpha = b(a)$, $\alpha'' = \overline{b}(d)$ *if* $\alpha = \overline{b}(a)$, *and otherwise* $\alpha'' = \alpha$.

   (b) $P' = P[x := a]$ *if* $\mathsf{within}_{u-U}(l[P][Q])$ *and otherwise* $P' = P$.

   (c) $Q' = Q''[x := a]$ *if* $\mathsf{within}_{u-U}(l[P][Q])$ *and otherwise* $Q' = Q''$
   where $Q'' = Q[a := d]$ *if* $a \in \mathsf{bn}(\alpha)$ *and* $\mathsf{within}_{u-U}(l[P][Q])$, *and otherwise* $Q'' = Q$.

   (d) $D' = D''[x := a]_{(u-U)}$ *where* $D'' = D[a := d]_{(l[P][Q]-\emptyset)}$ *if* $\mathsf{within}_{u-U}(l[P][Q])$ *and* $a \in \mathsf{bn}(\alpha)$, *and otherwise* $D'' = D$.

   (e) $t' = t''[x := a]_{(u-U')}$ *where* $U' = U \cup \{l[P'][Q']\}$ *if* $x \in \mathsf{bn}(\alpha)$ *and otherwise* $U' = U$
   where $t'' = t[a := d]_{(l[P][Q]-\emptyset)}$ *if* $a \in \mathsf{bn}(\alpha)$ *and* $\mathsf{within}_{u-U}(l[P][Q])$, $t'' = t[x := a]_{(u-\emptyset)}$ *if* $\alpha = b(x)$, *and otherwise* $t'' = t$.

   for $d \notin \mathsf{fn}(Q)$

2. $((\tau, l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle, D)t)[x := a]_{(u-U)} = ((\tau, l\langle l_0[P'_0][Q'_0], l_1[P'_1][Q'_1]\rangle, D')t')$ *where* $P_i \xrightarrow{b(n)} Q_i$ *and* $P_{1-i} \xrightarrow{\overline{b}(n)} Q_{1-i}$ *and:*

   (a) $P'_j = P_j[x := a]$ *if* $\mathsf{within}_{u-U}(ll_j[P_j][Q_j])$.

(b) $Q'_j = Q''_j[x := a]$ *if* $\mathsf{within}_{u-U}(ll_j[P_j][Q_j])$ *and* $n \neq x$, *and otherwise* $Q'_j = Q''_j$ *where* $Q''_j = Q_j[a := d]$ *if* $n = a$ *and either* $\mathsf{within}_{u-U}(l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle)$, *and otherwise* $Q''_j = Q_j$.

(c) $D' = D''[x := a]_{(u-U)}$ *where* $D'' = D[a := d]_{(l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle - \emptyset)}$ *if* $n = a$ *and* $\mathsf{within}_{u-U}(l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle)$, *and otherwise* $D'' = D$.

(d) $t' = t''[x := a]_{(u-U')}$ *where* $U' = U \cup \{l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle\}$ *if* $n = x$ *and otherwise* $U' = U$ *where* $t'' = t[a := d]_{(l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle - \emptyset)}$ *if* $n = a$ *and* $\mathsf{within}_{u-U}(l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle)$ *and otherwise* $t'' = t$.

for $d \notin \mathsf{fn}(Q_0) \cup \mathsf{fn}(Q_1)$.

Where we use the following definition of substitution on a set of causes:
$D[x := a]_{(u-U)} = \left\{(\alpha'', u'') \mid (\alpha', l[P][Q]) \in D \text{ and if } l[P][Q] \prec u \text{ or there exists } u' \in U \text{ such that } u' \prec l[P][Q], \text{ then } \alpha'' = \alpha' \text{ and } u'' = l[P][Q], \text{ otherwise } \alpha'' = \alpha'[x := a] \text{ and } u'' = l[P[x := a]][Q[x := a]]\right\}$

We then use this definition to prove that we can "α-convert" traces by substituting a bound name in an input or output action in the trace and then substituting that name on that location in the rest of the trace.

Lemma 6.6.12. *Consider a πIH-calculus process with empty extrusion histories, $(\emptyset, \emptyset, \emptyset) \vdash P$, and a trace, $t = (\mu_0, u_0, D_0), \ldots, (\mu_n, u_n, D_n)$ such that*

$$(\emptyset, \emptyset, \emptyset) \vdash P \xrightarrow[u_0, D_0]{\mu_0} \mathbf{H}_1 \vdash P_1 \ldots \mathbf{H}_n \vdash P_n \xrightarrow[u_n, D_n]{\mu_n} \mathbf{H}' \vdash P'$$

Then for any $(\mu_i, l_i[Q_i][Q'_i], D_i)$ such that either $\mu_i = \overline{b}(x)$ or $\mu_i = b(x)$, and we can define new traces $t_0 = (\mu_0, u_0, D_0) \ldots, (\mu_{i-1}, u_{i-1}, D_{i-1})$ and $t_1 = (\mu_{i+1}, u_{i+1}, D_{i+1}), \ldots, (\mu_n, u_n, D_n)[x := a]_{(l_i[Q_i][Q'_i] - \emptyset)}$ and combine them to get

$$t' = t_0(\mu'_i, l_i[Q_i][Q_i[x := a]], D'_i)t_1 = (\mu'_0, u'_0, D'_0), \ldots, (\mu'_n, u'_n, D'_n)$$

where $\mu'_i = b(a)$ if $\mu_i = b(x)$ and $\mu'_i = \overline{b}(a)$ if $\mu_i = \overline{b}(x)$, and $D'_i = (D_i \rightharpoonup b) \cup (a, u)$ if $(\overline{c}(a), u \in \overline{H}_i$ and otherwise $D'_i = (D_i \rightharpoonup b)$.

We can then use this trace to get

$$(\emptyset, \emptyset, \emptyset) \vdash P \xrightarrow[u'_0, D'_0]{\mu'_0} \ldots \xrightarrow[u'_n, D'_n]{\mu'_n}$$

Proof. We have a transition

$$\mathbf{H}_i \vdash P_i \xrightarrow[l_i[Q_i][Q'_i], D'_i]{\mu'_i} \mathbf{H}_{i+1} - (\mu_i, l_i[Q_i][Q'_i]) + (\mu'_i, l_i[Q_i][Q'_i[x := a]]) \vdash P'_{i+1}$$

created by simply receiving a different name.

We can then show that $\mathbf{H}_{i+1} - (\mu_i, l_i[Q_i][Q'_i]) + (\mu'_i, l_i[Q_i][Q'_i[x := a]]) \vdash P'_{i+1} \xrightarrow[u'_{i+1}, D'_{i+1}]{\mu_i + 1} \ldots \xrightarrow[u'_n, D'_n]{\mu'_n}$ by simple induction in the length of the trace. □

Finally we show our label-ignoring morphism from all the events generated by the operational semantics to the events generated by the denotational semantics. For this we use a label removing function $\mathsf{lr}((E, F, <, \sharp, \prec, \rhd, \lambda, \mathsf{Act})) = (E, F, <, \sharp, \prec, \rhd)$.

Theorem 6.6.13. *Given a standard πI-calculus process P with guarded choice, distinct names and $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ for some $\mathcal{N} \supseteq n(P)$ and $\mathcal{E}'$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \emptyset, \emptyset) \vdash P$, and f and f' defined as above, we have an RPES morphism $f'' : \mathsf{lr}(\mathcal{E}') \to \mathsf{lr}(P_B(\mathcal{E}))$ such that $\mathsf{dom}(f'') = E'$ and $f'' \upharpoonright \mathsf{dom}(f') = f'$.*

Proof. We prove this by induction on the structure and level of nesting of replication in P in Appendix D.17 by defining the morphisms for each case. The morphisms defined in this proof are extensions of the ones from the proof on Theorem 6.6.9. □

With these theorems, we can see that we have a tight correspondence between the operational and denotational event structure semantics of reversible πI-calculi. This builds on the correspondence between πIH and πIK we established in Section 6.3. Since it turns out that it is easier to define denotational true concurrency semantics of statically reversible calculi and operational true concurrency semantics, it is useful to know that the two methods create event structures with a tight correspondence. The fact that disregarding the labels of events created by computations which use α-conversion means they can synchronise with their non-substituted equivalents also helps assure us that both our semantics and our definition of morphisms are sensible.

6.7 Conclusion

We have now defined πIK and πIH, two early reversible semantics of π-calculi. All existing reversible versions of the π-calculus use reduction semantics [59,89] or late semantics [33,67], despite the early semantics being used more widely than the late in the forward-only setting. πIK and πIH are reversible forms of the *internal* π-calculus, where names being sent in output actions are always bound. As well as structural causation, as in CCS, the early form of the internal π-calculus also has a form of link causation created by the semantics being early, which is not present in other reversible π-calculi.

In πIH, adapted from [52], the past actions of all processes are stored together similarly to $\rho\pi$ [59] and ReSπ [88]. This required extending the extrusion histories used by [52] to also store communications.

In contrast, πIK, based on the framework for reversible π-calculi [67] and CCSK [80], uses static reversibility by annotating past actions with keys. Using keys to define πI-calculus semantics required more annotations that CCSK, since we now need to annotate substitutions, but less than a full π-calculus, since we do not need to worry about opening restrictions. We showed that a πIH process with extrusion histories can be mapped to a πIK process with keys, creating an operational correspondence (Theorem 6.3.8). This is similar to the correspondence proved between statically reversible CCS (CCSK) and dynamically reversible CCS (RCCS) by [57], however we are not aware of such a result for full π-calculi.

As we did for CCSK in Chapter 5, we defined event structure semantics of our new calculi. The event structure semantics of πIK are defined similarly to CCSK, but with added substitution and link causation. We show an operational correspondence between πIK processes and their event structure representations in Theorems 6.4.11 and 6.4.12.

The event structure semantics of πIH on the other hand are defined operationally, being derived from a labelled asynchronous transition system generated similarly to the full forward-only π-calculus in [52]. This creates a LRPES in which each event is a set of traces of the process, all of which end with the same action and contain the same causes of this action. Since πIH uses a similar approach to $\rho\pi$ [59] and ReSπ [88], it is possible that with minor changes, a similar method to the one used in Section 6.5 could be used to create true concurrency semantics of these calculi.

We define morphisms between the event structures created by equivalent πIK and πIH calculus processes, which show that not only do we have a correspondence between the statically and dynamically reversible calculi, but between the event structures generated by their operational and denotational event structure semantics *Theorem 6.6.13). This does not mean there are no benefits to using one calculus over the other. Having a notion of independence of transitions meant we could easily prove useful properties of πIH, which we do not have for πIK. On the other hand, not keeping past actions in a shared memory makes πIK more compositional than πIH. This is also reflected in the event structure semantics, which are more compositional for πIK, but were simpler to define and prove correctness of based on existing mappings for πIH.

As we now have event structures of three different calculi, it may now be worth considering more ways of analysing and comparing the event structures than just morphisms. Event Identifier Logic [81] and Logic for True Concurrency [9] are both useful for reasoning about causally reversible event structures, which includes the ones generated by our event structure semantics of uncontrolled calculi, but we have not yet investigated their application on our processes.

The key result of this chapter is that we have two ways of proving a correspondence between a statically and dynamically reversible π-calculus. One direct encoding and one proving isomorphism between the event structures they generate.

Chapter 7

Conclusion

The first interest in reversible programming came from the assertion that reversible actions do not generate heat [55], and therefore might consume less energy than irreversible. In sequential reversible programs, actions can be undone starting with the most recent, but in a concurrent setting we cannot always know what the most recent action is. This leads to a notion of causal reversibility where actions can reverse if and only if they have not caused other actions. Process calculi are strong high-level models of interactions between parallel processes, and reversible variants of a number of process calculi have been defined for modelling parallel reversible processes. In reversible processes, causation is divided into structural causation, caused by the structure of a process such as one action prefixing another, and link causation, caused by one action making a bound name available to another. Reversible process calculi can also distinguish true concurrency in a way forward-only processes cannot [79]. We have therefore aimed in this thesis to define true concurrency semantics with explicit causation for some of these reversible process calculi, specifically using reversible event structures to define true concurrency semantics of reversible CCS and internal π-calculus.

For this purpose, we first studied reversible event structures in a categorical setting. We have defined categories for configuration systems (CS), reversible prime event structures (RPES), reversible asymmetric event structures (RAES), reversible bundle event structures (RBES), reversible extended bundle event structures (REBES), reversible flow event structures (RFES) and reversible general event structures (RES). We have defined functors between these categories, showing that these event structures can be modelled as CSs and finitely enabled CSs can be modelled as RESs in a way that preserves morphisms. Unfortunately our RFES category is somewhat lacking compared to the others. The RFES-morphisms do not preserve reachable configurations, and consequently the mappings to and from RFESs do not preserve morphisms and are not functors. We would therefore consider that a different definition would be better, although this would probably prevent us from defining products between all RFESs, as the FES product only works for well-behaved FESs. We also constructed coproducts for each of these categories, though products only for RESs, RBESs, REBESs, RFESs, CSs and causal RPESs (Remark 4.2.22). Constructing products of RPESs and RAESs will likely be much more difficult than for RESs, since definitions of products of prime event structures are far more complex than those of general event structures [91], and products of asymmetric event structures have not been defined, to the best of our knowledge.

We showed that, unlike in the forward-only setting, reversible asymmetric event structures can be modelled as reversible general event structures, which in finitely-based cases will generate the same CS (Theorem 4.7.23).

We showed that our functors between RESs and finitely enabled CSs form an adjunction (Theorem 4.7.17), and defined a subset of finitely enabled CSs, for which our functors between RESs and finitely enabled CSs form an inverse in some cases (Theorem 4.7.16).

We defined stable subcategories of RESs and CSs, where we know which events have caused each event in a given configuration. We defined a functor from stable finitely enabled CSs to stable RESs. We defined a normal form of RESs and showed that the categories of stable RESs in normal form and finitely enabled stable CSs are isomorphic (Corollary 4.8.17). We also defined a normaliser function, which maps RESs to RESs in normal form which can be represented by the same CS. This significantly improves our ability to compare and combine stable systems modelled by CSs and RESs. Some of our work on the reversible event

structure categories has turned out to be unnecessary. Since we found a close correspondence between RESs and CSs, defining mappings from the other event structure categories to both of these has turned out to be unnecessary. And as we discovered a number of adjunctions between our categories, it turns out that some of our work on defining products and coproducts could have been avoided.

We have also defined cause-respecting subcategories, in which events cannot be reversed if they have caused other events, and causal subcategories, in which events can be reversed if and only if they have not caused other events.

We used a causal subcategory of RBESs to describe event structure semantics of uncontrolled CCSK, the first event structure semantics of a reversible calculus that we are aware of. We chose to use CCSK for this purpose because the way CCSK records previous actions preserves the structure of the process, meaning we can use very similar semantics for previous and future actions.

Unlike existing efforts to describe true concurrency semantics of a reversible process calculus using rigid families [34], we have used the way CCSK handles previous actions to generate both the event structure and the initial state directly from the process, rather than needing to first undo previous actions to get the original process and from there the rigid family, and then redo the actions to get the initial state. Our event structure semantics have shown that the same process at different points in its execution will generate isomorphic event structures with different initial states.

We have also proposed a variant of CCSK called Roll-CCSK, which uses the rollback described in [58] to achieve controlled static reversibility. We have given two semantics for this calculus, one in which the entire roll is completed in one step, and one in which each action being rolled back takes a step. We have proved an operational correspondence between these semantics. We used *non-causal* REBESs to define the event structure semantics of Roll-CCSK. Using event structures with non-causal reversibility allow us to treat rolls as normal events where $a_\gamma.\mathsf{roll}\ \gamma$ can have the event a_γ cause the event $\mathsf{roll}\ \gamma$, and at the same time have a_γ prevent $\mathsf{roll}\ \gamma$ from reversing until the roll is complete. It is our intention that this approach to describing event structure semantics of rollback can be used to add rollback to other true concurrency semantics without necessarily defining operational semantics of rollback in other calculi. Considering this it might have been better to define the event structure semantics of Roll-CCSK as a direct addition to the event structure semantics of CCSK instead of defining the two separately, but this is left as future work.

We have proved operational correspondence between the operational and event structure semantics of both CCSK (Theorems 5.1.18 and 5.1.19) and Roll-CCSK (Theorems 5.3.16, 5.3.17, 5.3.25, and 5.3.26).

All existing reversible versions of the π-calculus use reduction semantics [59,89] or late semantics [33,67], despite the early semantics being used more widely than the late in the forward-only setting. We have introduced πIH and πIK, the first reversible early π-calculi. They are reversible variants of the *internal* π-calculus, where names being sent in output actions are always bound. As well as structural causation, as in CCS, the early form of the internal π-calculus also has a form of link causation (causation not created by the structure of the process) created by the semantics being early, which is not present in other reversible π-calculi. In πIH past actions are tracked by using extrusion histories adapted from [52], which move past actions and their locations into separate histories for dynamic reversibility. πIK keeps the structure of the process intact but annotates past actions with keys, similarly to πK [67] and CCSK [80]. We showed that a process πIH with extrusion histories can be mapped to a πIK process with keys, creating an operational correspondence (Theorem 6.3.8). We discovered, thanks to the work of [63], that due to adopting the notion of independence used by [52], it was surprisingly simple to prove the standard causal reversibility results for our dynamically reversible πIH calculus. If we had known that earlier, we could perhaps have tried reversing the full π-calculus presented by [52] instead of restricting ourselves to the internal π-calculus.

We define event structure semantics of πIK inductively on the syntax of the process using RBESs, as we did for CCSK and Roll-CCSK. While causation in the internal π-calculus is simpler that in the full π-calculus, our early semantics means that we still have to handle link causation, in the form of an input receiving a free name being caused by a previous output of that free name. We show an operational correspondence between πIK processes and their event structure representations in Theorems 5.1.18 and 6.4.11.

The event structure semantics of πIH on the other hand are defined operationally, being derived from a labelled asynchronous transition system. This creates an LRPES in which each event is a set of traces of the process, all of which end with the same action and contain the same causes of this action.

The denotational semantics assume that the process has already been α-converted so all free and bound names are different, and only use α-conversion when expanding a replication. Therefore the denotational

event structure semantics do not include actions requiring α-conversion, and we do not get an exact correspondence between the two event structure semantics, as the events requiring α-conversion in the RPES created by the operational semantics will not have corresponding events with the same labels in the LRPES created by the denotational semantics. Instead we get an isomorphism between the LRPES created by the denotational semantics and the subset of events of the RPES created by the operational semantics which do not require α-conversion (Theorem 6.6.9). We can also expand the part of the morphism going from the RPES created by the operational semantics to the RPES created by the denotational semantics to a morphism mapping all the events of the LRPES created by the operational semantics to events in the LRPES created by the denotational semantics, but ignoring the labels (Theorem 6.6.13).

We wanted to use event structures to define true concurrency semantics of reversible process calculi. Defining categories of event structures helped us define compositional denotational semantics by providing a way to model parallel composition and choice using products and coproducts. We were able to describe small-step true concurrency semantics of the rollback operator by first defining big-step operational semantics of rollback in CCSK, and then using non-causal REBESs to treat a rollback as a normal event.

We also wanted to define the first reversible early π-calculus. We defined two early reversible π-calculi, the statically reversible πIK-calculus and the dynamically reversible πIH-calculus, and showed an operational correspondence between them. We used event structures to define true concurrency semantics denotationally on πIK and operationally on πIH. Using the categories of event structures, we were able to determine that the event structures generated by the two approaches were isomorphic.

7.1 Future Work

The purpose of defining event structure semantics of reversible calculi is partly to create a shared framework where we can compare and reason about them. In this thesis we have used isomorphisms to compare event structures in Theorems 5.1.18, 6.4.11, 6.6.9, and 6.6.13. Our morphisms have been useful in determining that the event structures generated by different processes or different methods applied to the same process are equivalent. However, in some contexts, other ways of reasoning about event structures would be more useful. For this, we want a notion of logic and bisimulation which we can apply to reversible processes. There exist *Event Identifier Logic* [81] or *Logic for True Concurrency* [9,10], which characterise hereditary history-preserving bisimulation [16] and are therefore useful for reasoning about causal reversibility. However, as these logics are only suitable for causal reversibility, they will fall short when it comes to Roll-CCSK and other forms of controlled reversibility.

We restricted our early reversible π-calculi to πI-calculi in order to focus on the link causation derived from being early, but that does not mean we do not want event structure semantics of a full reversible π-calculus. Three different statically reversible π-calculi are provided by [67], which we could define denotational semantics of. Additionally, πIH is based on the stable non-interleaving π-calculus presented in [52], which could be used to get operational event structure semantics of a reversible full π-calculus.

As mentioned above, we believe the event structure semantics of Roll-CCSK could form the basis of a more general method of adding rollback to event structure semantics of other calculi, or even event structures not generated from calculi. Since distributed operational semantics of rollback can be difficult to create for calculi with different ways of structuring their memories, this would be worth investigating.

Now that we have some equivalence results for our statically and dynamically reversible internal π-calculi, an obvious next step would be to look at expanding these results to the full π-calculus. In addition, the full π-calculus has two forms of dynamic reversibility, one using distributed and one using centralised memories. We have not discussed equivalence between the calculi with distributed memories and the others.

Chapter 8

Notation

$\rightarrow$	a forwards transition
$\rightsquigarrow$	a reverse transition
$\rightarrowtail$	a transition which may be forwards or reverse
$\restriction$	restriction on event structure (Definition 5.1.1)
$\|\|$	parallel composition
$+$	choice
$\leq_P$	Causal dependence on communication (Definition 5.2.1)
$\star$	projection (Definition 5.2.2)
$[n]$	a communication key denoting a past action
A	functor from **PES** to **AES** (Definition 3.3.7)
A_r	functor from **RPES** to **RAES** (Definition 4.3.15)
AES	Asymmetric event structure (Definition 3.3.1)
AES	Category of asymmetric event structures
B	functor from **PES** to **BES** (Definition 3.4.15)
B_a	functor from **AES** to **EBES** (Definition 3.5.10)
B_{ar}	functor from **RAES** to **REBES** (Definition 4.5.17)
B_e	functor from **BES** to **EBES** (Definition 3.5.11)
B_{er}	functor from **RBES** to **REBES** (Definition 4.5.13)
B_r	functor from **RPES** to **RBES** (Definition 3.4.15)
BES	Bundle event structure (Definition 3.4.1)
BES	Category of bundle event structures
C	functor from **ES** to **CS** (Definition 4.1.17)
C_{er}	functor from **REBES** to **CS** (Definition 4.5.4)
C_r	functor from **RES** to **CS** (Definition 4.7.5)
C_{ra}	functor from **RAES** to **CS** (Definition 4.3.2)
C_{rb}	functor from **RBES** to **CS** (Definition 4.4.4)
C_{rf}	functor from **FES** to **CS** (Definition 4.1.15)
C_{rp}	functor from **RPES** to **CS** (Definition 4.2.4)
CaCS	Causal CS (Definition 4.9.5)
CaFCS	Causal finitely enabled CS (Definitions 4.9.5 and 4.7.11)
CaSCS	Causal stable CS (Definitions 4.9.5 and 4.8.7)
CaSFCS	Causal stable finitely enabled CS (Definitions 4.9.5, 4.8.7, and 4.7.11)
Causal	a notion of reversibility where an action can be reversed if and only if all the actions it caused have been reversed
Cause-respecting	a notion of reversibility where an action cannot be reversed if it has caused another action
CCS	Calculus of communicating systems [73]
CCSK	CCS with communicating keys (Chapter 2, [80])

Consistent process	Definition 5.2.4
CRAES	Causal RAES (Definition 4.3.17)
CRBES	Causal RBES (Definition 4.4.13)
crCS	Cause-respecting CS (Definition 4.9.4)
crFCS	Cause-respecting finitely enabled CS (Definitions 4.9.4 and 4.7.11)
CREBES	Causal REBES (Definition 4.5.10)
CRES	Causal RES (Definition 4.9.2)
CRFES	Causal RFES (Definition 4.6.18)
CRPES	Causal RPES (Definition 4.2.18)
crRAES	Cause-respecting RAES (Definition 4.3.17)
crRBES	Cause-respecting RBES (Definition 4.4.13)
crREBES	Cause-respecting REBES (Definition 4.5.10)
crRES	Cause-respecting RES (Definition 4.9.1)
crRFES	Cause-respecting RFES (Definition 4.6.16)
crRPES	Cause-respecting RPES (Definition 4.2.18)
crSCS	Cause-respecting stable CS (Definitions 4.9.4 and 4.8.7)
crSFCS	Cause-respecting stable finitely enabled CS (Definitions 4.9.4, 4.8.7, and 4.7.11)
crRSES	Cause-respecting stable RES (Definitions 4.9.1 and 4.8.1)
CS	Configuration system (Definition 4.1.1
CS	Category of configuration systems
CRSES	Causal stable RES (Definitions 4.9.2 and 4.8.1)
D	a set of causes (Definition 6.2.7)
D_a	functor from **AES** to **Dom** (Definition 3.3.5)
D_d	functor from **BES** to **Dom** (Definition 3.4.16)
D_f	functor from **FES** to **Dom** (Definition 3.6.7)
D_p	functor from **PES** to **Dom** (Definition 3.2.5)
D_s	functor from **SES** to **Dom** (Definition 3.7.9)
Dom	Category of domains
Domain	Definition 3.1.1
E	mapping from πIH to πIK (Definition 6.3.6)
E_b	functor from **RBES** to **RES** (Definition 4.7.24)
E_{er}	functor from **REBES** to **RSES** (Definition 4.7.26)
E_f	functor from **FES** to **SES** (Definition 3.7.7)
E_{rf}	functor from **RFES** to **RSES** (Definition 4.7.28)
EBES	Extended bundle event structure (Definition 3.5.1)
EBES	Category of extended bundle event structures
ES	General event structure (Definition 3.7.1)
ES	Category of general event structures
F	mapping from BES to FES (Definition 3.6.10)
F_r	mapping from RBES to RFES (Definition 4.6.15)
F_p	functor from **PES** to **FES** (Definition 3.6.8)
F_{pr}	functor from **RPES** to **RFES** (Definition 4.6.13)
FCS	Finitely enabled CS (Definition 4.7.11)
FCS	Category of finitely enabled CSs
FES	Flow event structure (Definition 3.6.1)
FES	Category of flow event structures
Forwards-reachable	Definition 4.1.19
H	an extrusion history $(\overline{H}, \underline{H}, H)$
I	independence between locations (Definition 6.2.9)
$\mathsf{key}(P)$	The set of communication keys found in P
LATS	Labelled asynchronous transition system (Definition 6.5.1, [96])

lcopy	function copying the locations in extrusion histories (Definition 6.3.5)
LP	generates a PES from a LATS (Definition 6.5.4)
loc	the set of paths of a location (Definition 6.2.8)
location	Definition 6.2.1
location order	Definition 6.6.10
LR	mapping from $LPES$ to $CLRPES$ (Definition 6.5.6)
LRBES	Labelled RBES (Definition 4.4.20)
LREBES	Labelled REBES (Definition 4.5.18)
LRPES	Labelled RPES (Definition 4.2.23)
M	a marking from low to high-level Roll-CCSK (Definition 5.2.16)
mark$^\bullet$	function marking actions for rollback (Definition 5.2.13)
N	RES normaliser (definition 4.8.19)
nfRSES	RSES in normal form (Definition 4.8.14)
nfRSES	Category of RSESs in normal form
nfRES	RES in normal form (Definition 4.8.14)
nfRES	Category of RESs in normal form
ϕ	a mapping from Roll-CCSK to CCSK (Definition 5.2.10)
P_a	functor from **AES** to **RAES** (Definition 4.3.12)
P'_a	functor from **AES** to **CRAES** (Definition 4.3.12)
ϕ_a	functor from **RAES** to **AES** (Definition 4.3.10)
P_B	functor from **CLRBES** to **CLRPES** (Definition 6.6.5)
P_b	functor from **BES** to **RBES** (Definition 4.4.16)
ϕ_b	functor from **RBES** to **BES** (Definition 4.4.17)
P_d	functor from **Dom** to **PES** (Definition 3.7.5)
P_e	functor from **EBES** to **REBES** (Definition 4.5.14)
ϕ_e	functor from **REBES** to **EBES** (Definition 4.5.15)
P_f	functor from **FES** to **RFES** (Definition 4.6.11)
ϕ_I	mapping from πIH to π-calculus with extrusion histories (Definition 6.2.23)
P_p	functor from **PES** to **RPES** (Definition 4.2.12)
P'_p	functor from **PES** to **CRPES** (Definition 4.2.19)
ϕ_p	functor from **RPES** to **PES** (Definition 4.2.13)
P_{ps}	functor from **PES** to **SES** (Definition 3.7.5)
P_r	functor from **ES** to **RES** (Definition 4.7.19)
PES	Prime event structure (Definition 3.2.1)
PES	Category of prime event structures
π-calculus	[74, 75]
πI-calculus	Internal π-calculus [85]
πIH-calculus	πI-calculus with extrusion histories (Section 6.2)
πIK-calculus	πI-calculus with communicating keys (Section 6.1)
Possible cause	Definition 5.3.1 for LREBES and Definition 6.6.2 for CLRBES
ρ	function removing labels and their dependants (Definition 5.3.3)
R	functor from **FCS** to **RES** (Definition 4.7.14)
R'	functor from **FCS** to **RES** and **SCS** to **RSES** (Definition 4.8.11)
RAES	Reversible asymmetric event structure (Definition 4.3.1)
RAES	Category of reversible asymmetric event structures
RBES	Reversible bundle event structure (Definition 4.4.1)
RBES	Category of reversible bundle event structures
rCaCS	reachably causal configuration system (Definition 4.9.9)
RCCS	Reversible CCS (Chapter 2, [35])
Reach	a function removing the unreachable part of a CS (Definition 4.1.20)
Reachable	Definition 2.1.1
REBES	Reversible extended bundle event structure (Definition 4.5.2)
REBES	Category of reversible extended bundle event structures

RES	Reversible general event structure (Definition 4.7.1)
RES	Category of reversible general event structures
Reverse-ignoring bisimulation	Definition 5.2.15
RFES	Reversible flow event structure (Definition 4.6.1)
RFES	Category of reversible flow event structures
RPES	Reversible prime event structure (Definition 4.2.1)
RPES	Category of reversible prime event structures
rt	function finding the root of a process (Definition 6.3.3)
Σ	functor from **AES** to **PES** (Definition 3.3.8)
S	function adding keys to substituted names (Definition 6.3.2)
sbn	standard bound names (Definition 6.4.1)
SCS	Stable CS (Definition 4.8.7)
SES	Stable ES (Definition 3.7.4)
SES	Category of stable ESs
RSES	Stable RES (Definition 4.8.1)
RSES	Category of stable RESs
standard process	a process containing no past actions
std(P)	P is a standard process
τ	a silent action
tag	annotation on action used to denote the target of a roll
TS	generates a LATS from a process (Definition 6.5.2)
u	a location (Definition 6.2.1)
within	determines whether a location falls between other locations (Definition 6.6.10)

Bibliography

[1] Samson Abramsky. A structural approach to reversible computation. *Theoretical Computer Science*, 347(3):441 – 464, 2005.

[2] T. Altenkirch and J. Grattage. A functional quantum programming language. In *LICS*, page 249–258, 2005.

[3] Bogdan Aman, Gabriel Ciobanu, Robert Glück, Robin Kaarsgaard, Jarkko Kari, Martin Kutrib, Ivan Lanese, Claudio Antares Mezzina, Łukasz Mikulski, Rajagopal Nagarajan, Iain Phillips, G. Michele Pinna, Luca Prigioniero, Irek Ulidowski, and Germán Vidal. *Foundations of Reversible Computation*, pages 1–40. Springer, 2020.

[4] Clément Aubert and Ioana Cristescu. Contextual equivalences in configuration structures and reversibility. *Journal of Logical and Algebraic Methods in Programming*, 86(1):77 – 106, 2017.

[5] S. Awodey. *Category Theory, Second Edition*. Oxford Logic Guides. Oxford University Press, 2010.

[6] Holger Bock Axelsen and Robin Kaarsgaard. Join inverse categories as models of reversible recursion. In Bart Jacobs and Christof Löding, editors, *FOSSACS*, volume 9634 of *LNCS*, pages 73–90. Springer, 2016.

[7] Giorgio Bacci, Vincent Danos, and Ohad Kammar. On the statistical thermodynamics of reversible communicating processes. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *CALCO*, volume 8659 of *LNCS*, pages 1–18. Springer, 2011.

[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] Paolo Baldan and Silvia Crafa. A logic for true concurrency. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *LNCS*, pages 147–161. Springer, 2010.

[10] Paolo Baldan and Tommaso Padoan. Local model checking in a logic for true concurrency. In Javier Esparza and Andrzej S. Murawski, editors, *FOSSACS*, volume 10203 of *ECTS*, pages 407–423. Springer, 2017.

[11] Franco Barbanera, Mariangiola Dezani, and Ugo de'Liguoro. Compliance for reversible client/server interactions. In *BEAT*, volume 162, pages 35–42. EPTCS, 2014.

[12] Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de'Liguoro. Reversible client/server interactions. *Formal Aspects of Computing*, 28:697–722, 2016.

[13] Franco Barbanera, Ivan Lanese, and Ugo de'Liguoro. Retractable and speculative contracts. In Jean-Marie Jacquet and Mieke Massink, editors, *COORDINATION*, pages 119–137. Springer, 2017.

[14] Kamila Barylska, Evgeny Erofeev, Maciej Koutny, Łukasz Mikulski, and Marcin Piątkowski. Reversing transitions in bounded Petri nets. *Fundamenta Informaticae*, 157(4):341–357, 2018.

[15] Marek A Bednarczyk. *CATEGORIES OF ASYNCHRONOUS SYSTEMS*. PhD thesis, University of Sussex, 1988.

[16] Marek a. Bednarczyk. Hereditary history preserving bisimulations or what is the power of the future perfect in program logics. Technical report, Institute of Computer Science, Polish Academy of Sciences, Gdańsk, 1991.

[17] Michele Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195(2):205 – 226, 1998.

[18] Michele Boreale and Davide Sangiorgi. A fully abstract semantics for causality in the π-calculus. *Acta Informatica*, 35(5):353–400, 1998.

[19] Gérard Boudol. Flow event structures and flow nets. In Irène Guessarian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *LNCS*, pages 62–95. Springer, 1990.

[20] Gérard Boudol and Ilaria Castellani. On the semantics of concurrency: Partial orders and transition systems. In Hartmut Ehrig, Robert Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT*, volume 249 of *LNCS*, pages 123–137. Springer, 1987.

[21] Gérard Boudol and Ilaria Castellani. Permutation of transitions: An event structure semantics for CCS and SCCS. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 411–427. Springer, 1989.

[22] Gérard Boudol and Ilaria Castellani. Three equivalent semantics for CCS. In Irène Guessarian, editor, *Semantics of Systems of Concurrent Processes*, pages 96–141. Springer, 1990.

[23] Gérard Boudol and Ilaria Castellani. Flow models of distributed computations : event structures and nets. Research Report RR-1482, INRIA, 1991.

[24] Gérard Boudol and Ilaria Castellani. Flow models of distributed computations: Three equivalent semantics for CCS. *Information and Computation*, 114(2):247 – 314, 1994.

[25] William J Bowman, Roshan P James, and Amr Sabry. Dagger traced symmetric monoidal categories and reversible programming. In *RC*, pages 51—56. Ghent University, 2011.

[26] E. Cardoza, R. Lipton, and A. R. Meyer. Exponential space complete problems for Petri nets and commutative semigroups (preliminary report). In *STOC*, STOC '76, page 50–54, New York, NY, USA, 1976. Association for Computing Machinery.

[27] Simon Castellan, Jonathan Hayman, Marc Lasson, and Glynn Winskel. Strategies as concurrent processes. *Electronic Notes in Theoretical Computer Science*, 308:87–107, 2014.

[28] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Concurrent reversible sessions. In Roland Meyer and Uwe Nestmann, editors, *CONCUR*, volume 85 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:17, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[29] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Reversible sessions with flexible choices. *Acta Informatica*, 56(7-8):553–583, 2019.

[30] Ilaria Castellani and Zhang Guo-Qiang. Parallel product of event structures. *Theoretical Computer Science*, 179(1):203 – 215, 1997.

[31] Silvia Crafa, Daniele Varacca, and Nobuko Yoshida. Compositional event structure semantics for the internal π-calculus. In Luís Caires and Vasco T. Vasconcelos, editors, *CONCUR*, pages 317–332. Springer, 2007.

[32] Silvia Crafa, Daniele Varacca, and Nobuko Yoshida. Event structure semantics of parallel extrusion in the pi-calculus. In Lars Birkedal, editor, *FOSSACS*, volume 7213 of *LNCS*, pages 225–239. Springer, 2012.

[33] Ioana Cristescu, Jean Krivine, and Daniele Varacca. A compositional semantics for the reversible pi-calculus. In *LICS*, pages 388–397. IEEE, 2013.

[34] Ioana Cristescu, Jean Krivine, and Daniele Varacca. Rigid families for the reversible π-calculus. In *RC*, volume 9720 of *LNCS*, pages 3–19. Springer, 2016.

[35] Vincent Danos and Jean Krivine. Reversible communicating systems. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *LNCS*, pages 292–307. Springer, 2004.

[36] Vincent Danos and Jean Krivine. Transactions in RCCS. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3170 of *LNCS*, pages 398–412. Springer, 2004.

[37] Vincent Danos and Jean Krivine. Formal Molecular Biology Done in CCS-R. *Electronic Notes in Theoretical Computer Science*, 180(3):31 – 49, 2007. BioConcur.

[38] Vincent Danos, Jean Krivine, and Paweł Sobociński. General reversibility. *Electronic Notes in Theoretical Computer Science*, 175(3):75–86, June 2007.

[39] Edsko de Vries, Vasileios Koutavas, and Matthew Hennessy. Communicating Transactions. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *LNCS*, pages 569–583. Springer, 2010.

[40] P. Degano, R. De Nicola, and U. Montanari. On the consistency of 'truly concurrent' operational and denotational semantics. In *LICS*, pages 133–141, 1988.

[41] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. A partial ordering semantics for ccs. *Theoretical Computer Science*, 75(3):223–262, 1990.

[42] Pierpaolo Degano and Corrado Priami. Non-interleaving semantics for mobile processes. *Theoretical Computer Science*, 216(1-2):237–270, 1999.

[43] Mariangiola Dezani-ciancaglini and Paola Giannini. Reversible multiparty sessions with checkpoints. *Electronic Proceedings in Theoretical Computer Science*, 222:60–74, 2016.

[44] Javier Esparza and Mogens Nielsen. Decidability issues for Petri nets. *Petri nets newsletter*, 94:5–23, 1994.

[45] Harald Fecher, Mila Majster-Cederbaum, and Jinzhao Wu. Bundle event structures: A revised cpo approach. *Information Processing Letters*, 83(1):7 – 12, 2002.

[46] Elena Giachino, Ivan Lanese, and Claudio Antares Mezzina. Causal-consistent reversible debugging. In Gnesi, Stefania and Rensink, Arend, editor, *FASE*, volume 8411 of *LNCS*, pages 370–384. Springer, 2014.

[47] Elena Giachino, Ivan Lanese, Claudio Antares Mezzina, and Francesco Tiezzi. Causal-consistent rollback in a tuple-based language. *Journal of Logical and Algebraic Methods in Programming*, 88:99–120, 2017.

[48] Eva Graversen, Iain Phillips, and Nobuko Yoshida. Towards a categorical representation of reversible event structures. In *PLACES*, volume 246 of *EPTCS*, pages 49–60, 2017.

[49] Eva Graversen, Iain Phillips, and Nobuko Yoshida. Event structure semantics of (controlled) reversible CCS. In Jarkko Kari and Irek Ulidowski, editors, *RC*, volume 11106 of *LNCS*, pages 102–122. Springer, 2018.

[50] Eva Graversen, Iain Phillips, and Nobuko Yoshida. Towards a categorical representation of reversible event structures. *Journal of Logical and Algebraic Methods in Programming*, 104:16 – 59, 2019.

[51] Eva Graversen, Iain Phillips, and Nobuko Yoshida. Event structures for the reversible early internal π-calculus. In Ivan Lanese and Mariusz Rawski, editors, *RC*, volume 12227 of *LNCS*, pages 71–90. Springer, 2020.

[52] Thomas Troels Hildebrandt, Christian Johansen, and Håkon Normann. A stable non-interleaving early operational semantics for the pi-calculus. *Journal of Logical and Algebraic Methods in Programming*, 104:227–253, 2019.

[53] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437 – 486, 1995.

[54] Stefan Kuhn and Irek Ulidowski. Local reversibility in a calculus of covalent bonding. *Science of Computer Programming*, 151:18–47, 2018. Special issue of the 8th Conference on Reversible Computation (RC2016).

[55] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, July 1961.

[56] Ivan Lanese. From reversible semantics to reversible debugging. In Jarkko Kari and Irek Ulidowski, editors, *RC*, volume 11106 of *LNCS*, pages 34–46. Springer, 2018.

[57] Ivan Lanese, Doriana Medić, and Claudio Antares Mezzina. Static versus dynamic reversibility in CCS. *Acta Informatica*, 58:1–34, 2019.

[58] Ivan Lanese, Claudio Antares Mezzina, Alan Schmitt, and Jean-Bernard Stefani. Controlling reversibility in higher-order pi. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR*, volume 6901 of *LNCS*, pages 297–311. Springer, 2011.

[59] Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. Reversing higher-order pi. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *LNCS*, pages 478–493. Springer, 2010.

[60] Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. Reversibility in the higher-order π-calculus. *Theoretical Computer Science*, 625:25 – 84, 2016.

[61] Ivan Lanese, Claudio Antares Mezzina, and Francesco Tiezzi. Causal-Consistent Reversibility. *Bulletin of the EATCS*, 114:17, November 2014.

[62] Ivan Lanese, Adrián Palacios, and Germán Vidal. Causal-consistent replay debugging for message passing programs. In Jorge A. Pérez and Nobuko Yoshida, editors, *FORTE*, pages 167–184. Springer, 2019.

[63] Ivan Lanese, Iain Phillips, and Irek Ulidowski. An axiomatic approach to reversible computation. In Jean Goubault-Larrecq and Barbara König, editors, *FOSSACS*, volume 12077 of *LNCS*, pages 442–461. Springer, 2020.

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

[65] George B. Leeman, Jr. A formal approach to undo operations in programming languages. *Transactions on Programming Languages and Systems*, 8(1):50–87, January 1986.

[66] Rita Loogen and Ursula Goltz. Modelling nondeterministic concurrent processes with event structures. *Fundamenta Informaticae*, 14(1):39–73, 1991.

[67] Doriana Medić, Claudio Antares Mezzina, Iain Phillips, and Nobuko Yoshida. A parametric framework for reversible π-calculi. *Information and Computation*, 275:104644, 2020.

[68] Doriana Medić and Claudio Antares Mezzina. Static vs dynamic reversibility in CCS. In Simon Devitt and Ivan Lanese, editors, *RC*, LNCS, pages 36–51. Springer, 2016.

[69] Claudio Antares Mezzina. On reversibility and broadcast. In Jarkko Kari and Irek Ulidowski, editors, *RC*, pages 67–83. Springer, 2018.

[70] Claudio Antares Mezzina and Vasileios Koutavas. A safety and liveness theory for total reversibility. In *TASE*, pages 1–8. IEEE, 2017.

[71] Claudio Antares Mezzina and Jorge A. Pérez. Reversibility in session-based concurrency: A fresh look. *Journal of Logical and Algebraic Methods in Programming*, 90:2 – 30, 2017.

[72] Microsoft. Microsoft time travel debugging. https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/time-travel-debugging-overview – 14:04 – 21/09/2020.

[73] Robin Milner, editor. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[74] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Information and Computation*, 100(1):1 – 40, 1992.

[75] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, ii. *Information and Computation*, 100(1):41 – 77, 1992.

[76] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *ICALP*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[77] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains. In Gilles Kahn, editor, *Semantics of Concurrent Computation*, volume 70 of *LNCS*, pages 266–284. Springer, 1979.

[78] Anna Philippou and Kyriaki Psara. Reversible computation in Petri nets. In Jarkko Kari and Irek Ulidowski, editors, *RC*, volume 11106 of *LNCS*, pages 84–101. Springer, 2018.

[79] Iain Phillips and Irek Ulidowski. Reversibility and models for concurrency. *Electronic Notes in Theoretical Computer Science*, 192(1):93–108, 2007.

[80] Iain Phillips and Irek Ulidowski. Reversing algebraic process calculi. *Journal of Logic and Algebraic Programming*, 73(1-2):70–96, 2007.

[81] Iain Phillips and Irek Ulidowski. Event identifier logic. *Mathematical Structures in Computer Science*, 24(2), 2014.

[82] Iain Phillips and Irek Ulidowski. Reversibility and asymmetric conflict in event structures. *Journal of Logical and Algebraic Methods in Programming*, 84(6):781 – 805, 2015.

[83] Iain Phillips, Irek Ulidowski, and Shoji Yuen. A Reversible Process Calculus and the Modelling of the ERK Signalling Pathway. In Robert Glück and Tetsuo Yokoyama, editors, *RC*, volume 7581 of *LNCS*, pages 218–232. Springer, 2013.

[84] Iain Phillips, Irek Ulidowski, and Shoji Yuen. Modelling of Bonding with Processes and Events. In Gerhard W. Dueck and D. Michael Miller, editors, *RC*, volume 7948 of *LNCS*, pages 141–154. Springer, 2013.

[85] Davide Sangiorgi. π-calculus, internal mobility, and agent-passing calculi. *Theoretical Computer Science*, 167(1):235 – 274, 1996.

[86] Undo Software. UndoDB - time travel reverse debugger for C/C++ on linux. undo.io/solutions/products/undodb-reverse-debugger – 13:51 – 21/09/2020.

[87] Michael Kirkedal Thomsen and Holger Bock Axelsen. Interpretation and programming of the reversible functional language RFUN. In Ralf Lämmel, editor, *IFL*, pages 8:1–8:13. ACM, 2015.

[88] Francesco Tiezzi and Nobuko Yoshida. Towards reversible sessions. In Alastair F. Donaldson and Vasco T. Vasconcelos, editors, *PLACES*, volume 155 of *EPTCS*, pages 17–24, 2014.

[89] Francesco Tiezzi and Nobuko Yoshida. Reversible session-based pi-calculus. *Journal of Logical and Algebraic Methods in Programming*, 84(5):684 – 707, 2015.

[90] Irek Ulidowski, Iain Phillips, and Shoji Yuen. Concurrency and reversibility. In Shigeru Yamashita and Shin-ichi Minato, editors, *RC*, volume 8507 of *LNCS*, pages 1–14. Springer, 2014.

[91] Frits W Vaandrager. A simple definition for parallel composition of prime event structures. CS R 8903, Centre for Mathematics and Computer Science, 1989.

[92] Rob van Glabbeek and Frits Vaandrager. Bundle event structures and CCSP. In Roberto Amadio and Denis Lugiez, editors, *CONCUR*, volume 2761 of *LNCS*, pages 57–71. Springer, 2003.

[93] Glynn Winskel. *Events in computation.* PhD thesis, University of Edinburgh, 1980.

[94] Glynn Winskel. Event structure semantics for CCS and related languages. In Mogens Nielsen and Erik Meineche Schmidt, editors, *ICALP*, volume 140 of *LNCS*, pages 561–576. Springer, 1982.

[95] Glynn Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 325–392. Springer, 1987.

[96] Glynn Winskel and Mogens Nielsen. *Models for Concurrency*, page 1–148. Oxford University Press, Inc., USA, 1995.

[97] Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. Principles of a reversible programming language. In Alex Ramírez, Gianfranco Bilardi, and Michael Gschwind, editors, *CF*, pages 43–54. ACM, 2008.

Appendix A

Proofs from Chapter 3

A.1 Proposition 3.3.9

Proof. [82] showed that Σ maps AESs to PESs, so we need to show that for any AES morphism $f : \mathcal{E} \to \mathcal{E}'$, $\Sigma(f) : \Sigma(\mathcal{E}) \to \Sigma(\mathcal{E}')$ is a PES morphism satisfying the conditions of Definition 3.2.4:

1. This is condition one of an AES morphism.
2. If $\Sigma(f)(e) \sharp' \Sigma(f)(e')$ then $f(e) \lhd' f(e')$ and $f(e) \rhd' f(e')$, which by definition of an AES morphism implies $e \lhd e'$ and $e \rhd e'$. Hence $e \sharp e'$ in $\Sigma(\mathcal{E})$.
3. If $\Sigma(f)(e) = \Sigma(f)(e')$ and $e \neq e'$ then, again by definition of an AES morphism, $e \lhd e'$ and $e \rhd e'$.

Having established that Σ maps between the categories, we show it to be a functor. The proof is split in three parts:

$\Sigma(f : \mathcal{E} \to \mathcal{E}') = \Sigma(f) : \Sigma(\mathcal{E}) \to \Sigma(\mathcal{E}')$**:** Follows trivially since $\Sigma(f) = f$, $\Sigma(E) = E$, and $\Sigma(E') = E'$.

$\Sigma(1_{\mathcal{E}}) = 1_{\Sigma(\mathcal{E})}$**:** Follows because the identity function for all PES and AES objects is $f(e) = e$.

$\Sigma(f \circ f') = \Sigma(f) \circ \Sigma(f')$**:** Follows trivially since $\Sigma(f) = f$ and $\Sigma(f') = f'$.

□

A.2 Proposition 3.4.6

Proof. We show that $f(X)$ fulfils the conditions of Definition 3.4.3:

1. For any $e, e' \in X_0$, if $f(e) \sharp_1 f(e')$, then $e \sharp_0 e'$, and therefore if X_0 is conflict-free then $f(X_0)$ is conflict-free.
2. There exists a sequence $e_1, \ldots, e_n$ $(n \geq 0)$, such that $X_0 = \{e_1, \ldots, e_n\}$ and for all i, $1 \leq i \leq n$, if $Y \mapsto e_{i+1}$ then $\{e_1, \ldots, e_i\} \cap Y \neq \emptyset$. We then get $f(X_0) = \{f(e_1), \ldots, f(e_0)\}$, and for all i, if $Y_1 \mapsto f(e_{i+1})$, then there exists Y_0 such that $Y_0 \mapsto e_{i+1}$, $f(Y_0) \subseteq Y_1$, and if $e' \in Y_0$, then $f(e') \neq \perp$. Since $Y_0 \cap \{e_1, \ldots, e_i\} \neq \emptyset$, we get that $Y_1 \cap \{f(e_1), \ldots, f(e_i)\} \neq \emptyset$.

□

A.3 Proposition 3.4.7

Proof. Partial functions are associative and $f(e) = e$ functions as a an identity arrow, so we need to show that the morphisms are composable:

If $\mathcal{E}_0 = (E_0, \mapsto_0, \sharp_0)$, $\mathcal{E}_1 = (E_1, \mapsto_1, \sharp_1)$, and $\mathcal{E}_2 = (E_2, \mapsto_2, \sharp_2)$ are BESs and $f : E_0 \to E_1$ and $g : E_1 \to E_2$ are morphisms, we show that $f \circ g : E_0 \to E_2$ is also a morphism:

1. If $g(f(e))\ \sharp_2\ g(f(e'))$ then $f(e)\ \sharp_1\ f(e')$, and therefore $e\ \sharp_0\ e'$.

2. If $g(f(e)) = g(f(e'))$ and $e \neq e'$, then either $f(e) = f(e')$, in which case $e\ \sharp_0\ e'$, or $f(e) \neq f(e')$, in which case $f(e)\ \sharp_1\ f(e')$, and therefore $e\ \sharp_0\ e'$.

3. If $X_2 \mapsto_2 g(f(e))$ then there exist $X_1 \subseteq E_1$ and $X_0 \subseteq E_0$ such that $X_1 \mapsto_1 f(e)$, $X_0 \mapsto_0 e$, $g(X_1) \subseteq X_2$, $f(X_0) \subseteq X_1$ and if $e_1 \in X_1$ then $g(e_1) \neq \bot$ and if $e_0 \in X_0$ then $f(e_0) \neq \bot$. This means that $g(f(X_0)) \subseteq X_2$, and if $e_0 \in X_0$ then $g(f(e_0)) \neq \bot$.

□

A.4 Proposition 3.4.10

Proof. We first show that π_0 and π_1 are morphisms:

1. If $\pi_i(e)\ \sharp_i\ \pi_i(e')$, then by definition $e\ \sharp\ e'$.

2. If $\pi_i(e) = \pi_i(e')$ and $e \neq e'$, then $\pi_{1-i}(e) \neq \pi_{1-i}(e')$, and therefore $e\ \sharp\ e'$.

3. If $X_i \mapsto \pi_i(e)$, then $\{e' \in E \mid \pi_i(e') \in X_i\} \mapsto e$. Clearly $\pi_i(\{e' \in E \mid \pi_i(e') \in X_i\}) = X_i$, and for all $e' \in \{e' \in E \mid \pi_i(e') \in X_i\})$, $\pi_i(e') \neq \bot$.

4. If X is a configuration of $\mathcal{E}_0 \times \mathcal{E}_1$, then we show that $\pi_i(X)$ satisfies the requirements of a configuration of $\mathcal{E}_i$:

   (a) As shown above $\pi_i(e)\ \sharp_i\ \pi_i(e') \Rightarrow e\ \sharp\ e'$, which means that X conflict-free implies $\pi_i(X)$ conflict-free.

   (b) If there exists a sequence $e_1, \ldots, e_n$ such that $X = \{e_1, \ldots, e_n\}$ and for all j, $1 \leq j \leq n$, if $Y \mapsto e_{j+1}$ then $\{e_1, \ldots, e_j\} \cap Y \neq \emptyset$, then $\pi_i(X) = \{\pi_i(e_1), \ldots, \pi_i(e_n)\}$ and if $\pi_i(e_{j+1}) \neq \bot$, then whenever $Y_i \mapsto \pi_i(e_{j+1})$, we have $\{e' \in E \mid \pi_i(e') \in Y_i\} \mapsto e_{j+1}$, meaning $\{e' \in E \mid \pi_i(e') \in Y_i\} \cap \{e_1, \ldots, e_j\} \neq \emptyset$. Therefore, we must get $Y_i \cap \{\pi_i(e_1), \ldots, \pi_i(e_j)\} \neq \pi_i(\emptyset) = \emptyset$.

We then show that for any BES, $\mathcal{E}_2 = (E_2, \mapsto_2, \sharp_2)$, if there exist morphisms $f_0 : \mathcal{E}_2 \to \mathcal{E}_0$ and $f_1 : \mathcal{E}_2 \to \mathcal{E}_1$, then there exists a unique morphism $f : \mathcal{E}_2 \to \mathcal{E}$, such that $f_2 \circ \pi_0 = f_0$ and $f_2 \circ \pi_1 = f_1$. Since BES-morphisms are functions, they are all unique.

We define f by $f(e) = (f_0(e), f_1(e))$, meaning the morphisms clearly commute as described above, and prove it to be a morphism:

1. If $f(e)\ \sharp\ f(e')$ then there exists $i \in \{0, 1\}$ such that either $\pi_i(f(e))\ \sharp_i\ \pi_i(f(e'))$, in which case clearly $f_i(e)\ \sharp_i\ f_i(e')_i$, and therefore $e\ \sharp_2\ e'$, or $\pi_i(f(e))) = \pi_i(f(e')) \neq \bot$ and $\pi_{1-i}(f(e)) \neq \pi_{1-i}(f(e'))$, in which case $f_i(e) = f_i(e') \neq \bot$, and $e \neq e'$, meaning $e\ \sharp_2\ e'$.

2. If $f(e) = f(e') \neq \bot$ then $f_0(e) = f_0(e') \neq \bot$ or $f_1(e) = f_1(e') \neq \bot$, meaning if $e \neq e'$ then $e\ \sharp_2\ e'$.

3. For $X \subseteq E$, if $X \mapsto f(e)$, then there exists $i \in \{0, 1\}$ and $X_i \subseteq E_i$ such that $X_i \mapsto \pi_i(e)$ and $X = \{e' \in E \mid \pi_i(e') \in X_i\}$. And since $X_i \mapsto f_i(e)$, there exists $X_2 \subseteq E_2$ such that $X_2 \mapsto_2 e$, $f_i(X_2) \subseteq X_i$, and if $e' \in X_2$ then $f_i(e') \neq \bot$. Clearly $f(X_2) \subseteq X$.

4. If X is a configuration of $\mathcal{E}_2$, then we show that $f(X)$ satisfies the requirements of a configuration of $\mathcal{E}_0 \times \mathcal{E}_1$:

   (a) As shown above, if $f(e)\ \sharp\ f(e')$, then $e\ \sharp_2\ e'$, meaning if X is conflict-free, then $f(X)$ is conflict-free.

   (b) If there exists a sequence $e_1, \ldots, e_n$ such that $X = \{e_1, \ldots, e_n\}$ and for all j, $1 \leq j \leq n$, if $Y \mapsto e_{j+1}$ then $\{e_1, \ldots, e_j\} \cap Y \neq \emptyset$, then $f(X) = \{f(e_1), \ldots, f(e_n)\}$ and if $f(e_{j+1}) \neq \bot$, then whenever $Y' \mapsto f(e_{j+1})$, we have $\{e' \in E \mid f(e') \in Y'\} \mapsto e_{j+1}$, meaning $\{e' \in E \mid f(e') \in Y'\} \cap \{e_1, \ldots, e_j\} \neq \emptyset$. Therefore, we must get $Y' \cap \{f(e_1), \ldots, f(e_j)\} \neq f(\emptyset) = \emptyset$.

□

A.5 Proposition 3.4.14

Proof. It is straightforward to show that $\mathcal{E}$ is a BES, and i_0 and i_1 are morphisms, so we simply need to prove that if there exists a BES $\mathcal{E}_2 = (E_2, \mapsto_2, \sharp_2)$ and morphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}_2$ and $f_1 : \mathcal{E}_1 \to \mathcal{E}_2$, then there exists a unique BES-morphism $f : \mathcal{E} \to \mathcal{E}_2$ such that $i_0 \circ f = f_0$ and $i_1 \circ f = f_1$.

Since $E_0 + E_1$, i_0, and i_1 make up a coproduct in the category of sets and partial functions, f must be unique.

We define f as $f((j,e)) = f_j(e)$ and prove it to be a morphism:

- If $f(e) \sharp_2 f(e')$ then $e = (j, e_j)$, $e' = (j', e_{j'})$ $f_j(e_j) = f(e)$, $f_{j'}(e_{j'}) = f(e')$, and either $j \neq j'$ or $j = j'$. If $j \neq j'$, then by definition $e \sharp e'$. If $j = j'$, then $f_j(e_j) \sharp_2 f_j(e_{j'})$, meaning $e_j \sharp_j e_{j'}$, and therefore $e \sharp e'$.
- If $f(e) = f(e') \neq \bot$ then $e = (j, e_j)$ and $e' = (j', e_{j'})$ and $f_j(e_j) = f(e) = f(e') = f_{j'}(e_{j'})$.

  If $j = j'$ then $e \neq e'$ means that $e_j \neq e_{j'}$, which means that $e_j \sharp_j e_{j'}$ and therefore $e \sharp e'$.

  If $j \neq j'$, then by definition $e \sharp e'$.
- If $X_2 \mapsto f(e)$, then $e = (j, e_j)$, and there exists X_j such that $X_j \mapsto_j e_j$, $f_j(X_j) \subseteq X_2$, and if $e'_j \in X_j$ then $f_j(e'_j) \neq \bot$. This means $\{(j, e'_j) \mid e'_j \in X_j\} \mapsto e$, $f(\{(j, e'_j) \mid e'_j \in X_j\}) \subseteq X_2$, and if $e' \in \{(j, e'_j) \mid e'_j \in X_j\}$ then $e' \neq \bot$.

And the diagram commutes. □

A.6 Propositions 3.5.5

Proof. Partial functions are associative and $f(e) = e$ works as a an identity arrow, and so we need to show that the morphisms are composable.

If $\mathcal{E}_0 = (E_0, \mapsto_0, \rhd_0)$, $\mathcal{E}_1 = (E_1, \mapsto_1, \rhd_1)$, and $\mathcal{E}_2 = (E_2, \mapsto_2, \rhd_2)$ are EBESs and $f : E_0 \to E_1$ and $g : E_1 \to E_2$ are morphisms, we show that $f \circ g : E_0 \to E_2$ is also a morphism:

1. If $g(f(e)) \lhd_2 g(f(e'))$ then $f(e) \lhd_1 f(e')$, and therefore $e \lhd_0 e'$.
2. If $g(f(e)) = g(f(e'))$ and $e \neq e'$, then either $f(e) = f(e')$, in which case $e \lhd_0 e'$, or $f(e) \neq f(e')$, in which case $f(e) \lhd_1 f(e')$, and therefore $e \lhd_0 e'$.
3. If $X_2 \mapsto_2 g(f(e))$ then there exist $X_1 \subseteq E_1$ and $X_0 \subseteq E_0$ such that $X_1 \mapsto_1 f(e)$, $X_0 \mapsto_0 e$, $g(X_1) \subseteq X_2$, $f(X_0) \subseteq X_1$ and if $e_1 \in X_1$ then $g(e_1) \neq \bot$ and if $e_0 \in X_0$ then $f(e_0) \neq \bot$. This means that $g(f(X_0)) \subseteq X_2$, and if $e_0 \in X_0$ then $g(f(e_0)) \neq \bot$.
4. If X_0 is a configuration of $\mathcal{E}_0$ then $f(X_0)$ is a configuration of $\mathcal{E}_1$, and therefore $g(f(X_0))$ is a configurations of $\mathcal{E}_2$.

□

A.7 Proposition 3.5.7

Proof. We first show that π_0 and π_1 are morphisms:

1. If $\pi_i(e) \lhd_i \pi_i(e')$, then by definition $e \lhd e'$.
2. If $\pi_i(e) = \pi_i(e')$ and $e \neq e'$, then $\pi_{1-i}(e) \neq \pi_{1-i}(e')$, and therefore $e \lhd e'$.
3. If $X_i \mapsto \pi_i(e)$, then $\{e' \in E \mid \pi_i(e') \in X_i\} \mapsto e$. Clearly $\pi_i(\{e' \in E \mid \pi_i(e') \in X_i\}) = X_i$, and for all $e' \in \{e' \in E \mid \pi_i(e') \in X_i\})$, $\pi_i(e') \neq \bot$.
4. If X is a configuration of $\mathcal{E}_0 \times \mathcal{E}_1$, then we show that $\pi_i(X)$ satisfies the requirements of a configuration of $\mathcal{E}_i$. We show that if the requirements of Definition 3.5.2 hold for $e_0, \ldots e_n$, then they hold for $\pi_i(e_0), \ldots, \pi_i(e_n)$:

(a) By definition $\{\pi_i(e_0), \ldots, \pi_i(e_n)\} = \pi_i(X)$.

(b) If $\pi_i(e_j) \lhd_i \pi_i(e_{j'})$, then as shown above, $e_j \lhd e_{j'}$, meaning $j < j'$.

(c) Whenever $Y_i \mapsto \pi_i(e_{j+1})$, we know $\{e' \in E \mid \pi_i(e') \in Y_i\} \mapsto e_{j+1}$, meaning $\{e' \in E \mid \pi_i(e') \in Y_i\} \cap \{e_1, \ldots, e_j\} \neq \emptyset$. Therefore, we must get $Y_i \cap \{\pi_i(e_1), \ldots, \pi_i(e_j)\} \neq \pi_i(\emptyset) = \emptyset$.

We then show that for any EBES, $\mathcal{E}_2 = (E_2, \mapsto_2, \rhd_2)$, if there exist morphisms $f_0 : \mathcal{E}_2 \to \mathcal{E}_0$ and $f_1 : \mathcal{E}_2 \to \mathcal{E}_1$, then there exists a unique morphism $f : \mathcal{E}_2 \to \mathcal{E}$, such that $f \circ \pi_0 = f_0$ and $f \circ \pi_1 = f_1$. Since EBES-morphisms are functions, they are all unique.

We define f by $f(e) = (f_0(e), f_1(e))$, meaning the morphisms clearly commute as described above, and prove it to be a morphism:

1. If $f(e) \lhd f(e')$ then there exists $i \in \{0, 1\}$ such that either $\pi_i(f(e)) \lhd_i \pi_i(f(e'))$, in which case clearly $f_i(e) \lhd_i f_i(e')_i$, and therefore $e \lhd_2 e'$, or $\pi_i(f(e))) = \pi_i(f(e')) \neq \bot$ and $\pi_{1-i}(f(e)) \neq \pi_{1-i}(f(e'))$, in which case $f_i(e) = f_i(e') \neq \bot$, and $e \neq e'$, meaning $e \lhd_2 e'$.

2. If $f(e) = f(e') \neq \bot$ then $f_0(e) = f_0(e') \neq \bot$ or $f_1(e) = f_1(e') \neq \bot$, meaning if $e \neq e'$ then $e \lhd_2 e'$.

3. For $X \subseteq E$, if $X \mapsto f(e)$, then there exists $i \in \{0, 1\}$ and $X_i \subseteq E_i$ such that $X_i \mapsto \pi_i(e)$ and $X = \{e' \in E \mid \pi_i(e') \in X_i\}$. And since $X_i \mapsto f_i(e)$, there exists $X_2 \subseteq E_2$ such that $X_2 \mapsto_2 e$, $f_i(X_2) \subseteq X_i$, and if $e' \in X_2$ then $f_i(e') \neq \bot$. Clearly $f(X_2) \subseteq X$.

4. If X is a configuration of $\mathcal{E}_2$, then we show that $f(X)$ satisfies the requirements of a configuration of $\mathcal{E}_0 \times \mathcal{E}_1$. We show that if the requirements of Definition 3.5.2 hold for $e_0, \ldots, e_n$, then they hold for $f(e_0), \ldots, f(e_n)$:

   (a) By definition $\{f(e_0), \ldots, f(e_n)\} = f(X)$.

   (b) If $f(e_j) \lhd f(e_{j'})$, then as shown above, $e_j \lhd_2 e_{j'}$, meaning $j < j'$.

   (c) Whenever $Y \mapsto f(e_{j+1})$, we know $\{e' \in E \mid f(e') \in Y\} \mapsto e_{j+1}$, meaning $\{e' \in E \mid f(e') \in Y\} \cap \{e_1, \ldots, e_j\} \neq \emptyset$. Therefore, we must get $Y \cap \{f(e_1), \ldots, f(e_j)\} \neq f(\emptyset) = \emptyset$.

□

A.8 Proposition 3.7.8

Proof. That $E_f((E, \prec, \sharp))$ is an ES was shown by [21]. We therefore need to show that if $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an FES morphisms, $E_f(f) : E_f(\mathcal{E}_0) \to E_f(\mathcal{E}_1)$ is an ES morphism satisfying the conditions of Definition 3.7.3.

1. For all $e_0 \in E_0$, if $f(e_0) \neq \bot$ and $X \vdash_0 e_0$, then $f(X) \vdash_1 f(e_0)$.

2. If $X \in \mathsf{Con}_0$ then $f(X) \in \mathsf{Con}_1$ because if $f(e) \sharp_1 f(e')$ then $e \sharp_0 e'$.

3. For all $e_0, e_0' \in E_0$, if $\{e_0, e_0'\} \in \mathsf{Con}_0$ and $f(e_0) = f(e_0') \neq \bot$ then $e_0 \not\sharp_0 e_0'$, and therefore $e_0 = e_0'$.

The rest of the proof is similar to previous functors. □

Appendix B

Proofs from Chapter 4

B.1 Proposition 4.1.4

Proof. Since CS morphisms are partial functions they are associative, and $f(e) = e$ is an identity arrow. Hence we only need to prove composability, meaning that given CSs $\mathcal{C}_0$, $\mathcal{C}_1$, and $\mathcal{C}_2$ and morphisms $f : \mathcal{C}_0 \to \mathcal{C}_1$ and $g : \mathcal{C}_1 \to \mathcal{C}_2$, we have $f \circ g : \mathcal{C}_0 \to \mathcal{C}_2$ is a morphism. Each condition of Definition 4.1.3 is proved separately:

1. Since f is a CS morphism, $f(X) \in \mathsf{C}_1$, and since g is a CS morphism, for any $X_1 \in \mathsf{C}_1$, $g(X_1) \in \mathsf{C}_2$, and therefore $g(f(X)) \in \mathsf{C}_2$.

2. Since f is a CS morphism, $f(X) \xrightarrow{f(A) \cup f(\underline{B})}_2 f(Y)$ and, since g is a CS morphism, this implies that $g(f(X)) \xrightarrow{g(f(A)) \cup g(f(\underline{B}))}_2 g(f(Y))$.

3. If $g(f(e_0)) = g(f(e_0')) \neq \bot$ then $f(e_0) = e_1$ and $f(e_0') = e_1'$ for some $e_1, e_1' \in E_1$ such that $g(e_1) = g(e_1') \neq \bot$. Hence either $e_1 = e_1'$ or no $X_1 \in \mathsf{C}_1$ exists such that $e_1, e_1' \in X_1$.

   In the first case, since f is a morphism, by definition, if $e_0 \neq e_0'$ then no $X \in \mathsf{C}_0$ exists such that $e_0, e_0' \in X$.

   In the second, by definition for any $X_0 \in \mathsf{C}_0$, $f(X_0) \in \mathsf{C}_1$, and therefore there cannot exist any $X \in \mathsf{C}_0$ such that $e_0, e_0' \in X$.

□

B.2 Proposition 4.1.6

Proof. Let $\mathcal{C}_0 = (E_0, F_0, \mathsf{C}_0, \to_0)$, $\mathcal{C}_1 = (E_1, F_1, \mathsf{C}_1, \to_1)$, and $\mathcal{C}_0 + \mathcal{C}_1 = (E, F, \mathsf{C}, \to)$. Then, since events from E_i interact with each other in the same way in $\mathcal{C}_0 + \mathcal{C}_1$ as in $\mathcal{C}_i$ and events from E_0 are never in the same configurations as events from E_1, i_0 and i_1 are clearly morphisms.

The coproduct $\mathcal{C}_0 + \mathcal{C}_1$ of $\mathcal{C}_0$ and $\mathcal{C}_1$ must satisfy the universal property that if there exist morphisms $f_0 : \mathcal{C}_0 \to \mathcal{C}_2$ and $f_1 : \mathcal{C}_1 \to \mathcal{C}_2$ then there exists a unique morphism $f : \mathcal{C}_0 + \mathcal{C}_1 \to \mathcal{C}_2$ such that $i_0 \circ f = f_0$ and $i_0 \circ f = f_1$. In this case we define f as $f((j, e)) = f_j(e)$.

By definition, $i_0 \circ f = f_0$ and $i_1 \circ f = f_1$, so we prove f is a morphism, meaning it satisfies the conditions in Definition 4.1.3:

1. Suppose $X \xrightarrow{A \cup \underline{B}} Y$ in $\mathcal{C}_0 + \mathcal{C}_1$. Then there exists $j \in \{0, 1\}$ and $X_j, Y_j, A_j, B_j \subseteq E_j$ such that $i_j(X_j) = X$, $i_j(Y_j) = Y$, $i_j(A_j) = A$, $i_j(B_j) = B$, and $X_j \xrightarrow{A_j \cup \underline{B_j}}_j Y_j$. Therefore $f_j(X_j) \xrightarrow{f_j(A_j) \cup \underline{f_j(B_j)}}_2 f_j(Y_j)$.

2. Suppose $X \in \mathsf{C}$. Then there exist $j \in \{0, 1\}$ and $X_j \in \mathsf{C}_j$ such that $i_j(X_j) = X$. This implies $f_j(X_j) \in \mathsf{C}_2$.

3. Suppose $f((j,e)) = f((j',e')) \neq \perp$ and $(j,e) \neq (j',e')$. Then either $j = j'$ or $j \neq j'$.

   If $j = j'$ then $f_j(e) = f_j(e')$, implying since $e \neq e'$, there cannot exist $X_j \in \mathsf{C}_j$ such that $e, e' \in X_j$, and therefore there does not exist $X \in \mathsf{C}$ such that $(j,e),(j',e') \in X$.

   If $j \neq j'$ then there does not exist $X \in \mathsf{C}$ such that $(j,e),(j',e') \in X$.

□

B.3 Proposition 4.1.8

Proof. Let $\mathcal{C}_0 = (E_0, F_0, \mathsf{C}_0, \to_0)$, $\mathcal{C}_1 = (E_1, F_1, \mathsf{C}_1, \to_1)$, and $\mathcal{C}_0 \times \mathcal{C}_1 = (E, F, \mathsf{C}, \to)$. Then π_0 and π_1 as in Definition 4.1.7 are clearly morphisms.

A product is formally defined in the following way: if there exist morphisms $f_0 : \mathcal{C}_2 \to \mathcal{C}_0$ and $f_1 : \mathcal{C}_2 \to \mathcal{C}_1$ then there exists a unique morphism $f : \mathcal{C}_2 \to \mathcal{C}_0 \times \mathcal{C}_1$ such that $f \circ \pi_0 = f_0$ and $f \circ \pi_1 = f_1$. We define f as $f(e) = (f_0(e), f_1(e))$, where $f_i(e) = *$ if it is not defined.

Clearly $f \circ \pi_0 = f_0$ and $f \circ \pi_1 = f_1$, so we simply need to show that f is a morphism, meaning it satisfies the conditions of Definition 4.1.3:

1. If $X \in \mathsf{C}_2$ then $f_0(X) \in \mathsf{C}_0$ and $f_1(X) \in \mathsf{C}_1$, and since f_0 and f_1 are morphisms, there cannot exist $e_2, e_2' \in X$ such that $f_0(e_2) = f_0(e_2') \neq \perp$ or $f_1(e_2) = f_1(e_2') \neq \perp$ and $e_2 \neq e_2'$, implying $f(X) \in \mathsf{C}$.

2. If $X \xrightarrow{A \cup \underline{B}}_2 Y$ then $f_0(X) \xrightarrow{f_0(A) \cup \underline{f_0(B)}}_0 f_0(Y)$ and $f_1(X) \xrightarrow{f_1(A) \cup \underline{f_1(B)}}_1 f_1(Y)$. Therefore $f(X) \xrightarrow{f(A) \cup \underline{f(B)}} f(Y)$.

3. If $f(e) = f(e') \neq \perp$ and $e \neq e'$ then for $j \in \{0,1\}$, we have $f_j(e) = f_j(e')$, and there exists $j' \in \{0,1\}$ such that $f_{j'}(e) = f_{j'}(e') \neq \perp$. This implies there does not exist $X \in \mathsf{C}_2$ such that $e, e' \in X$.

□

B.4 Proposition 4.1.10

Proof. It was established in [82] that $\boldsymbol{C}_p$ maps PESs to CSs, so we need to show that if $f : \mathcal{E}_0 \to \mathcal{E}_1$ is a PES morphism then $f : \boldsymbol{C}_p(\mathcal{E}_0) \to \boldsymbol{C}_p(\mathcal{E}_1)$ is a CS morphism satisfying the conditions of Definition 4.1.3:

1. Suppose $X \in \mathsf{C}_0$. Then X is conflict-free. Since, by definition of a PES morphism, $f(e_0) \sharp_1 f(e_0') \Rightarrow e_0 \sharp_0 e_0'$, this implies that $f(X)$ is conflict-free, and therefore $f(X) \in \mathsf{C}_1$.

2. Suppose $X \xrightarrow{A \cup \underline{B}}_0 Y$. Then $f(X) \xrightarrow{f(A) \cup \underline{f(B)}}_1 f(Y)$ because we can observe that F_0 and therefore $\boldsymbol{B}$ is empty and:

   (a) $f(X), f(Y) \in \mathsf{C}_1$ since $X, Y \in \mathsf{C}_0$, as implied by item 1.

   (b) $f(Y) = f(X) \cup f(A)$ since $Y = X \cup A$.

   (c) $f(A) \cap f(X) = \emptyset$ since $A \cap X = \emptyset$, and by definition of a PES morphism, for all $e_0, e_0' \in E_0$, if $f(e_0) = f(e_0') \neq \perp$ and $e_0 \neq e_0'$ then $e_0 \sharp_0 e_0'$, implying $e_0, e_0' \notin (X \cup A) = Y \in \mathsf{C}_0$, since Y is conflict-free.

   (d) For every $a \in f(A)$, $\{e \in E_1 \mid e <_1 f(a)\} \subseteq f(X)$ because for every $a_0 \in A$, $\{e \in E_0 \mid e <_0 a_0\} \subseteq X$, and by definition of a PES morphism, $\{e_1 \mid e_1 <_1 f(a_0)\} \subseteq \{f(e) \mid e <_0 a_0\}$.

3. Suppose $f(e_0) = f(e_0') \neq \perp$ and $e_0 \neq e_0'$. By definition of a PES morphism, $e_0 \sharp_0 e_0'$, and since all $X \in \mathsf{C}_0$ are conflict-free, there exists no $X \in \mathsf{C}_0$ such that $e_0, e_0' \in X$.

Then to prove it is a functor we simply need to show that:

$\boldsymbol{C}_p(f : \mathcal{E}_1 \to \mathcal{E}_2) = \boldsymbol{C}_p(f) : \boldsymbol{C}_p(\mathcal{E}_1) \to \boldsymbol{C}_p(\mathcal{E}_2)$, which we get because $\boldsymbol{C}_p(f) = f$, $\boldsymbol{C}_p(E_0) = E_0$, and $\boldsymbol{C}_p(E_1) = E_1$.

$C_p(1_{\mathcal{E}}) = 1_{C_p(\mathcal{E})}$ since the identity function for all PES and CS objects is $f(e) = e$.

$C_p(f \circ f') = C_p(f) \circ C_p(f')$ since $C_p(f) = f$ and $C_p(f') = f'$.

□

B.5 Proposition 4.1.12

Proof. It was established in [82] that C_a maps AESs to CSs, so we need to show that if $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an AES morphism then $f : C_a(\mathcal{E}_0) \to C_a(\mathcal{E}_1)$ is a CS morphism because it satisfies the conditions of Definition 4.1.3:

1. Suppose $X \in \mathsf{C}_0$. Then $\lhd_0$ is well-founded on X. Since, by definition of an AES morphism, $f(e_0) \lhd_1 f(e_0') \Rightarrow e_0 \lhd_0 e_0'$, this implies that $\lhd_1$ is well-founded on $f(X)$, and therefore $f(X) \in \mathsf{C}_1$.

2. Suppose $X \xrightarrow{A \cup \underline{B}}_0 Y$. Then $f(X) \xrightarrow{f(A) \cup f(\underline{B})}_1 f(Y)$ because we can observe that F_0 and therefore B must be empty, and:

   (a) $f(X), f(Y) \in \mathsf{C}_1$ because $X, Y \in \mathsf{C}_0$ and we already proved item 1.

   (b) $f(Y) = f(X) \cup f(A)$, since $Y = X \cup A$.

   (c) $f(A) \cap f(X) = \emptyset$ because $A \cap X = \emptyset$, and by definition of an AES morphism, for all $e_0, e_0' \in E_0$, if $f(e_0) = f(e_0') \neq \bot$ and $e_0 \neq e_0'$ then $e_0 \sharp_0 e_0'$. Hence $e_0, e_0' \notin (X \cup A) = Y \in \mathsf{C}_0$, since $\lhd_0$ is well-founded on Y.

   (d) For every $a \in f(A)$, there must exist an $a_0 \in A$ such that $f(a_0) = a$, and $\{e \in E_0 \mid e \rhd_0 a_0\} \cap (X \cup A) = \emptyset$. If there exists an $e_1 \in \{e \in E_1 \mid e \rhd_1 a\} \cap (f(X) \cup f(A))$ then there must also exist an $e_0 \in X \cup A$ such that $f(e_0) = e_1$. Since $f(e_0) \rhd_1 f(a_0)$, by definition of an AES morphism, $e_0 \rhd_0 a_0$, but because $\{e \in E_0 \mid e \rhd_0 a_0\} \cap (X \cup A) = \emptyset$ we have a contradiction. Therefore $\{e \in E_1 \mid e \rhd_1 a\} \cap (f(X) \cup f(A)) = \emptyset$.

   (e) For every $a \in f(A)$, $\{e \in E_1 \mid e <_1 a\} \subseteq f(X)$ because for every $a_0 \in A$, $\{e \in E_1 \mid e <_0 a_0\} \subseteq X$, and by definition of an AES morphism, $\{e_1 \mid e_1 <_1 f(a_0)\} \subseteq \{f(e) \mid e <_0 a_0\}$.

3. Suppose $f(e_0) = f(e_0') \neq \bot$ and $e_0 \neq e_0'$. By definition of an AES morphism, $e_0 \sharp_0 e_0'$, and since $\lhd_0$ is well-founded on all $X \in \mathsf{C}_0$ we have $e_0, e_0' \notin X$.

The rest of the proof is identical to that of Proposition 4.1.10. □

B.6 Proposition 4.1.16

Proof. We first prove that for any FES $\mathcal{E} = (E, \prec, \sharp)$, we get a CS $C(\mathcal{E}) = (E, F, \mathsf{C}, \to)$, meaning that whenever $X \xrightarrow{A \cup \underline{B}} Y$:

1. $A \cap X = \emptyset$, $B \subseteq F = \emptyset$, and $Y = X \cup A$ according to the definition of C_f.

2. For all $A' \subseteq A$ we have $X \xrightarrow{A'} X \cup A' \xrightarrow{A \setminus A'} Y$ because the conditions for transitions in Definition 4.1.15 are clearly satisfied for both transitions.

We then show that if $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an FES morphism then $f : C_f(\mathcal{E}_0) \to C_f(\mathcal{E}_1)$ is a CS morphism satisfying the conditions of Definition 4.1.3:

1. Suppose $X \in \mathsf{C}_0$. Then X is conflict free, and therefore $f(X)$ is conflict free.

2. Suppose $X \xrightarrow{A \cup \underline{B}}_0 Y$ and $f(A \cup B) \neq \emptyset$. Then $f(X) \xrightarrow{f(A) \cup f(\underline{B})}_1 f(Y)$ because:

   (a) $f(Y) = f(X) \cup f(A)$, since $Y = X \cup A$.

(b) $f(A) \cap f(X) = \emptyset$ because $A \cap X = \emptyset$, and for all $e, e' \in A \cup X$, $e \not\sharp_0 e'$, meaning either $e = e'$ or $f(e) \neq f(e')$.

(c) for all $e \in f(A)$, if $e' \prec_1 e$ and $e' \notin X$, then there exists $e'' \in f(X)$ such that $e' \sharp_1 e''$ and $e'' \prec_1 e$.

3. Suppose $f(e_0) = f(e'_0) \neq \bot$ and $e_0 \neq e'_0$. Then $e_0 \sharp e'_0$, and therefore there does not exist $X \subseteq E_0$ such that $e_0, e'_0 \in X$ and X is conflict free.

Having proved that C_f maps from **FES** to **CS** we prove it to be a functor similarly to previous functors. □

B.7 Proposition 4.1.18

Proof. We first prove that for any ES $\mathcal{E} = (E, \mathsf{Con}, \vdash)$, we generate a CS $C(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, meaning that whenever $X \xrightarrow{A \cup \underline{B}} Y$:

1. $A \cap X = \emptyset$, $B \subseteq F = \emptyset$, and $Y = X \cup A$ according to the definition of C.

2. For all $A' \subseteq A$, we have $X \xrightarrow{A'} X \cup A' \xrightarrow{A \setminus A'} Y$ because the conditions for transitions in Definition 4.1.17 are clearly satisfied for both transitions.

We then show that if $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an ES morphism then $f : C(\mathcal{E}_0) \to C(\mathcal{E}_1)$ is a CS morphism satisfying the conditions of Definition 4.1.3:

1. Suppose $X \in \mathsf{C}_0$. Then for all $X' \subseteq_{\text{fin}} X$, $X' \in \mathsf{Con}_0$. Hence $f(X') \in \mathsf{Con}_1$, and therefore $f(X) \in \mathsf{C}_1$.

2. Suppose $X \xrightarrow{A \cup \underline{B}}_0 Y$. Then $f(X) \xrightarrow{f(A) \cup f(\underline{B})}_1 f(Y)$ because we can observe that F_0 and therefore B must be empty, and:

   (a) $f(X), f(Y) \in \mathsf{C}_1$ since $X, Y \in \mathsf{C}_0$, this is implied by the first item.

   (b) $f(Y) = f(X) \cup f(A)$, since $Y = X \cup A$.

   (c) $f(A) \cap f(X) = \emptyset$ because $A \cap X = \emptyset$, and for all $e, e' \in Y$, $\{e, e'\} \in \mathsf{Con}_0$, meaning either $e = e'$ or $f(e) \neq f(e')$.

   (d) For every $a \in f(A)$, there exists an $X' \subseteq_{\text{fin}} f(X)$ such that $X' \vdash_1 a$ because for every $a_0 \in A$, there exists an $X'_0 \subseteq_{\text{fin}} X$ such that $X'_0 \vdash_0 a_0$, implying $f(X'_0) \vdash_1 f(a_0)$.

3. Suppose $f(e_0) = f(e'_0) \neq \bot$ and $e_0 \neq e'_0$. Then $\{e_0, e'_0\} \notin \mathsf{Con}_0$, and therefore there does not exist $X \subseteq E_0$ such that $e_0, e'_0 \in X$ and for all $X' \subseteq_{\text{fin}} X$, $X' \in \mathsf{Con}_0$.

Having proved that C maps from **ES** to **CS** we prove it to be a functor similarly to previous functors. □

B.8 Proposition 4.2.3

Proof. Since RPES morphisms are partial functions they are associative, and $f(e) = e$ is an identity arrow; so we only need to prove composability, meaning that given RPESs $\mathcal{E}_0$, $\mathcal{E}_1$, and $\mathcal{E}_2$ and morphisms $f : \mathcal{E}_0 \to \mathcal{E}_1$ and $g : \mathcal{E}_1 \to \mathcal{E}_2$, we have $f \circ g : \mathcal{E}_0 \to \mathcal{E}_2$ is a morphism.

Each of the six conditions of Definition 4.2.2 is proved separately:

1. If $g(f(e)) \neq \bot$ then there must exist $e_1 \in E_1$ and $e_2 \in E_2$ such that $f(e) = e_1$ and $g(e_1) = e_2$. Since f and g are morphisms, $\{e'_1 \mid e'_1 <_1 e_1\} \subseteq \{f(e') \mid e' <_0 e\}$ and $\{e'_2 \mid e'_2 <_2 e_2\} \subseteq \{g(e'_1) \mid e'_1 <_1 e_1\}$, and therefore $\{e'_2 \mid e'_2 <_2 g(f(e))\} \subseteq \{g(f(e')) \mid e' <_0 e\}$.

2. If $g(f(e)) \neq \bot \neq g(f(e'))$ then there must exist $e_1, e'_1 \in E_1$ and $e_2, e'_2 \in E_2$ such that $f(e) = e_1$, $f(e') = e'_1$, $g(e_1) = e_2$, and $g(e'_1) = e'_2$. By definition of a morphism, if $e_2 \sharp e'_2$ then $e_1 \sharp e'_1$, and if $e_1 \sharp e'_1$ then $e \sharp e'$.

3. Similar to the first condition.

4. Similar to the second condition.

5. If $g(f(e)) = g(f(e')) \neq \bot$ then there must exist $e_1, e_1' \in E_1$ and $e_2 \in E_2$ such that $f(e) = e_1$, $f(e') = e_1$, and $g(e_1) = g(e_1') = e_2$. If $e_1 = e_1'$ then, since f is a morphism, $e \sharp_0 e'$. If $e_1 \neq e_1'$ then $e_1 \sharp_1 e_1'$, implying that $e \sharp_0 e'$.

6. Follows from $f(F_0) \subseteq F_1$ and $g(F_1) \subseteq F_2$.

□

B.9 Proposition 4.2.5

Proof. Firstly, [82] showed that if $\mathcal{E}$ is an RPES, then $\boldsymbol{C}_{rp}(\mathcal{E})$ is a CS, so we show that if $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an RPES morphism, then $f : \boldsymbol{C}_{rp}(\mathcal{E}_0) \to \boldsymbol{C}_{rp}(\mathcal{E}_1)$ is a CS morphism, satisfying the conditions of Definition 4.1.3

1. Suppose $X \in \mathsf{C}_0$. Then X is conflict-free. Since, by definition of an RPES morphism, $f(e_0) \sharp_1 f(e_0') \Rightarrow e_0 \sharp_0 e_0'$, this implies $f(X)$ is conflict-free, and therefore $f(X) \in \mathsf{C}_1$.

2. Suppose $X \xrightarrow{A \cup \underline{B}} Y$. Then $f(X) \xrightarrow{f(A) \cup f(\underline{B})}_1 f(Y)$ according to Definition 4.2.4 because:

   (a) Since $X, Y \in \mathsf{C}_0$, we have $f(X), f(Y) \in \mathsf{C}_1$ according to previous item.

   (b) $f(Y) = (f(X) \setminus f(B)) \cup f(A)$, since $Y = (X \setminus B) \cup A$ and $B \subseteq X$.

   (c) $f(A) \cap f(X) = \emptyset$ because $A \cap X = \emptyset$, and by definition of an RPES morphism, for all $e_0, e_0' \in E_0$, if $f(e_0) = f(e_0') \neq \bot$ and $e_0 \neq e_0'$ then $e_0 \sharp e_0'$, implying $e_0, e_0' \notin (X \cup A)$, since $X \cup A$ is conflict-free.

   (d) $f(B) \subseteq f(X)$ because $B \subseteq X$.

   (e) $f(X) \cup f(A)$ is conflict-free because $X \cup A$ is conflict-free, and for all $e_0, e_0' \in E_0$, if $f(e_0) \sharp_1 f(e_0')$ then $e_0 \sharp_0 e_0'$.

   (f) For every $a_1 \in f(A)$, if $c_1 <_1 a_1$ then $c_1 \in f(X) \setminus f(B)$ because for every $a_0 \in A$, if $c_0 <_0 a_0$ then $c_0 \in X \setminus B$, implying $f(c) \in f(X) \setminus f(B)$, and by definition of an RPES $\{e_1 \mid e_1 <_1 f(a_0)\} \subseteq \{f(e') \mid e' <_0 a_0\}$.

   (g) For every $b_1 \in f(B)$, if $d_1 \prec_1 \underline{b_1}$ then $d_1 \in f(X) \setminus (f(B) \setminus \{b_1\}$ for similar reasons to above.

   (h) For every $b_1 \in f(B)$, if $d_1 \rhd_1 \underline{b_1}$ then $d_1 \notin f(X) \cup f(A)$ because, since $b_1 \in f(B)$ there must exist a $b_0 \in B$ such that $f(b_0) = \overline{b_1}$. If $d_1 \in f(X) \cup f(A)$ then there must exist a $d_0 \in X \cup A$ such that $f(d_0) = d_1$. Since $f(d_0) \rhd_1 f(b_0)$, by definition of an RAES morphism $d_0 \rhd_0 b_0$, implying $d_0 \notin X \cup A$ which is a contradiction.

3. Suppose $f(e_0) = f(e_0') \neq \bot$ and $e_0 \neq e_0'$. By definition of an RPES morphism, $e_0 \sharp_0 e_0'$, and since all $X \in \mathsf{C}_0$ are conflict-free, there cannot exist $X \in \mathsf{C}$ such that $e_0, e_0' \in X$.

Having shown that $\boldsymbol{C}_{rp}$ maps from **RPES** to **CS**, the rest of the proof is similar to previous functors. □

B.10 Proposition 4.2.9

Proof. Let $\mathcal{E}_0 = (E_0, F_0, <_0, \sharp_0, \prec_0, \rhd_0)$, $\mathcal{E}_1 = (E_1, F_0, <_1, \sharp_1, \prec_1, \rhd_1)$, and $\mathcal{E}_0 + \mathcal{E}_1 = \mathcal{E} = (E, F, <, \sharp, \prec, \rhd)$. Then it is straightforward to show that $\mathcal{E}$ is an RPES, and i_0 and i_1 are morphisms, so we simply need to prove that if there exists an RPES $\mathcal{E}_2 = (E_2, F_2, <_2, \sharp_2, \prec_2, \rhd_2)$ and morphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}_2$ and $f_1 : \mathcal{E}_1 \to \mathcal{E}_2$, then there exists a unique RPES morphism $f : \mathcal{E} \to \mathcal{E}_2$ such that $i_0 \circ f = f_0$ and $i_1 \circ f = f_1$.

Since $E_0 + E_1$, i_0, and i_1 comprise a coproduct in the category of sets and partial functions, f must be unique.

We define f as $f((j, e)) = f_j(e)$, giving us $i_0 \circ f = f_0$ and $i_1 \circ f = f_1$, and prove f to be a morphism:

1. Suppose $e_2 <_2 f(e)$. Then there exists a $j \in \{0,1\}$ and an $e_j \in E_j$ such that $e = (j, e_j)$ and $e_2 <_2 f_j(e_j)$. Since f_j is a morphism, there must exist an $e'_j \in E_j$ such that $f_j(e'_j) = e_2$ and $e'_j <_j e_j$. Hence that $(j, e'_j) < e$.

2. Suppose $f(e') \sharp_2 f(e)$. The $e' = (j', e_{j'})$ and $e = (j, e_j)$ for some $j, j' \in \{0,1\}$, $e_{j'} \in E_{j'}$, and $e_j \in E_j$, and therefore $f(e) = f_j(e_j)$ and $f(e') = f_{j'}(e_{j'})$.

   If $j = j'$, then $e_{j'} \sharp_j e_j$, and therefore $(j, e_{j'}) \sharp (j, e_j)$.

   If $j \neq j'$ then $e' \sharp e$.

3. Suppose $e_2 \prec_2 \underline{f(e)}$. Then there exists a $j \in \{0,1\}$ and an $e_j \in E_j$ such that $e = (j, e_j)$ and $e_2 \prec_2 \underline{f_j(e_j)}$. Since f_j is a morphism, there must exist an $e'_j \in E_j$ such that $f_j(e'_j) = e_2$ and $e'_j \prec_j \underline{e_j}$. Therefore $(j, e'_j) \prec \underline{e}$, and $f(j, e'_j) = e_2$.

4. Suppose $f(e') \rhd_2 \underline{f(e)}$. Then there must exist a $j \in \{0,1\}$ and $e_j, e'_j \in E_j$ such that $f_j(e_j) = f(e)$, $f_j(e'_j) = f(e')$ and $e'_j \sharp_j e_j$. Since $e'_j \rhd_j \underline{e_j}$, we get $(j, e'_j) \rhd \underline{(j, e_j)}$.

5. Suppose $f(e) = f(e') \neq \perp$. Then $e = (j, e_j)$ and $e' = (j', e_{j'})$ and $f_j(e_j) = f(e) = f(e') = f_{j'}(e_{j'})$.

   If $j = j'$ then $e \neq e'$ implies that $e_j \neq e_{j'}$. Hence $e_j \sharp_j e_{j'}$ and therefore $e \sharp e'$.

   If $j \neq j'$ then by definition $e \sharp e'$.

6. Suppose $e \in F$ and $f(e) \neq \perp$. Then there exist $j \in \{0,1\}$ and $e_j \in F_j$ such that $e = (j, e_j)$ and $f_j(e_j) = f(e)$. Since $e_j \in F_j$, we get $f_j(e) \in F_2$.

□

B.11 Proposition 4.2.15

Proof. Firstly, [82] showed that given an RPES $\mathcal{E}$, we have $\Phi_p(\mathcal{E})$ is a PES, so we show that given an RPES morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$, we have $f : \Phi_p(\mathcal{E}_0) \to \Phi_p(\mathcal{E}_1)$ is a PES morphism, satisfying the conditions of Definition 3.2.4

1. This condition is the same as condition 1 of Definition 4.2.2 of an RPES morphism.

2. Suppose $f(e_0) \sharp'_1 f(e'_0)$. Then either $f(e_0) \sharp_1 f(e'_0)$, $f(e'_0) \sharp'_1 f(e_0)$, or there exists some $e_1 \in E_1$ such that $f(e''_0) < f(e'_0)$ and $f(e_0) \sharp'_1 f(e''_0)$.

   If $f(e_0) \sharp_1 f(e'_0)$ then by definition $e_0 \sharp_0 e'_0$, and therefore $e_0 \sharp'_0 e'_0$.

   If $f(e'_0) \sharp'_1 f(e_0)$ then by definition $e'_0 \sharp'_0 e_0$, implying $e_0 \sharp'_0 e_0$.

   If there exists some $e_1 \in E_1$ such that $e_1 <_1 f(e'_0)$ and $e_1 \sharp'_1 f(e''_0)$, then there exists an $e''_0 \in E_0$ such that $f(e''_0) = e_1$ and $e''_0 <_0 e'_0$. This implies $e''_0 \sharp'_0 e_0$, and therefore $e_0 \sharp'_0 e'_0$.

3. Suppose $f(e_0) = f(e'_0)$ and $e'_0 \neq e_0$. Then, by condition 5 of an RPES morphism, $e_0 \sharp_0 e'_0$, and therefore $e_0 \sharp' e'_0$

Having shown that Φ_p maps from **RPES** to **PES**, the rest of the proof is similar to previous functors. □

B.12 Proposition 4.3.8

Proof. Let $\mathcal{E}_0 = (E_0, F_0, \prec_0, \lhd_0)$, $\mathcal{E}_1 = (E_1, F_1, \prec_1, \lhd_1)$, and $\mathcal{E}_0 + \mathcal{E}_1 = \mathcal{E} = (E, F, \prec, \lhd)$. Then clearly $\mathcal{E}$ is an RAES, and i_0 and i_1 as in Definition 4.3.7 are morphisms, and so we simply need to prove that if there exists an RAES $\mathcal{E}_2 = (E_2, F_2, \prec_2, \rhd_2)$ and morphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}_2$ and $f_1 : \mathcal{E}_1 \to \mathcal{E}_2$, then there exists a unique RAES morphism $f : \mathcal{E} \to \mathcal{E}_2$ such that $i_0 \circ f = f_0$ and $i_1 \circ f = f_1$.

Since $E_0 + E_1$, i_0, and i_1 comprise a coproduct in the category of sets and partial functions, f must be unique.

We define f as $f((j,e)) = f_j(e)$. It is clear that $i_0 \circ f = f_0$ and $i_1 \circ f = f_1$. So it remains to show that f is an RAES morphism:

1. If $e_2 \prec_2 f(e)^*$ then there exists a $j \in \{0,1\}$ and an $e_j^* \in E_j \cup \underline{F_j}$ such that $e^* = (j,e_j)^*$ and $e_2 \prec_2 f_j(e_j)^*$. Since f_j is a morphism, there must exist an $e_j' \in E_j$ such that $f_j(e_j') = e_2$ and $e_j' \prec_j e_j^*$. Hence $(j,e_j') \prec e^*$.

2. If $f(e')^* \lhd_2 f(e)$ then there must exist a $j \in \{0,1\}$ and $e_j \in E_j$ and $e_j'^* \in E_j \cup \underline{F_j}$ such that $f_j(e_j) = f(e)$, $f_j(e_j') = f(e')$ and $e_j'^* \lhd_j e_j$. Since $e_j'^* \lhd_j e_j$, we have $(j,e_j')^* \lhd (j,e_j)$.

3. If $f(e) = f(e') \neq \bot$ then $e = (j,e_j)$ and $e' = (j',e_{j'})$ and $f_j(e_j) = f(e) = f(e') = f_{j'}(e_{j'})$.

   If $j = j'$ then $e \neq e'$ implies $e_j \neq e_{j'}$. Hence $e_j \sharp_j e_{j'}$ and therefore $e \sharp e'$.

   If $j \neq j'$ then by definition $e \sharp e'$.

4. For $j \in \{0,1\}$, if $(j,e_j) \in F$ then $e_j \in F_j$, and $f_j(F_j) \subseteq F_2$.

□

B.13 Proposition 4.3.3

Proof. [82] showed that for an RAES $\mathcal{E}$, we have $C_{ra}(\mathcal{E})$ is a CS.

Let $\mathcal{E}_0 = (E_0, F_0, \prec_0, \lhd_0)$ and $\mathcal{E}_1 = (E_1, F_1, \prec_1, \lhd_1)$ be RAESs, and let $C_{ra}(\mathcal{E}_0) = (E_0, F_0, \mathsf{C}_0, \rightarrow_0)$ and $C_{ra}(\mathcal{E}_1) = (E_1, F_1, \mathsf{C}_1, \rightarrow_1)$. Suppose $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an RAES morphism. Then $f : C_{ra}(\mathcal{E}_0) \to C_{ra}(\mathcal{E}_1)$ is a CS morphism because it satisfies the conditions of Definition 4.1.3:

1. If $X \in \mathsf{C}_0$, then $\lhd_0$ is well-founded on X. Since, by definition of an RAES morphism, $f(e_0) \lhd_1 f(e_0') \Rightarrow e_0 \lhd_0 e_0'$, this implies that $\lhd_1$ is well-founded on $f(X)$, and therefore $f(X) \in \mathsf{C}_1$.

2. $f(X) \xrightarrow{f(A) \cup f(\underline{B})}_1 f(Y)$ according to Definition 4.3.2:

   (a) Since $X, Y \in \mathsf{C}_0$, this is implied by the first item.

   (b) Follows from $Y = (X \setminus B) \cup A$ and $B \subseteq X$.

   (c) $A \cap X = \emptyset$, and by definition of an RAES morphism, for all $e_0, e_0' \in E_0$, if $f(e_0) = f(e_0') \neq \bot$ and $e_0 \neq e_0'$ then $e_0 \sharp_0 e_0'$, implying $e_0, e_0' \notin (X \cup A)$, since $\lhd_0$ is well-founded on X, and for every $a_0 \in A$, if $c_0 \rhd_0 a_0$ then $c_0 \notin X \cup A$.

   (d) $B \subseteq X$.

   (e) For every $a_0 \in A$, if $c_0 \prec_0 a_0$ then $c_0 \in X \setminus B$, implying $f(c) \in f(X) \setminus f(B)$, and by definition of an RAES $\{e_1 \mid e_1 \prec_1 f(a_0)\} \subseteq \{f(e') \mid e' \prec_0 a_0\}$.

   (f) Since $a_1 \in f(A)$ there must exist an $a_0 \in A$ such that $f(a_0) = a_1$. If $c_1 \in f(X) \cup f(A)$ then there must exist a $c_0 \in X \cup A$ such that $f(c_0) = c_1$. Since $f(c_0) \rhd_1 f(a_0)$, by definition of an RAES morphism $c_0 \rhd_0 a_0$, implying $c_0 \notin X \cup A$ which is a contradiction.

   (g) Similar to above.

   (h) Similar to above.

3. By definition of an RAES morphism, $e_0 \sharp_0 e_0'$, and since $\lhd_0$ is well-founded on all $X \in \mathsf{C}_0$, $e_0, e_0' \notin X$.

Having established that C_{ra} is a mapping, the rest of the proof is similar to previous functors. □

B.14 Proposition 4.3.11

Proof. According to [82] if $\mathcal{E}$ is an RAES, then $\Phi_a(\mathcal{E})$ is an AES.

If $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an RAES morphism then $\Phi_a(f) = f' : \Phi_a(\mathcal{E}_0) \to \Phi_a(\mathcal{E}_1)$ is a AES morphism because it satisfies the conditions of Definition 3.3.3:

1. Suppose $e_1 <'_1 f'(e_0)$. Then there must exist some sequence $e_1 \prec_1 e_{1_1} \prec \cdots \prec_1 e_{1_k} \prec_1 f(e_0)$ for some $0 \leq k$. Since f is an RAES morphism, there must exists a corresponding sequence $e'_0 \prec_0 e_{0_1} \prec_0 \cdots \prec_0 e_{1_k} \prec_0 e_0$ such that $f(e'_0) = e_1$ and $f(e_{0_i}) = e_{1_i}$. This implies that $e'_0 <'_0 e_0$.

2. Suppose $f'(e_0) \lhd'_1 f'(e'_0)$. Then either $f'(e_0) \lhd_1 f'(e'_0)$, $f'(e_0) <'_1 f'(e'_0)$, or there exists some $e_1 \in E_1$ such that $e_1 <'_1 f'(e'_0)$ and $f'(e_0) \sharp'_1 e_1$.

   If $f'(e_0) \rhd_1 f'(e'_0)$ then by definition $e_0 \rhd_0 e'_0$, and therefore $e_0 \rhd'_0 e'_0$.

   If $f'(e'_0) <'_1 f'(e_0)$ then there exists e''_0 such that $f'(e''_0) = f'(e'_0)$ and $e''_0 <'_0 e_0$. If $e''_0 = e'_0$ then by definition $e'_0 <'_0 e_0$, implying $e_0 \rhd'_0 e'_0$.

   If there exists some $e_1 \in E_1$ such that $e_1 <'_1 f'(e'_0)$ and $f'(e_0) \sharp'_1 e_1$, then there exists an $e''_0 \in E_0$ such that $f'(e''_0) = e_1$ and $e''_0 <'_0 e'_0$. Hence $e_0 \sharp'_0 e''_0$, and therefore $e_0 \lhd'_0 e'_0$.

3. Suppose $f(e_0) = f(e'_0)$ and $e_0 \neq e'_0$. Then according to the definition of an RAES morphism, $e_0 \rhd_0 e'_0$, and therefore $f(e_0) \rhd'_0 f(e'_0)$.

Once established that Φ_a is a mapping from **RAES** to **AES**, the rest of the proof is similar to previous functors. □

B.15 Proposition 4.3.16

Proof. Firstly, [82] showed that if $\mathcal{E}$ is an RPES, then $A_r(\mathcal{E})$ is an RAES.

If $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an RPES morphism then $A_r(f) = f : A_r(\mathcal{E}_0) \to A_r(\mathcal{E}_1)$ is an RAES morphism because it satisfies the conditions of Definition 4.3.5:

1. According to the definition of A_r, we have $\prec'_0 = <_0 \cup \prec_0$ and $\prec'_1 = <_1 \cup \prec_1$, and according to the definition of an RPES morphism for all $e \in E_0$, if $f(e) \neq \bot$ then $\{e_1 \mid e_1 <_1 f(e)\} \subseteq \{f(e') \mid e' <_0 e\}$, and for all $e \in F_0$, if $f(e) \neq \bot$ then $\{e_1 \mid e_1 \prec_1 \underline{f(e)}\} \subseteq \{f(e') \mid e' \prec_0 \underline{e}\}$.

2. According to the definition of A_r, we have $\lhd'_0 = \ll_0 \cup \sharp_0 \cup \lhd_0$ and $\lhd'_1 = \ll_1 \cup \sharp_1 \cup \lhd_1$, and according to the definition of an RPES morphism for all $e, e' \in E_0$, if $f(e) \neq \bot \neq f(e')$ and $f(e) \sharp_1 f(e')$ then $e \sharp_0 e'$, and for all $e \in E_0$ and $e' \in F_0$, if $f(e) \neq \bot \neq f(e')$ and $f(e) \rhd_1 \underline{f(e')}$ then $e \rhd_0 \underline{e'}$.

   If $f(e') \ll_1 f(e)$ then $f(e') <_1 f(e)$ and either $f(e') \notin F_1$ or $f(e) \rhd \underline{f(e')}$. Since $\{e_1 \mid e_1 <_1 f(e)\} \subseteq \{f(e') \mid e' <_0 e\}$, we know $f(e') <_1 f(e)$ implies that there exists $e'' \in E_0$ such that $f(e'') = f(e')$ and $e'' <_0 e$. If $f(e') \notin F_1$ then, since $f(F_0) \subseteq F_1$, we have $e' \notin F_0$ and $e'' \ll_0 e$. If $f(e) \rhd_1 f(e')$ then, as shown above, $e \rhd_0 e'$ and again $e'' \ll_0 e$. If $e'' = e'$ then this is enough. Otherwise, since $\sharp_0$ is hereditary with respect to $\ll_0$ and $e'' \sharp_0 e'$, we get $e'' \sharp_0 e$.

3. This is item 5 of the definition of an RPES morphism.

4. This is item 6 of the definition of an RPES morphism.

Once established that A_r is a mapping from **RPES** to **RAES**, the rest of the proof is similar to previous functors. □

B.16 Proposition 4.3.20

Proof. For any CRAES $\mathcal{E} = (E, F, \prec, \lhd)$, we have $P'_a(\Phi_a(\mathcal{E})) = (E', E', \prec', \rhd')$. We define the unit $\eta : \mathcal{E} \to P'_p(\Phi_p(\mathcal{E}))$ as $\eta(e) = e$ for all $e \in E$. We show that η is a CRAES morphism, satisfying the conditions of Definition 4.3.5:

1. $\{e \mid e < \eta(e')\} = \{\eta(e) \mid e < e'\} \cap E'$ and if $e' \in F$ and $e \prec' \underline{\eta(e')}$, then $e = e'$ and therefore $e' \prec \underline{e}$.
2. Suppose $e \in E$, $e'^* \in E \cup \underline{F}$, and $e \rhd' e'^*$. If $e'^* = e'$ then clearly $e \rhd e'$. And if $e'^* = \underline{e'}$ then $e' \lhd e$, and since $\mathcal{E}$ is a CRAES, $e \rhd \underline{e'}$.
3. No $e, e' \in E$ exist such that $\eta(e) = \eta(e')$ but $e \neq e'$.
4. Clearly $F \subseteq E$.

Furthermore, η is clearly a natural transformation since $\eta(e) = e$.

Let $\mathcal{E}_0$ be a CRAES, $\mathcal{E}_1$ be an AES, and $g : \mathcal{E}_0 \to P'_a(\mathcal{E}_1)$ be a CRAES morphism. If $\bar{g} : \Phi_a(\mathcal{E}_0) \to \mathcal{E}_1$ is g restricted to E'_0, then clearly $g(\mathcal{E}_0) = \Phi_a(\bar{g}) \circ \eta(\mathcal{E}_0)$ and it is straightforward to show that $\bar{g}$ is an AES morphism.

1. Suppose $e_1 <_1 f(e)$. Then there exists an event $e_0 \in E_0$ such that $e_0 \prec_0 e$ and $f(e_0) = e_1$. We therefore need to show that $e_0 \in E'_0$. As $\mathcal{E}_0$ is a CRAES, $\ll_0 = \prec_{0_E}$, and therefore $\prec_{0_E} = \prec^+_{0_E}$. Hence any $\prec \cup \lhd$-cycle in $\{e' \in E \mid e' = e_0 \text{ or } e' \prec^+_E e\}$ would be a $\lhd$-cycle in $\{e' \mid e' = e_0 \text{ or } e' \prec e_0\}$, which is impossible.
2. Suppose $f(e) \lhd_1 f(e')$. Then $f(e) \lhd_1 f(e')$, and therefore $e \lhd_{0_E} e'$.

   Suppose $f(e) = f(e') \neq \bot$. Then $e \sharp e'$ by definition of an RAES morphism.

□

B.17 Proposition 4.4.5

Proof. We first show that if $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ is an RBES, then $\boldsymbol{C}_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \to)$ is a CS, as defined in Definition 4.1.1, meaning that for $X, Y \in \mathsf{C}$, $A \subseteq E$, and $B \subseteq F$, if $X \xrightarrow{A \cup \underline{B}} Y$, then:

1. $A \cap X = \emptyset$, $B \subseteq X \cap F$, and $Y = (X \setminus B) \cup A$ by definition.
2. For all $A' \subseteq A$ and $B' \subseteq B$, $(X \setminus B') \cup A = Z \in \mathsf{C}$ because $Z \subseteq X \cup A \in \mathsf{C}$. Moreover $X \xrightarrow{A' \cup \underline{B'}} Z$ and $Z \xrightarrow{(A \setminus A') \cup \underline{(B \setminus B')}} Y$ obviously fulfil the conditions for transitions.

We then show that if $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an RBES morphism then $f : \boldsymbol{C}_{br}(\mathcal{E}_0) \to \boldsymbol{C}_{br}(\mathcal{E}_1)$ is a CS morphism satisfying the conditions of Definition 4.1.3:

1. Suppose $X \in \mathsf{C}_0$. Then X is conflict-free. Since, by definition of an RBES morphism, $f(e_0) \sharp_1 f(e'_0) \Rightarrow e_0 \sharp_0 e'_0$, this implies that $f(X)$ is conflict-free, and therefore $f(X) \in \mathsf{C}_1$.
2. Suppose $X \xrightarrow{A \cup \underline{B}}_0 Y$. Then $f(X) \xrightarrow{f(A) \cup f(\underline{B})}_1 f(Y)$ because:
   (a) $f(X), f(Y) \in \mathsf{C}_1$ since $X, Y \in \mathsf{C}_0$, as implied by item 1.
   (b) $f(Y) = (f(X) \setminus f(B)) \cup f(A)$ since $Y = (X \setminus B) \cup A$.
   (c) $f(A) \cap f(X) = \emptyset$ since $A \cap X = \emptyset$, and by definition of an RBES morphism, for all $e_0, e'_0 \in E_0$, if $f(e_0) = f(e'_0) \neq \bot$ and $e_0 \neq e'_0$ then $e_0 \sharp_0 e'_0$, implying $e_0, e'_0 \notin (X \cup A) = Y \in \mathsf{C}_0$, since Y is conflict-free.
   (d) $f(B) \subseteq f(X)$ since $B \subseteq$.

(e) $f(X) \cup f(A)$ is conflict-free because $X \cup A$ is conflict-free and $f(e_0) \sharp_1 f(e_0') \Rightarrow e_0 \sharp_0 e_0'$.

(f) For all $e \in B$, if $e' \rhd \underline{e}$ then $e' \notin X \cup A$.

(g) For all $e_1 \in f(B)$, if $e_1' \rhd_1 \underline{e_1}$ then $e_1' \notin f(X) \cup f(A)$ because for any $e_0 \in B$, $e_0' \in E_0$ such that $f(e_0) = e_1$ and $f(e_0') = e_1'$ we have $e_0' \rhd \underline{e_0}$ and therefore $e_0' \notin X \cup A$.

(h) For all $e_1 \in f(A)$ and $X_1 \subseteq E_1$, if $X_1 \mapsto_1 e_1$ then $X_1 \cap (f(X) \setminus f(B)) \neq \emptyset$ because then there exists e_0 and X_0 such that $f(e_0) = e_1$, $X_0 \mapsto_0 e_0$, $f(X_0) \subseteq X_1$, and if $e_0' \in X_0$ then $f(e_0') \neq \bot$. Therefore, $X_0 \cap (X \setminus B) \neq \emptyset$, and $f(X_0) \cap (f(X) \setminus f(B)) \neq \emptyset$.

(i) For all $e_1 \in f(B)$ and $X_1 \subseteq E_1$, if $X_1 \mapsto_1 \underline{e_1}$ then $X_1 \cap (f(X)1 \setminus (f(B) \setminus \{e_1\})) \neq \emptyset$ for similar reasons to the previous condition.

(j) For every $a \in f(A)$, $\{e \in E_1 \mid e <_1 f(a)\} \subseteq f(X)$ because for every $a_0 \in A$, $\{e \in E_0 \mid e <_0 a_0\} \subseteq X$, and by definition of a PES morphism, $\{e_1 \mid e_1 <_1 f(a_0)\} \subseteq \{f(e) \mid e <_0 a_0\}$.

3. Suppose $f(e_0) = f(e_0') \neq \bot$ and $e_0 \neq e_0'$. By definition of an RBES morphism, $e_0 \sharp_0 e_0'$, and since all $X \in \mathsf{C}_0$ are conflict-free, there exists no $X \in \mathsf{C}_0$ such that $e_0, e_0' \in X$.

Then to prove it is a functor we simply need to show that:

$\mathcal{C}_{br}(f : \mathcal{E}_1 \to \mathcal{E}_2) = \mathcal{C}_{br}(f) : \mathcal{C}_{br}(\mathcal{E}_1) \to \mathcal{C}_{br}(\mathcal{E}_2)$, which is obvious since $\mathcal{C}_{br}(f) = f$, $\mathcal{C}_{br}(E_0) = E_0$, and $\mathcal{C}_{br}(E_1) = E_1$.

$\mathcal{C}_{br}(1_{\mathcal{E}}) = 1_{\mathcal{C}_{br}(\mathcal{E})}$ since the identity function for all RBES and CS objects is $f(e) = e$.

$\mathcal{C}_{br}(f \circ f') = \mathcal{C}_{br}(f) \circ \mathcal{C}_{br}(f')$ since $\mathcal{C}_{br}(f) = f$ and $\mathcal{C}_{br}(f') = f'$. □

B.18 Proposition 4.4.15

Proof. 1. By Proposition 4.4.14, X is forwards-reachable, meaning for every $e \in X$, if $X' \mapsto e$ then there exists e' such that $X' \cap X = \{e'\}$. For each $e_b \in B$, if $X_b \mapsto e_b$ and $X_b \cap X = \{e_b'\}$ then either $e_b' \notin F$ or $e_b \rhd \underline{e_b'}$, meaning $e_b' \notin B$, and therefore clearly $Y \xrightarrow{B} X$.

2. For any forwards-reachable configuration $X \in \mathsf{C}$ and $A, B \subseteq F$, if $X \xrightarrow{A \cup \underline{B}} (X \cup A) \setminus B$ then $(X \cup A) \setminus B \xrightarrow{B \cup \underline{A}} X$ according to Definition 4.4.4:

(a) Follows from Definition 4.4.13.

(b) Follows from Definition 4.4.13.

(c) Follows from Definition 4.4.13.

(d) Follows from Definition 4.4.13.

(e) For all $e \in A$ and $e' \in E$, if $e' \rhd \underline{e}$, then either $e' \sharp e$, or there exists $X' \subseteq E$ such that $X' \mapsto e'$ and $e \in X'$.

If $e' \sharp e$, then, as $X \cup A$ is conflict-free, $e' \notin X \cup A$.

If there exists $X' \subseteq E$ such that $X' \mapsto e'$ and $e \in X'$ then for all $e'' \in X' \setminus \{e\}$ we know $e'' \sharp e$, meaning $e'' \notin X \cup A$. This means $X \cap X' = \emptyset$, and therefore for all $X'' \subseteq X$, $X'' \xrightarrow{e'}\!\!\!\!\!\!\!\!/\;\;$, and consequently $e' \notin X \cup A$.

(f) For all $e \in B$ and $X' \subseteq E$, if $X' \mapsto e$, then, since X is forwards-reachable, $X' \cap X \neq \emptyset$. If $X' \cap X \setminus B = \emptyset$, then there exists $e' \in X' \cap B$. But this means $e \rhd \underline{e'}$, conflicting with $X \xrightarrow{A \cup \underline{B}}$.

(g) For all $e \in A$ and $X' \subseteq E$, if $X' \mapsto \underline{e}$, then $e \in X'$.

□

B.19 Proposition 4.4.18

Proof. For any RBES $\mathcal{E} = (E, F, \mapsto, \sharp, \triangleright)$, clearly $P_b(\Phi_b(\mathcal{E})) = (E, \emptyset, \sharp, \mapsto_E, \emptyset \times \emptyset)$ with $\mapsto_E = \mapsto \cap (2^E \times E)$, and we define the function $\eta : P_b(\Phi_b(\mathcal{E})) \to \mathcal{E}$ such that for all $e \in E$, $\eta(e) = e$, and prove that it is an RBES-morphism fulfilling the conditions of Definition 4.4.2:

1. Follows from Definitions 4.4.17 and 4.4.16.
2. If $f(e) = f(e')$ then $e = e'$.
3. This means $f(e)^* = f(e)$, and clearly if $X \mapsto_E f(e)$, then $X = f(X) \mapsto e$.
4. There are no $e' \in \emptyset$.

We then show that given a BES $\mathcal{E}_A = (E_A, \mapsto_A, \sharp_A)$, an RBES $\mathcal{E}_B = (E_B, F_B, \mapsto_B, \sharp_B, \triangleright_B)$, and an RBES morphism $g : P_p(A) \to B$, we get a BES morphism $f : A \to \Phi_p(B)$ according to Definition 3.4.4:

1. Follows from Definitions 4.4.17 and 4.4.16.
2. If $f(e) = f(e')$ then $e = e'$.
3. P_p and Φ_p do not affect $\mapsto_{A_E}$ or $\mapsto_{B_E}$.

□

B.20 Proposition 4.7.4

Proof. Since RES morphisms are partial functions they are associative, and $f(e) = e$ is the identity arrow. Then we only need to prove that given RESs $\mathcal{E}_0$, $\mathcal{E}_1$, and $\mathcal{E}_2$ and morphisms $f : \mathcal{E}_0 \to \mathcal{E}_1$ and $g : \mathcal{E}_1 \to \mathcal{E}_2$, we get a morphism $f \circ g : \mathcal{E}_0 \to \mathcal{E}_2$ satisfying the conditions in Definition 4.7.3:

1. Suppose $g(f(e)) \neq \bot$ and $X \otimes Y \vdash_0 e$. Then there exists an $e_1 \in E_1$ and an $e_2 \in E_2$ such that $f(e) = e_1$ and $g(e_1) = e_2$. It is clear from Definition 4.7.3 that there exists a $Y_1 \subseteq E_1$ such that for all $e_0' \in E_0$ such that $f(e_0') \in Y_1$, $e_0' \in Y$, and $f(X) \otimes Y_1 \vdash_1 f(e)^*$, and therefore there exists a $Y_2 \subseteq E_2$ such that for all $e_1' \in E_1$ such that $g(e_1') \in Y_2$, $e_1' \in Y_1$, and $g(f(X)) \otimes Y_2 \vdash_2 g(f(e))^*$.
2. Suppose $X_0 \in \mathsf{Con}_0$. Since f is a morphism, $f(X_0) \in \mathsf{Con}_1$, and using the same argument, $g(f(X_0)) \in \mathsf{Con}_2$.
3. Suppose $g(f(e)) = g(f(e')) \neq \bot$ and $e \neq e'$. Then $f(e) = e_1$ and $f(e') = e_1'$ for some $e_1, e_1' \in E_1$ such that $g(e_1) = g(e_1') \neq \bot$. Therefore either $e_1 = e_1'$ or no $X_1 \in \mathsf{Con}_1$ exists such that $e_1, e_1' \in X_1$.

   In the first case, since f is a morphism, by definition if $e \neq e'$ then no $X \in \mathsf{Con}_0$ exists such that $e, e' \in X$.

   In the second, by definition for any $X_0 \in \mathsf{Con}_0$, we get $f(X_0) \in \mathsf{Con}_1$, and therefore there cannot exist any $X \in \mathsf{Con}_0$ such that $e, e' \in X$.

□

B.21 Proposition 4.7.7

Proof. Let $\mathcal{E}_0 = (E_0, \mathsf{Con}_0, \vdash_0)$, $\mathcal{E}_1 = (E_1, \mathsf{Con}_1, \vdash_1)$, and $\mathcal{E}_0 + \mathcal{E}_1 = \mathcal{E} = (E, \mathsf{Con}, \vdash)$. Then it is simple to check that $\mathcal{E}$ is an RES and i_0 and i_1 are morphisms, so we simply need to prove that if there exists an RES $\mathcal{E}_2 = (E_2, F_2, \prec_2, \triangleright_2)$ and morphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}_2$ and $f_1 : \mathcal{E}_1 \to \mathcal{E}_2$, then there exists a unique RES morphism $f : \mathcal{E} \to \mathcal{E}_2$ such that $i_0 \circ f = f_0$ and $i_1 \circ f = f_1$.

Since $E_0 + E_1$, i_0, and i_1 comprise a coproduct in the category of sets and partial functions, f must be unique.

We define f as $f((j, e)) = f_j(e)$, giving us $i_0 \circ f = f_0$ and $i_1 \circ f = f_1$, and we prove that f is a morphism, satisfying the conditions of Definition 4.7.3:

1. Suppose $X \otimes Y \vdash e^*$ and $e = (j, e_j)$ for some $j \in \{0,1\}$. Then there exist $X_j, Y_j \subseteq E_j$ such that $i_j(X_j) = X$, $Y = i_j(Y_j) \cup (E \setminus i_j(E_j))$, and $X_j \otimes Y_j \vdash e_j$. If $f(e) \neq \bot$, meaning $f_j(e_j) \neq \bot$, then there exists a $Y_2 \subseteq E_2$ such that for all $e_j \in E_j$ if $f_j(e_j) \in Y_2$ then $e_j \in Y_j$, and $f_j(X_j) \otimes Y_2 \vdash_2 f(e)^*$. In addition, for all $(j', e') \in E$, such that $j \neq j'$, $(j', e') \in Y$, implying that if $f_{j'}((j', e')) \in Y_2$ then $(j', e') \in Y$.

2. Suppose $X \in \mathsf{Con}$. Then there must exist $j \in \{0,1\}$ and $X_j \in \mathsf{Con}_j$ such that $i_j(X_j) = X$. Since $X_j \in \mathsf{Con}_j$, we have $f(X) = f_j(X_j) \in \mathsf{Con}_2$.

3. Suppose $f(e) = f(e') \neq \bot$. Then $e = (j, e_j)$ and $e' = (j', e_{j'})$ and $f_j(e_j) = f(e) = f(e') = f_{j'}(e_{j'})$.

   If $j \neq j'$ then by definition no $X \in \mathsf{Con}$ exists such that $e, e' \in X$.

   If $j = j'$ then $e \neq e'$ implies that $e_j \neq e_{j'}$, and since f_j is an RES morphism we get that no $X_j \in \mathsf{Con}_j$ exists such that $e_j, e_{j'} \in X_j$. For there to exist an $X \in \mathsf{Con}$ such that $e, e' \in X$, we need an $X'_j \in \mathsf{Con}_j$ such that $i(X'_j) = X$, which does not exist.

□

B.22 Proposition 4.7.9

Proof. Let $\mathcal{E}_0 = (E_0, F_0, \mathsf{Con}_0, \vdash_0)$, $\mathcal{E}_1 = (E_1, F_1, \mathsf{Con}_1, \vdash_1)$, and $\mathcal{E}_0 \times \mathcal{E}_1 = \mathcal{E} = (E, F, \mathsf{Con}, \vdash)$. Then it is simple to check that $\mathcal{E}$ is an RES and π_0 and π_1, are RES morphisms. We then need to prove that for any RES $\mathcal{E}_2$ such that there exist RES morphisms $f_0 : \mathcal{E}_2 \to \mathcal{E}_0$ and $f_1 : \mathcal{E}_2 \to \mathcal{E}_1$, there exists a unique RES morphism $f : \mathcal{E}_2 \to (\mathcal{E}_0 \times \mathcal{E}_1)$ such $f \circ \pi_0 = f_0$ and $f \circ \pi_1 = f_1$.

Since $E_0 \times_* E_1$, π_0, and π_1 comprise a product in the category of sets and partial functions, f must be unique.

We define f as $f(e) = (f_0(e), f_1(e))$ and prove that it is a morphism, meaning it satisfies the conditions of Definition 4.7.3:

1. Suppose $X \otimes Y \vdash_2 e^*$ and $f(e) \neq \bot$. If $f_0(e) \neq \bot$ then there exists a $Y_0 \subseteq E_0$ such that for any $e' \in E_2$, if $f_0(e') \in Y_0$ then $e' \in Y$, and $f_0(X) \otimes Y_0 \vdash_0 f_0(e)^*$, and if $f_1(e) \neq \bot$ then there exists a $Y_1 \subseteq E_1$ such that for any $e' \in E_2$, if $f_1(e') \in Y_1$ then $e' \in Y$, and $f_1(X) \otimes Y_1 \vdash_1 f(e)^*$. We define $Y' = \{e \mid \pi_0(e) \in Y_0 \text{ or } \pi_1(e) \in Y_1\}$, where $Y_i = \emptyset$ if $f_i(e) = \bot$, and we get $f(X) \otimes Y' \vdash f(e)^*$ and for any $e' \in E_2$, if $f(e') \in Y'$ then $e' \in Y$.

2. Suppose $X \in \mathsf{Con}_2$. Then $f_0(X) \in \mathsf{Con}_0$ and $f_1(X) \in \mathsf{Con}_1$, and there are no $e, e' \in X$ such that $e \neq e'$ and either $f_0(e) = f_0(e')$ or $f_1(e) = f_1(e')$, implying $f(X) \in \mathsf{Con}$.

3. Suppose $f(e) = f(e') \neq \bot$ and $e \neq e'$. Then $f_0(e) = f_0(e')$ and $f_1(e) = f_1(e')$. Since we cannot have $f_0(e) = f_1(e) = \bot$ without $f(e) = \bot$, and f_0 and f_1 are morphisms, there cannot exist $X \in \mathsf{Con}_2$ such that $e, e' \in X$.

□

B.23 Proposition 4.7.12

Proof. We first show that if $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$ is an $(E, F, \mathsf{Con}, \vdash)$, then $\mathbf{C}_r(\mathcal{E}) = (E, F, \mathsf{C}, \to)$ is a CS, as defined in Definition 4.1.1, meaning that for $X, Y \in \mathsf{C}$, $A \subseteq E$, and $B \subseteq F$, if $X \xrightarrow{A \cup B} Y$, then:

1. $A \cap X = \emptyset$, $B \subseteq X \cap F$, and $Y = (X \setminus B) \cup A$ by definition.

2. For all $A' \subseteq A$ and $B' \subseteq B$, $(X \setminus B') \cup A = Z \in \mathsf{C}$ because $Z \subseteq X \cup A \in \mathsf{C}$. Moreover there exists a transition $X \xrightarrow{A' \cup B'} Z$ because for any $e \in A$, $X' \otimes Z' \vdash e$ for some $X' \subseteq_{\text{fin}} X \setminus B \subseteq X \setminus B'$, and $Z' \cap (X \cup A') \subseteq Z' \cap (X \cup A') = \emptyset$, and similarly for $b \in B'$. There exists a transition $Z \xrightarrow{(A \setminus A') \cup (B \setminus B')} Y$ for similar reasons.

We then need to show that $\mathcal{C}_r(\mathcal{E})$ is finitely enabled:

1. Clearly C is downwards-closed, since Con is downwards-closed.
2. $X \in \mathsf{C}$ if and only if for all $X' \subseteq_{\text{fin}} X$, $X' \in \mathsf{Con}$, implying $X' \in \mathsf{C}$.
3. If $X \xrightarrow{e}$ then there exist X', Z such that $X' \otimes Z \vdash e$, $X' \subseteq_{\text{fin}} X$, and $Z \cap (X \cup \{e\}) = \emptyset$. Hence $X'' \xrightarrow{e}$ whenever $X' \subseteq X'' \subseteq X$. Similarly for $X \xrightarrow{\bar{e}}$.

We then show that an RES morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$ is also a CS morphism (Definition 4.1.3) $f : \mathcal{C}_r(\mathcal{E}_0) \to \mathcal{C}_r(\mathcal{E}_1)$.:

1. This is implied by item 2 of the definition of an RES morphism.
2. Suppose $X \xrightarrow{A \cup \underline{B}}_0 Y$. Then $f(X) \xrightarrow{f(A) \cup f(\underline{B})}_1 f(Y)$ because:
   (a) $f(Y) = (f(X) \setminus f(B)) \cup f(A)$ because $Y = (X \setminus B) \cup A$ and $B \subseteq X$.
   $f(A) \cap f(X) = \emptyset$ because $A \cap X = \emptyset$, implying that if $e_1 \in f(A) \cap f(X)$ then there must exist $E_0 \in A$ and $e_0' \in X$ such that $e_1 = f(e_0) = f(e_0')$. However, by definition of an RES morphism, this implies that no $X' \in \mathsf{Con}_0$ exists such that $e_0, e_0' \in X'$, which cannot be true, since $X \cup A \in \mathsf{Con}_0$.
   $f(B) \subseteq f(X)$ because $B \subseteq X$.
   $f(X) \cup f(A) \in \mathsf{C}_1$ according to item 1 because $X \cup A \in \mathsf{C}_0$.
   (b) For all a in A, $X' \otimes Z \vdash a$ for some X', Z such that $X' \subseteq_{fin} X \setminus B$ and $Z \cap (X \cup A) = \emptyset$. If $X' \otimes Z \vdash a$ then there exists a Z' such that for all $e_0 \in E_0$, if $f(e_0) \in Z'$ then $e_0 \in Z$ and $f(X') \otimes Z' \vdash_1 f(e)^*$. Since $X \subseteq B$ and $X' \subseteq_{fin} X \setminus B$, we have $f(X') \subseteq_{fin} f(X) \setminus f(B)$, and since $e_0 \in Z$ for all $e_0 \in E_0$ such that $f(e_0) \in Z'$, we have $Z' \cap (f(X) \cup f(A)) \subseteq f(Z \cap (X \cup A)) = \emptyset$.
   (c) Similar to previous condition.
3. This is implied by item 3 of the definition of an RES morphism.

Having proved that $\mathcal{C}_r$ maps from **RES** to **FCS**, the proof of it being a functor is similar to previous functors. □

B.24 Proposition 4.7.15

Proof. We first show that given an FCS $\mathcal{C}$, then $\mathcal{R}(\mathcal{C})$ is an RES according to Definition 4.7.1:

1. Suppose $X \otimes Y \vdash e^*$. Then $X \cap Y = \emptyset$, and $Y = E \setminus (X' \cup A)$ for some X' and A such that $e^* \in A \cup \underline{X'}$. Therefore $(X \cup \{e\}) \cap Y = \emptyset$.
2. Suppose $X \otimes Y \vdash \underline{e}$, then clearly $e \in X$.
3. Suppose $X \otimes Y \vdash e^*$, $X \subseteq X' \in \mathsf{Con}$, and $X' \cap Y = \emptyset$. Then:
   (a) $X' \cap Y = \emptyset$;
   (b) if $e^* = \underline{e}$ then $e \in X \subseteq X'$;
   (c) and there exists $X'' \subseteq E$ such that $X'' \xrightarrow{A \cup \underline{B}}$ for some A and B such that $e^* \in A \cup \underline{B}$, $X'' \setminus B \subseteq X \subseteq X'$ and $Y = E \setminus (X'' \cup A)$.

We then show that given a CS morphism $f : \mathcal{C}_0 \to \mathcal{C}_1$, we have that $f : \mathcal{R}(\mathcal{C}_0) \to \mathcal{R}(\mathcal{C}_1)$ is an RES morphism, satisfying the conditions of Definition 4.7.3:

1. Suppose $f(e) \neq \perp$ and $X \otimes Y \vdash_0 e^*$. Then there exists an $X' \subseteq E$ such that $X' \xrightarrow{A \cup \underline{B}}_0$ for some A and B such that $e^* \in A \cup \underline{B}$, $X' \setminus B \subseteq X$, and $Y = E \setminus (X' \cup A)$.
   Then $f(X) \otimes (E_1 \setminus (f(X') \cup f(A))) \vdash_1 f(e)^*$ because:

(a) $X \cap Y = \emptyset$ and $f(Y) = f(E \setminus (X' \cup A)) \subseteq E_1 \setminus (f(X') \cup f(A))$, implying $f(X) \cap f(Y) = \emptyset$.

(b) If $f(e)^* = \underline{f(e)}$ then $e \in X$, implying $f(e) \in f(X)$.

(c) $f(X') \xrightarrow{f(A) \cup \underline{f(B)}}_1$, $f(e)^* \in f(A) \cup \underline{f(B)}$, and $f(X') \setminus f(B) \subseteq f(X)$.

And for all $e \in E_0$, if $f(e_0) \in (E_1 \setminus (f(X') \cup f(A))$, then clearly $e_0 \in E \setminus (X' \cup A)$.

2. Suppose $X \in \mathsf{Con}_0$. Then there exists $X_0' \in \mathsf{C}_0$ such that $X \subseteq_{\text{fin}} X_0'$. As $f(X_0') \in \mathsf{C}_1$, and $f(X) \subseteq_{\text{fin}} f(X_0')$, clearly $f(X) \in \mathsf{Con}_1$.

3. Suppose $f(e) = f(e')$ and $e \neq e'$. Then for all $X \in \mathsf{C}_0$, $\{e, e'\} \not\subseteq X$. Therefore for all $X' \in \mathsf{Con}_0$, $\{e, e'\} \not\subseteq X'$.

Having shown that R is a mapping from **CS** to **RES**, the rest of the proof is similar to previous functors. □

B.25 Theorem 4.7.17

Proof. For any RES $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$, we define $\epsilon : R(C_r(\mathcal{E})) \to \mathcal{E}$ such that for all $e \in E$ $\epsilon(e) = e$, and prove that it is an RES morphism.

If $C_r(\mathcal{E}) = (E, F, \mathsf{C}, \to)$ and $R(C_r(\mathcal{E})) = (E, \mathsf{Con}', \vdash')$ then:

1. Suppose $X \otimes Y \vdash' e^*$. Then clearly there exists $X' \subseteq E$ such that $X' \xrightarrow{A \cup \underline{B}}$ for some A, B such that $e^* \in A \cup \underline{B}$, $X' \setminus B \subseteq X$ and $Y = E \setminus (X' \cup A)$. And since $X' \xrightarrow{A \cup \underline{B}}$, we get $X'' \otimes Y' \vdash e^*$ for some X'', Y' such that $X'' \subseteq X' \setminus (B \setminus \{e\})$ and $Z \cap (X' \cup A) = \emptyset$. We then have $Y' \subseteq Y$, and since $X'' \subseteq X$ and $X \cap Y = \emptyset$ we get $X \otimes Y' \vdash e^*$.

2. Suppose $X \in \mathsf{Con}'$. Then $X \subseteq_{\text{fin}} X' \in \mathsf{C}$, $\mathsf{C} = \{X' \mid \forall X''' \subseteq_{\text{fin}} X'.X' \in \mathsf{Con}\}$, and Con is downwards closed, implying $f(X) \in \mathsf{Con}$.

3. No $e, e' \in E$ exist such that $\epsilon(e) = \epsilon(e')$ and $e \neq e'$.

Since $\epsilon(e) = e$, it is a natural transformation.

We then show that given an FCS $\mathcal{C}_0 = (E_0, F_0, \mathsf{C}_0, \to_0)$, an RES $\mathcal{E}_1 = (E_1, F_1, \mathsf{Con}_1, \vdash_1)$, and an RES morphism $f : R(\mathcal{C}_0) \to \mathcal{E}_1$, it holds that $f : \mathcal{C}_0 \to R(\mathcal{E}_1)$ is an FCS morphism.

Suppose $R(\mathcal{C}_0) = (E_0, F_0, \mathsf{Con}_0, \vdash_0)$ and $C_r(\mathcal{E}_1) = (E_1, F_1, \mathsf{C}_1, \to_1)$. Then f satisfies the conditions of Definition 4.1.3 because:

1. Suppose $C_0 \in \mathsf{C}_0$. Then for all $X \subseteq_{\text{fin}} C_0$, $X \in \mathsf{Con}_0$. Therefore $f(X) \in \mathsf{Con}_1$ if $f(X) \subseteq_{\text{fin}} f(C_0)$, and consequently $f(C_0) \in \mathsf{C}_1$.

2. Suppose $X \xrightarrow{A \cup \underline{B}}_0 Y$. Then $X \setminus B \otimes E \setminus (X \cup A) \vdash_0 e$ for all $e \in A$, and $X \setminus (B \setminus \{e\}) \otimes E \setminus (X \cup A) \vdash_0 \underline{e}$ for all $e \in B$. This implies that $f(X) \setminus (f(B) \setminus \{f(e)\}) \otimes Y_1 \vdash_1 f(e)^*$ for some Y_1 such that for all $e_0 \in E$, if $f(e_0) \in Y_1$ then $e_0 \in E \setminus (X \cup A)$. Therefore $f(X) \xrightarrow{f(A) \cup \underline{f(B)}}_1 f(Y)$.

3. If $C_0 \in \mathsf{C}_0$ then for all $X \subseteq_{\text{fin}} C_0$, $X \in \mathsf{Con}_0$, and if $f(e_0) = f(e_0') \neq \perp$, then there exists no $X_0 \in \mathsf{Con}_0$ such that $e_0, e_0' \in X_0$.

□

B.26 Proposition 4.7.18

Proof. Let $\mathcal{E}_0 = (E_0, F_0, \mathsf{Con}_0, \vdash_0)$, $\mathcal{E}_1 = (E_1, F_1, \mathsf{Con}_1, \vdash_1)$, $\mathcal{E}_0 + \mathcal{E}_1 = (E, F, \mathsf{Con}, \vdash)$, $C_r(\mathcal{E}_0) = (E_0, F_0, \mathsf{C}_0, \to_0)$, $C_1(\mathcal{E}_1) = (E_1, F_1, \mathsf{C}_1, \to_1)$, $C_r(\mathcal{E}_0 + \mathcal{E}_1) = (E, F, \mathsf{C}, \to)$, and $C_r(\mathcal{E}_0) + C_r(\mathcal{E}_1) = (E', F', \mathsf{C}', \to')$. Then $C_r(\mathcal{E}_0) + C_r(\mathcal{E}_1) = C_r(\mathcal{E}_0 + \mathcal{E}_1)$ because:

1. $C = \{X \mid \exists i \in \{0,1\}, X_i . X = \{i\} \times X_i \text{ and } \forall X'_i \subseteq_{\text{fin}} X_i . X'_i \in \mathsf{Con}_i\} = \{X \mid \forall X' \subseteq_{\text{fin}} X . \exists i \in \{0,1\}, X_i \in \mathsf{Con}_i . X' = \{i\} \times X_i\} = \mathsf{C}'$.
2. If $X \xrightarrow{A \cup \underline{B}} Y$ then $X \xrightarrow{A \cup \underline{B}}{}' Y$ because:
   (a) $Y = (X \setminus B) \cup A$, $A \cap X = \emptyset$, $B \subseteq X$, and $X \cup A$.
   (b) For all $(i,e) \in A$, there exists $X' \otimes Z \vdash_i e$ for some X', Z such that $\{i\} \times X' \subseteq_{\text{fin}} X \subseteq B$ and $\{i\} \times Z \cap (X \cup A) = \emptyset$.
   (c) For all $(i,e) \in B$, there exists $X' \otimes Z \vdash_i \underline{e}$ for some X', Z such that $\{i\} \times X' \subseteq_{\text{fin}} X \subseteq (B \subseteq (i,e))$ and $\{i\} \times Z \cap (X \cup A) = \emptyset$.
3. If $X \xrightarrow{A \cup \underline{B}}{}' Y$ then $X \xrightarrow{A \cup \underline{B}} Y$ for similar reasons.

$C_r(\mathcal{E}_0) \times C_r(\mathcal{E}_1) \cong C_r(\mathcal{E}_0 \times \mathcal{E}_1)$ is implied by Theorem 4.7.17, since right adjoints preserve products up to isomorphism. □

B.27 Proposition 4.7.20

Proof. It was shown in [84] that for an ES $\mathcal{E}$, $P_r(\mathcal{E})$ is an RES.

If $f : \mathcal{E}_0 \to \mathcal{E}_1$ is an ES morphism, then $f : P_r(\mathcal{E}_0) \to P_r(\mathcal{E}_1)$ satisfies the conditions of Definition 4.7.3:

1. When $Y = Y_1 = \emptyset$, this is analogous to condition 1 of ES morphisms.
2. This is condition 2 of ES morphisms.
3. This is condition 3 of ES morphisms.

Once established that P_r is a mapping, the rest of the proof is similar to previous functors. □

B.28 Proposition 4.7.22

Proof. We first show that given an RAES $\mathcal{E} = (E, F, \prec, \lhd)$, $P_{ar}(\mathcal{E})$ is an RES, satisfying the conditions of Definition 4.7.1:

1. Suppose $X \otimes Y \vdash e^*$. Then $(X \cup \{e\}) \cap Y = \emptyset$ because by definition of P_{ar}, $X \cap Y = \emptyset$ and by definition of an RAES, since $\lhd$ is irreflexive and $e \prec \underline{e}$ one cannot have $e \rhd e^*$.
2. Suppose $X \otimes Y \vdash \underline{e}$. Then $e \in X$ because $X = \{e' \in E \mid e' \prec \underline{e}\}$ and by definition $e \prec \underline{e}$.
3. Suppose $X \otimes Y \vdash e^*$, $X \subseteq X' \in \mathsf{Con}$, and $X' \cap Y = \emptyset$. Then it is clear from the definition of $\vdash$, that $X' \otimes Y \vdash e^*$.

We then show that given an RAES morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$, $f : P_{ar}(\mathcal{E}_0) \to P_{ar}(\mathcal{E}_1)$ is an RES morphism, satisfying the conditions of Definition 4.7.3:

1. Suppose $X \otimes Y \vdash_0 e$. Then $\{e' \in E_0 \mid e' \prec_0 e\} \subseteq X \in \mathsf{Con}_0$, $Y = \{e' \in E_0 \mid e' \rhd_0 e\}$, and $X \cap Y = \emptyset$. Since $X \in \mathsf{Con}_0$, we have $f(X) \in \mathsf{Con}_1$. By definition of P_{pr}, $\{e' \in E_0 \mid e' \prec_0 e\} \subseteq X$, and by definition of an RAES morphism, $\{e_1 \mid e_1 \prec_1 f(e)\} \subseteq \{f(e') \mid e' \prec_0 e\}$, implying $\{e_1 \mid e_1 \prec_1 f(e)\} \subseteq f(X)$. And since $f(e') \rhd_1 f(e)$ implies $e' \rhd_0 e$, we get that for all $e' \in E_0$ such that $f(e') \in \{e' \mid e' \rhd_1 f(e)\}$, $e' \in Y$. If $e_1 \in f(X) \cap \{e' \mid e' \rhd_1 \underline{f(e)}\}$, then there must exist an $e_0 \in X$ such that $f(e_0) = e_1$, and therefore $e_0 \rhd_0 f(e)$, implying $e_0 \in \overline{X \cap Y}$, which is not possible. Since $f(e) \in f(F_0) \subseteq F_1$, we can conclude that $f(X) \otimes \{e' \mid e' \rhd_1 f(e)\} \vdash_1 f(e)$.

   Suppose $X \otimes Y \vdash_0 \underline{e}$. Then the argument is similar.

2. Suppose $X_0 \in \mathsf{Con}_0$. Then $\lhd_0$ is well-founded on X_0, and for all $e, e' \in X_0$, if $f(e) \rhd_1 f(e')$, then $e \rhd_0 e'$. Hence $\lhd_1$ is also well-founded on $f(X_0)$, and therefore $f(X_0) \in \mathsf{Con}_1$.

3. Suppose $f(e) = f(e') \neq \perp$ and $e \neq e'$. Then $e \sharp_0 e'$ and $\mathsf{Con}_0 = \{Y \subseteq E_0 \mid \lhd_0 \text{ is well-founded on } Y\}$. Therefore $X \in \mathsf{Con}_0$ implies $\{e, e'\} \not\subseteq X$.

Having proved that P_{ar} is a mapping from **RAES** to **RES**, the rest of the proof is similar to previous functors. □

B.29 Theorem 4.7.23

Proof. Let $C_{ra}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, $P_{ar}(\mathcal{E}) = (E, F, \mathsf{Con}, \vdash)$, and $C_r(P_{ar}(\mathcal{E})) = (E, F, \mathsf{C}', \rightarrow')$.

We first prove that $\mathsf{C} = \mathsf{C}'$. Suppose $X \in \mathsf{C}$. Then $\lhd$ is well-founded on X, implying for all $X' \subseteq_{\text{fin}} X$, $\lhd$ is well-founded on X', and therefore $X' \in \mathsf{Con}$, implying $X \in \mathsf{C}'$.

Suppose $X' \in \mathsf{C}'$. This implies that for all $X \subseteq_{\text{fin}} X'$, $X \in \mathsf{Con}$ and therefore $\lhd$ is well-founded on X. Hence there cannot exist a finite $\lhd$-cycle in X', and therefore $\lhd$ is well-founded on X', implying $X' \in \mathsf{C}$.

It remains to show that $\rightarrow = \rightarrow'$. Suppose $X \xrightarrow{A \cup \underline{B}} Y$. Then $X, Y \in \mathsf{C}$, $A \subseteq E$, $B \subseteq F$, $Y = (X \setminus B) \cup A$, $A \cap X = \emptyset$, and $B \subseteq X$, so we need to prove that for all $a \in A$, $X' \otimes Z \vdash a$ for some $X' \subseteq X \setminus B$, Z, such that $Z \cap (A \cup X) = \emptyset$. Clearly this holds for $X' = \{e \mid e \prec a\}$ and $Z = \{e \mid e \rhd a\}$. Similarly for all $b \in B$, we have that $X' \otimes Z \vdash \underline{b}$ for some $X' \subseteq X \setminus (B \setminus \{b\})$, Z, such that $Z \cap (A \cup X) = \emptyset$.

Suppose $X \xrightarrow{A \cup \underline{B}}{}' Y$. Then $X, Y \in \mathsf{C}$, $A \subseteq E$, $B \subseteq F$, $Y = (X \setminus B) \cup A$, $A \cap X = \emptyset$, and $B \subseteq X$. Additionally for all $e \in A$, if $e' \prec e$ then whenever $X' \otimes Z \vdash e$, $e' \in X'$, and if $e' \rhd e$ then whenever $X' \otimes Z \vdash e$, $e' \in Z$. Also for every $e \in B$, if $e' \prec \underline{e}$ then whenever $X' \otimes Z \vdash \underline{e}$, we have $e' \in X'$, and if $e' \rhd \underline{e}$ then whenever $X' \otimes Z \vdash \underline{e}$, we have $e' \in Z$. □

B.30 Proposition 4.8.6

Proof. Let $\mathcal{E}_0 = (E_0, F_0, \mathsf{Con}_0, \vdash_0)$, $\mathcal{E}_1 = (E_1, F_1, \mathsf{Con}_1, \vdash_1)$, and $\mathcal{E}_0 \times \mathcal{E}_1 = (E, F, \mathsf{Con}, \vdash)$.

Then suppose $X \otimes Y \vdash e^*$, $X' \otimes Y' \vdash e^*$, and $X \cup X' + e^* \in \mathsf{Con}$. Then either $e^* = (e_0, *)^*$, $e^* = (*, e_1)^*$, or $e^* = (e_0, e_1)^*$

Suppose $e^* = (e_0, *)^*$. Then $\pi_0(X) \otimes Y_0 \vdash \pi_0(e)^*$, for some Y_0 such that $Y = \{e' \mid \pi_0(e') \in Y_0\}$, $\pi_0(X') \otimes Y_0' \vdash \pi_0(e)^*$ for some Y_0' such that $Y' = \{e' \mid \pi_0(e') \in Y_0'\}$, and $\pi_0(X \cup X' + e^*) \in \mathsf{Con}_0$. Hence $\pi_0(X) \cup \pi_0(X') + e_0^* \in \mathsf{Con}_0$, and therefore $\pi_0(X) \cap \pi_0(X') \otimes \pi_0(Y) \vdash e_0^*$. Since if $e^* = \underline{e}$ then $e^* \in X \cap X'$, we therefore have $X \cap X' \otimes Y \vdash e^*$.

Suppose $e^* = (*, e_1)^*$. Then similar logic can be used.

Suppose $e^* = (e_0, e_1)^*$. Then $\pi_1(X) \otimes Y_1 \vdash \pi_1(e)^*$, $\pi_1(X') \otimes Y_1' \vdash \pi_1(e)^*$, $\pi_0(X) \otimes Y_0 \vdash \pi_0(e)^*$, and $\pi_0(X') \otimes Y_0' \vdash \pi_0(e)^*$, for some Y_0, Y_0', Y_1, Y_1' such that $Y = \{e' \mid \pi_0(e') \in Y_0 \text{ or } \pi_1(e') \in Y_1\}$ and $Y' = \{e' \mid \pi_0(e') \in Y_0' \text{ or } \pi_1(e') \in Y_1'\}$, and $\pi_1(X \cup X' + e^*) \in \mathsf{Con}_1$ and $\pi_0(X \cup X' + e^*) \in \mathsf{Con}_0$, implying $\pi_0(X) \cup \pi_0(X') + e_0^* \in \mathsf{Con}_0$ and $\pi_1(X) \cup \pi_1(X') + e_1^* \in \mathsf{Con}_1$, and therefore $\pi_0(X) \cap \pi_0(X') \otimes Y_0 \vdash e_0^*$ and $\pi_1(X) \cap \pi_1(X') \otimes Y_1 \vdash e_1^*$, and, since if $e^* = \underline{e}$ then $e^* \in X \cap X'$, we get $X \cap X' \otimes Y \vdash e^*$. □

B.31 Proposition 4.8.9

Proof. We prove that $C_{fr}(\mathcal{E})$ fulfils the conditions of Definition 4.8.7.

1 If X is conflict-free then $X' \subseteq X$ is conflict-free.

2a $X_3 \cap A = \emptyset$, $B \subseteq X_1$, $X_3 \cup A$ is conflict-free, for all $e \in B$, if $e' \rhd \underline{e}$ then $e' \notin X_3 \cup A$, for all $e \in A$, if $e' \prec e$ and $e' \notin X_1 \setminus B$, then there exists $e'' \in X_1 \setminus B$ such that $e' \sharp e''$ and $e'' \prec e$, and for all $e \in B$, if $e' \prec \underline{e}$ and $e' \notin X_1 \setminus (B \setminus \{e\})$, then there exists $e'' \in X_1 \setminus (B \setminus \{e\})$ such that $e' \sharp e''$ and $e'' \prec \underline{e}$.

2b: $X_1 \cap A = X_2 \cap A = \emptyset$, $B \subseteq X_1 \cap X_2$, $X_1 \cup A$, $X_2 \cup A$, and $X_1 \cup X_2$ are conflict-free, for all $e \in B$, if $e' \rhd \underline{e}$ then $e' \notin X_1 \cup X_2 \cup A$, for all $e \in A$, if $e' \prec e$ and $e' \notin X_1 \setminus B$, then there exists $e'' \in X_1 \setminus B$ such that $e' \sharp e''$ and $e'' \prec e$ and if $e' \notin X_2 \setminus B$, then there exists $e'' \in X_2 \setminus B$ such that $e' \sharp e''$ and $e'' \prec e$, and for all $e \in B$, if $e' \prec \underline{e}$ and $e' \notin X_1 \setminus (B \setminus \{e\})$, then there exists $e'' \in X_1 \setminus (B \setminus \{e\})$ such that $e' \sharp e''$ and $e'' \prec \underline{e}$ and if $e' \prec \underline{e}$ and $e' \notin X_2 \setminus (B \setminus \{e\})$, then there exists $e'' \in X_2 \setminus (B \setminus \{e\})$ such that $e' \sharp e''$ and $e'' \prec \underline{e}$.

2c: $X_3 \cap A \subseteq (X_1 \cup X_2) \cap A = \emptyset$ and $B \subseteq X_1 \cap X_2 \subseteq X_3$, $X_3 \cup A \subseteq X_0 \cup X_1 \cup A$, which is conflict free, for all $e \in B$, if $e' \rhd \underline{e}$ then $e' \notin X_1 \cup X_2 \cup A$, for all $e \in A$, if $e' \prec e$ and $e' \notin (X_1 \cap X_2) \setminus B$, then there exists $e'' \in (X_1 \cap X_2) \setminus B$ such that $e' \sharp e''$ and $e'' \prec e$, and for all $e \in B$, if $e' \prec \underline{e}$ and $e' \notin (X_1 \cap X_2) \setminus (B \setminus \{e\})$, then there exists $e'' \in (X_1 \cap X_2) \setminus (B \setminus \{e\})$ such that $e' \sharp e''$ and $e'' \prec \underline{e}$.

2d: $X_3 = (((X_0 \setminus B_0) \cup A_0) \setminus B_1) \cup A_1 = (((X_0 \setminus B_1) \cup A_1) \setminus B_0) \cup A_0$, $X_0 \cap A_0 = X_0 \cap A_1 = \emptyset$, $B_0 \subseteq X_0$ and $B_1 \subseteq X_0$, $X_0 \cup A_0$ and $X_0 \cup A_1$ are conflict-free, for all $e \in B_0 \cup B_1$, if $e' \rhd \underline{e}$ then $e' \notin X_0 \cup A_0 \cap A_1$, for all $e \in A) \cup A_1$, if $e' \prec e$ and $e' \notin X_0 \setminus (B_0 \cup B_1)$, then there exists $e'' \in X_0 \setminus (B_0 \cup B_1)$ such that $e' \sharp e''$ and $e'' \prec e$, and for all $e \in B_1 \cup B_0$, if $e' \prec \underline{e}$ and $e' \notin X_0 \setminus (B_0 \cup B_1 \setminus \{e\})$, then there exists $e'' \in X_0 \setminus (B_0 \cup B_1 \setminus \{e\})$ such that $e' \sharp e''$ and $e'' \prec \underline{e}$.

□

B.32 Proposition 4.8.12

Proof. We first show that given an SFCS, $\mathcal{C} = (E, F, \mathsf{C}, \rightarrow)$, $\boldsymbol{R}'(\mathcal{C})$ is an RES, as described in Definition 4.7.1:

1. Suppose $X'' \otimes Y \vdash e^*$. Then clearly there exist X' and X such that $X' \subseteq X'' \subseteq X$, such that $Y = E \setminus X$, and therefore $X \cap Y = \emptyset$.

2. Suppose $X'' \otimes Y \vdash \underline{e}$. Then $e \in X''$, because there exists $X' \subseteq X''$ such that $X' \xrightarrow{\{e\}}$.

3. Suppose $X'' \otimes Y \vdash e^*$, $X'' \subseteq X'''' \in \mathsf{Con}$, and $X'''' \cap Y = \emptyset$. Then, since $X'' \subseteq X'''' \subseteq E \setminus Y$, clearly $X'''' \otimes Y \vdash e^*$.

We then show that $\boldsymbol{R}'(\mathcal{C})$ is stable:

Whenever $X_0 \otimes Y_0 \vdash e$, $X_1 \otimes Y_1 \vdash e$, and $X_0 \cup X_1 \cup \{e\} \in \mathsf{Con}$ we know that for all configurations $X_0', X_1' \in \mathsf{C}$, if $X_0 \subseteq X_0' \subseteq E \setminus (Y_2 \cup \{e\})$ and $X_1 \subseteq X_1' \subseteq E \setminus (Y_1 \cup \{e\})$ then $X_0' \xrightarrow{\{e\}}$ and $X_1' \xrightarrow{\{e\}}$. Hence $X_0 \cup X_1 \cup \{e\} \in \mathsf{C}$, implying $X_0 \cap X_1 \xrightarrow{\{e\}}$, and $X_0 \cap X_1 \subseteq X_0' \cap X_1' \subseteq X_1'$. Hence $X_0' \cap X_1' \xrightarrow{\{e\}}$, and therefore for all $X \in \mathsf{C}$, if $X_0 \cap X_1 \subseteq X \subseteq (E \setminus (Y_1 \cup \{e\})) \cup (E \setminus (Y_0 \cup \{e\}))$ then $X \xrightarrow{\{e\}}$. This gives us $X_0 \cap X_1 \otimes Y_0 \cap Y_1 \vdash e$.

If $X_0 \otimes Y_0 \vdash \underline{e}$, $X_1 \otimes Y_1 \vdash \underline{e}$, and $X_0 \cup X_1 \setminus \{e\} \in \mathsf{Con}$ the argument is similar.

We then show that given an SFCS morphism $f : \mathcal{C}_0 \to \mathcal{C}_1$, $f : \boldsymbol{R}'(\mathcal{C}_0) \to \boldsymbol{R}'(\mathcal{C}_1)$ is an RES morphism, satisfying the conditions of Definition 4.7.3:

1. Suppose $f(e) \neq \perp$ and $X'' \otimes Y \vdash_0 e^*$. Then there exists a minimum X' such that whenever $X' \subseteq X''' \subseteq E_0 \setminus Y$ for $X''' \in \mathsf{C}_0$, we have a transition $X''' \xrightarrow{\{e^*\}}$. Therefore $f(X''') \xrightarrow{\{f(e)^*\}}$. We also have $X' \cup \{e'\} \in \mathsf{C}_0$ for all $e' \in E_0 \setminus Y$, and C is downwards-closed, whenever $f(X') \subseteq X'''' \subseteq f(E_0 \setminus Y)$, $X'''' \xrightarrow{\{f(e)^*\}}$. This implies that there exists a Y_1 such that $f(X) \otimes Y_1 \vdash f(e)^*$, and if $f(e') \in Y_1$, then $e' \in Y$.

2. Suppose $X_0 \in \mathsf{Con}_0$. Then there exists $X_0' \in \mathsf{C}_0$ such that $X_0 \subseteq_{\text{fin}} X_0'$. Clearly $f(X_0') \in \mathsf{C}_1$ and $f(X_0) \subseteq_{\text{fin}} f(X_0')$, implying $f(X_0) \in \mathsf{Con}_1$.

3. Suppose $f(e) = f(e')$ and $e \neq e'$. Then there does not exist $X \in \mathsf{C}_0$ such that $e, e' \in C$. Since for all $X_0 \in \mathsf{Con}_0$, there exists $X_0' \in \mathsf{C}_0$ such that $X \subseteq_{\text{fin}} X_0'$, there also does not exist $X_0'' \in \mathsf{Con}_0$, such that $e, e' \in X_0''$.

□

B.33 Proposition 4.8.13

Proof. Let $\mathcal{E}$ be an RSES and recall from Proposition 4.7.12 that $\boldsymbol{C}_r(\mathcal{E})$ is an FCS. Then we need to show that $\boldsymbol{C}_r(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$ fulfils the conditions of Definition 4.8.7:

1. C is downwards-closed because $\boldsymbol{C}_r(\mathcal{E})$ is an FCS.
2. For $X_1, X_2, X_3 \in \mathsf{C}$:
   (a) This means that for $e^* \in A \cup \underline{B}$, there exist X_1', X_3', Y_1, Y_3 such that $X_1' \subseteq X_1 \setminus (B \setminus \{e\})$, $X_3' \subseteq X_3 \setminus (B \setminus \{e\})$, $X_1' \otimes Y_1 \vdash e^*$, $X_3' \otimes Y_3 \vdash e^*$, $Y_1 \cap (X_1 \cup A) = \emptyset$, and $Y_3 \cap (X_3 \cup A) = \emptyset$, and therefore, since $X_1' \cap X_3' \otimes Y_1 \cap Y_3 \vdash e^*$, we get $X_2 \xrightarrow{A \cup \underline{B}}$.
   (b) This means that for $e^* \in A \cup \underline{B}$, there exist X_1', X_2', Y_1, Y_2 such that $X_1' \subseteq X_1 \setminus (B \setminus \{e\})$, $X_2' \subseteq X_2 \setminus (B \setminus \{e\})$, $X_1' \otimes Y_1 \vdash e^*$, $X_2' \otimes Y_2 \vdash e^*$, $Y_1 \cap (X_1 \cup A) = \emptyset$, and $Y_2 \cap (X_2 \cup A) = \emptyset$, and therefore, since $X_1' \cap X_2' \otimes Y_1 \cap Y_2 \vdash e^*$ and C is downwards-closed, we get $X_1 \cap X_2 \xrightarrow{A \cup \underline{B}}$.
   (c) This means that for $e^* \in A_0 \cup \underline{B_0}$, there exist Y_0, $Y_1 \subseteq E \setminus X_1$, and $Y_2 \subseteq E \setminus X_2$, such that $X_1 \cap X_2 \otimes Y_0 \vdash e^*$, $X_1 \otimes Y_1 \vdash e^*$ and $X_2 \otimes Y_2 \vdash e^*$, and clearly $(X_1 \cap X_2) \cap X_1 \cap X_2 = X_1 \cap X_2$, and $Y_0 \cap Y_1 \cap Y_2 \subseteq E \setminus (X_1 \cup X_2)$, implying $X_1 \cap X_2 \otimes Y_0 \cap Y_1 \cap Y_2 \vdash e^*$. Hence $X_3 \xrightarrow{A \cup \underline{B}}$.
   (d) This means that for all $e^* \in A_0 \cup \underline{B_0}$, there exist X_0', X_2', Y_0, Y_2 such that $X_0' \subseteq X_0 \setminus (B_0 \setminus \{e\})$, $X_2' \subseteq X_2 \setminus (B_0 \setminus \{e\})$, $X_0' \otimes Y_0 \vdash e^*$, $X_2' \otimes Y_2 \vdash e^*$, $Y_0 \cap (X_0 \cup A_0) = \emptyset$, and $Y_2 \cap (X_2 \cup A_0) = \emptyset$. Hence $X_0' \cap X_2' \otimes Y_0 \cap Y_2 \vdash e^*$ and $X_0 \cap X_2 \subseteq X_0 \setminus B_1$ and $Y_0 \cap Y_2 \subseteq Y_2$, and similarly for $e'^* \in A_1 \cup \underline{B_1}$. Therefore $X_0 \xrightarrow{A_0 \cup A_1 \cup \underline{B_0 \cup B_1}}$.

□

B.34 Proposition 4.8.15

Proof.

1. If $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$, $\boldsymbol{C}_r(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, and $R'(\boldsymbol{C}_r(\mathcal{E})) = (E, F, \mathsf{Con}', \vdash')$, then we have $\mathsf{C} = \{C \mid X \subseteq_{\text{fin}} C \Rightarrow X \in \mathsf{Con}\}$, and $\mathsf{Con}' = \{X \mid \exists X' \in \mathsf{C}. X \subseteq_{\text{fin}} X'\}$, and $\boldsymbol{C}_r(\mathcal{E})$ is an FCS, and therefore $\mathsf{Con}' = \mathsf{Con}$.

   Suppose $X \otimes Y \vdash e$. Then for all $X' \in \mathsf{C}$, if $X' \cup \{e\} \in \mathsf{C}$ and $X \subseteq_{\text{fin}} X' \subseteq E \setminus (Y \cup \{e\})$, then $X' \xrightarrow{\{e\}}$, which implies that there must exist some minimum $X'' \subseteq X$ and maximum $X''' \supseteq E \setminus (Y \cup \{e\})$, such that if $X'' \subseteq X'''' \subseteq X'''$, and $X'''', X'''' \cup \{e\} \in \mathsf{C}$, then $X'''' \xrightarrow{\{e\}}$. As $X \otimes Y \vdash e$ is a necessary enabling $E \setminus (X''' \cup \{e\}) = Y$, giving us $X \otimes Y \vdash' e$. Similar arguments apply if $X \otimes Y \vdash \underline{e}$.

   Suppose $X \otimes Y \vdash' e$. Then there exists an $X' \subseteq X$ such that $X' \xrightarrow{\{e\}}$, and if $X'', X'' + e \in \mathsf{C}$ and $X' \subseteq X'' \subseteq E \setminus (Y \cup \{e\})$, then $X \xrightarrow{\{e\}}$. As $\mathcal{E}$ is stable, this requires $X' \otimes Y' \vdash e$ for some $Y' \supseteq Y$ such that if $X \cap Y' = \emptyset$ and $X \cap Y \neq \emptyset$ then $X \cup \{e\} \notin \mathsf{C}$. Since all enablings in $\mathcal{E}$ are minimal, it must be that $Y = Y'$, and therefore $X \otimes Y \vdash e$. Similar arguments apply if $X \otimes Y \vdash' \underline{e}$.

2. If $\mathcal{C} = (E, F, \mathsf{C}, \rightarrow)$, $R'(\mathcal{C}) = (E, F, \mathsf{Con}, \vdash)$, and $\boldsymbol{C}_r(R'(\mathcal{C})) = (E, F, \mathsf{C}', \rightarrow')$, then $\mathsf{C}' = \mathsf{C}$ for the same reasons as in Theorem 4.7.16.

   Suppose $X \xrightarrow{A \cup \underline{B}} Y$. Then for all $e^* \in A \cup \underline{B}$, whenever $X \subseteq X' \subseteq Y - e^*$, $X' \xrightarrow{\{e^*\}}$. Hence there exists $Y' \subseteq E \setminus (Y \cup B)$ such that $X \otimes Y' \vdash e^*$, and therefore $X \xrightarrow{A \cup \underline{B}}'$.

   Suppose $X \xrightarrow{A \cup \underline{B}}' Y$. Then for all $e^* \in A \cup \underline{B}$, there exists $X' \subseteq X$ and Y' such that $(X \cup A) \cap Y' = \emptyset$, and $X' \otimes Y' \vdash e^*$. This in turn requires $X'' \xrightarrow{\{e^*\}}$ whenever $X' \subseteq X'' \subseteq E \setminus (Y \cup \{e\})$. Since $\mathcal{C}$ is stable, this implies $X \xrightarrow{A \cup \underline{B}} Y$.

□

B.35 Proposition 4.8.20

Proof. We first show that given an RES $\mathcal{E}$ it is the case that $N(\mathcal{E}) = (E, F, \mathsf{Con}, \vdash_{\mathsf{nec}})$ is an RES. For every enabling $X \otimes Y \vdash_{\mathsf{nec}} e^*$, by definition $(X \cup \{e\}) \cap Y = \emptyset$, and if $e^* = \underline{e}$ then, since $X' \otimes Y' \vdash \underline{e}$ for some $X' \subseteq X$, $e \in X$. If $X \subseteq X'' \in \mathsf{Con}$ and $X'' \cap Y = \emptyset$ then clearly $X' \otimes Y' \vdash e^*$ can be used to generate the enabling $X'' \otimes Y \vdash_{\mathsf{nec}} e^*$ according to Definition 4.8.19.

Consider any enabling $X \otimes Y \vdash_{\mathsf{nec}} e^*$. We show that it is both necessary and minimal.

We first show that it is necessary. From Definition 4.8.19 we know that there exists $X_{\min} \subseteq X$ such that $X_{\min} \otimes Y \vdash_{\min} e^*$ and no $X' \subseteq X_{\min}$ and $Y' \subseteq Y$ exist such that $X' \otimes Y' \vdash_{\min} e^*$ and not both $X' = X_{\min}$ and $Y' = Y$. Clearly this implies $X_{\min} \otimes Y \vdash_{\mathsf{nec}} e^*$. If for all $X' \subseteq X$ such that $X' \otimes Y \vdash_{\mathsf{nec}} e^*$, there exists a $Y' \subsetneq Y$ such that $X' \otimes Y' \vdash_{\mathsf{nec}} e^*$, then we have a contradiction when $X' = X_{\min}$.

We then show that $X \otimes Y \vdash_{\mathsf{nec}} e^*$ is minimal. If $X' \otimes Y' \vdash e^*$ generated $X \otimes Y \vdash_{\min} e^*$, then $Y = Y' \cup \{e' \mid X' \cup \{e, e'\} \notin \mathsf{Con}\}$ and $X' \otimes Y \vdash_{\min} e^*$. Clearly $X_{\min} \subseteq X'$, and therefore whenever $X_{\min} \cup \{e, e'\} \notin \mathsf{Con}$, $e' \in Y$.

We then need to show that given RESs $\mathcal{E}_0$ and $\mathcal{E}_1$ and RES morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$, $f : N(\mathcal{E}_0) \to N(\mathcal{E}_1)$ is an RES morphism, satisfying the conditions of Definition 4.7.3:

1. Suppose $f(e) \neq \bot$ and $X \otimes Y \vdash_{0_{\min}} e^*$. Then there exist X' and Y' such that $X' \otimes Y' \vdash e^*$, $X' \setminus X$, no $X'' \subsetneq X'$ exists such that $X'' \otimes Y \vdash e^*$, and $e' \in Y$ iff $e \in Y'$ or $X' \cup \{e, e'\} \notin \mathsf{Con}$. Therefore $f(X') \otimes Y_1' \otimes \vdash_1 f(e)^*$ for some Y_1' such that if $f(e') \in Y_1$ then $e' \in Y$. We then find a minimum $X'' \subseteq f(X')$ such that $X'' \otimes Y_1' \vdash_1 f(e)^*$, and we need to show $f(X) \otimes Y_1' \cup \{e' \mid X'' \cup \{f(e), e'\} \notin \mathsf{Con}_1\} \vdash_{1_{\min}} f(e)^*$ and if $f(e') \in Y_1' \cup \{e' \mid X'' \cup \{f(e), e'\} \notin \mathsf{Con}_1\}$ then $e' \in Y$.

   We show the latter first. Suppose $f(e') \in Y_1'$. Then $e' \in Y' \subseteq Y$. Suppose $f(e') \in \{e' \mid X'' \cup \{f(e), e'\} \notin \mathsf{Con}_1\}$. Then since $X'' \subseteq f(X')$ we know there exists $X_0'' \subseteq X'$ such that $f(X_0'') = X''$ and $X'' \cup \{e, e'\} \notin \mathsf{Con}_0$. Since Con_0 is downwards-closed we then have $X' \cup \{e, e'\} \notin \mathsf{Con}_0$, and therefore $e' \in Y$.

   We then show $f(X) \otimes Y_1' \cup \{e' \mid X'' \cup \{f(e), e'\} \notin \mathsf{Con}_1\} \vdash_{1_{\min}} f(e)^*$. Clearly $f(X) \in \mathsf{Con}_1$, and $(f(X) \cup \{f(e)\}) \cap (Y_1' \cup \{e' \mid X'' \cup \{f(e), e'\}) = \emptyset$ because, as shown above, if $f(e') \in Y_1' \cup \{e' \mid X'' \cup \{f(e), e'\}$ then $e' \in Y$, and therefore $(X \cup \{e\}) \cap Y = \emptyset$. Hence $X'' \otimes Y_1' \vdash_1 f(e)^*$ can generate $f(X) \otimes Y_1' \cup \{e' \mid X'' \cup \{f(e), e'\} \notin \mathsf{Con}_1\} \vdash_{1_{\min}} f(e)^*$.

   Suppose $f(e) \neq \bot$ and $X \otimes Y \vdash_{0_{\mathsf{nec}}} e^*$. Then $X \otimes Y \vdash_{0_{\min}} e^*$ and there exists an $X_{\min} \subseteq X$ such that $X_{\min} \otimes Y \vdash_{0_{\min}} e^*$ and no $X' \subseteq X_{\min}$ and $Y' \subseteq Y$ exist such that $X' \otimes Y' \vdash_{0_{\min}} e^*$ and not both $X_{\min} = X'$ and $Y = Y'$. There therefore must exist a minimal Y_1 such that $f(X_{\min}) \otimes Y_1 \vdash_{1_{\min}} f(e)^*$ and if $f(e') \in Y_1$ then $e' \in Y$, and a minimal $X_{1_{\min}} \subseteq f(X_{\min})$ such that $X_{1_{\min}} \otimes Y_1 \vdash_{1_{\min}} f(e)^*$ and therefore $X_{1_{\min}} \otimes Y_1 \vdash_{1_{\mathsf{nec}}} f(e)^*$. Clearly $X_{1_{\min}} \subseteq f(X)$, $f(X) \in \mathsf{Con}_1$, and $f(X) \cap Y_1 = \emptyset$. Hence, since $N(\mathcal{E}_1)$ is an RES, $f(X) \otimes Y_1 \vdash_{1_{\mathsf{nec}}} f(e)^*$.

2. N does not change Con.

3. N does not change Con.

Once established that N is a mapping from **RES** to **nfRES**, the rest of the proof is similar to previous functors. □

B.36 Proposition 4.9.6

Proof.

1. Suppose $\mathcal{E} = (E, F, \mathsf{Con}, \vdash)$ and $\boldsymbol{C}_r(\mathcal{E}) = (E, F, \mathsf{C}, \to)$. Then whenever $X \in \mathsf{C}$ is a configuration, $A \subseteq E$, $B \subseteq F$, and $X \xrightarrow{A \cup \underline{B}} (X \cup A) \setminus B$, we have that (1) for all $a \in A$, there exist $X' \subseteq_{\mathrm{fin}} X \setminus B$ and

$Z \subseteq E \setminus (X \cup A)$ such that $X' \otimes Z \vdash a$ and (2) for all $b \in B$, there exist $X' \subseteq_{\text{fin}} X \setminus (B \setminus \{b\})$ and $Z \subseteq E \setminus (X \cup A)$ such that $X' \otimes Z \vdash \underline{b}$. Hence $X' \setminus \{b\} \otimes Z \vdash b$, and we therefore have $X \setminus B \xrightarrow{A \cup \underline{B}} X \cup A$.

2. Suppose $C = (E, F, \mathsf{C}, \rightarrow)$ and $R(C) = (E, F, \mathsf{Con}, \vdash)$. Then whenever $X \otimes Y \vdash \underline{e}$, by definition of R, there exists $X' \in \mathsf{C}$, $A \subseteq E$, and $B \subseteq F$ such that $X' \xrightarrow{A \cup \underline{B}}$, $e \in B$, $X' \setminus B \subseteq X$, and $Y = E \setminus (X' \cup A)$. We therefore have $X' \setminus B \xrightarrow{A \cup B} X' \cup A$, and consequently $X \setminus \{e\} \otimes Y \vdash e$.

3. Suppose $C = (E, F, \mathsf{C}, \rightarrow)$ and $R(C) = (E, F, \mathsf{Con}, \vdash)$. Then whenever $X \otimes Y \vdash \underline{e}$, by definition of R' we have for all $X' \in \mathsf{C}$, if $X \subseteq X' \subseteq (E \setminus Y)$, $X' \xrightarrow{\underline{e}} X' \setminus \{e\}$, implying $X' \setminus \{e\} \xrightarrow{\{\underline{e}\}} X'$. We therefore get $X \setminus \{e\} \otimes Y' \vdash e$ for some $Y' \subseteq Y$.

□

B.37 Proposition 4.9.7

Proof.

1. Suppose $X \in \mathsf{C}$ is a configuration, $A, B \subseteq F$, and $X \xrightarrow{A \cup \underline{B}} (X \setminus B) \cup A$. Then $(X \setminus B) \cup A \xrightarrow{B \cup \underline{A}} X$ according to the definition of C_r (Definition 4.7.5) because:

   (a) By definition $(X \setminus B) \cup A = (X \setminus B) \cup A$, $B \cap ((X \setminus B) \cup A) = \emptyset$, $A \subseteq ((X \setminus B) \cup A)$, and $((X \setminus B) \cup A) \cup B = X \cup A \in \mathsf{C}$.

   (b) For all $a \in A$, we have $X' \otimes Z \vdash e$ for some X', Z such that $X' \subseteq_{\text{fin}} X \setminus B$ and $Z \cap (X \cup A) = \emptyset$. Therefore $X' \cup \{e\} \otimes Z \vdash \underline{e}$, and consequently $X' \cup \{e\} \subseteq_{\text{fin}} ((X \setminus B) \cup A) \setminus (A \setminus e) = (X \cup \{e\}) \setminus B$ and $Z \cap (((X \setminus B) \cup A) \cup B) = \emptyset$.

   (c) For all $e \in B$, we have $X' \otimes Z \vdash \underline{e}$ for some X', Z such that $X' \subseteq_{\text{fin}} X \setminus (B \setminus \{e\})$ and $Z \cap (X \cup A) = \emptyset$. Therefore $X' \setminus \{e\} \otimes Z \vdash e$, and consequently $X' \setminus \{e\} \subseteq_{\text{fin}} ((X \setminus B) \cup A) \setminus A = X \setminus B$ and $Z \cap (((X \setminus B) \cup A) \cup B) = \emptyset$.

2. Suppose $X \otimes Y \vdash e$. Then there exists a transition $X' \xrightarrow{A \cup \underline{B}} E \setminus (Y \cup B)$, such that $X' \setminus B \subseteq X$ and $e \in A$. Hence $E \setminus (Y \cup B) \xrightarrow{B \cup \underline{A}} X'$. As $(E \setminus (Y \cup B)) \setminus A = (X' \cup A) \setminus B \subseteq X \setminus \{e\}$ and $Y = E \setminus ((E \setminus (Y \cup B)) \cup B)$, we get $X \cup \{e\} \otimes Y \vdash \underline{e}$.

   Similar arguments apply when $X \otimes Y \vdash \underline{e}$.

3. Suppose $X \otimes Y \vdash e$. Then Y is a maximum set of events such that for all $X' \in \mathsf{C}$ such that $X \subseteq X' \subseteq E \setminus Y$ and $X' \cup \{e\} \in \mathsf{C}$, there exists a transition $X' \xrightarrow{\{e\}}$. Therefore Y is a maximum set of events such that for all $X' \in \mathsf{C}$ such that $X \cup \{e\} \subseteq X' \subseteq E \setminus Y$ and $X' \setminus \{e\} \in \mathsf{C}$, there exists a transition $X' \xrightarrow{\{\underline{e}\}}$. Therefore we get $X \cup \{e\} \otimes Y \vdash \underline{e}$.

   Similar arguments apply when $X \otimes Y \vdash \underline{e}$.

□

B.38 Proposition 4.9.11

Proof. Given a forwards-reachable configuration, $X \in \mathsf{C}$, and $A, B \subseteq F$, if $X \xrightarrow{A \cup \underline{B}} (X \cup A) \setminus B$ then we must show that $(X \cup A) \setminus B \xrightarrow{B \cup \underline{A}} X$ according to Definition 4.6.2:

1. Follows from Definition 4.6.18.

2. Follows from Definition 4.6.18.

3. Follows from Definition 4.6.18.

4. Follows from Definition 4.6.18.

5. For all $e \in A$, if $e' \rhd \underline{e}$ then either $e \sharp e'$ or $e \prec e'$. If $e \sharp e'$, then, since $X \cup A$ is conflict-free, $e' \notin X \cup A$. If $e \prec e'$ and $e' \in X \cup A$, then, since $X \cup A$ is forwards-reachable and $e \notin X$, there exists $e'' \in X$ such that $e'' \sharp e$, meaning $X \overset{e}{\not\rightarrow}$, contradicting $X \xrightarrow{A \cup \underline{B}}$.

6. For all $e \in B$, if $e' \prec e$ and $e' \notin X \setminus B$, then since X is forwards-reachable and $B \subseteq X$, either $e' \in B$ or there exists $e'' \in X$ such that $e'' \sharp e'$ and $e'' \prec e$.

   If $e' \in B$, then we know $e \rhd \underline{e'}$, meaning $X \overset{\{\underline{e'}\}}{\not\longrightarrow}$, which contradicts $X \xrightarrow{A \cup \underline{B}}$.

   If $e'' \in X$ such that $e'' \sharp e'$ and $e'' \prec e$, then $e \rhd \underline{e''}$, and for the same reasons as above, $e'' \notin B$.

7. If $e' \prec \underline{e}$, then $e' = e$ or $e \sharp e'$.

□

Appendix C

Proofs from Chapter 5

C.1 Proposition 5.1.4

Proof. $\mathcal{E}$ is an LRBES, and i_0 and i_1 are morphisms, so we simply need to prove that if there exists an LRBES $\mathcal{E}_2 = (E_2, F_2, \mapsto_2, \sharp_2, \rhd_2, \lambda_2, \mathsf{Act}_2)$ and morphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}_2$ and $f_1 : \mathcal{E}_1 \to \mathcal{E}_2$, then there exists a unique LRBES-morphism $f : \mathcal{E} \to \mathcal{E}_2$ such that $f_0 = i_0 \circ f$ and $f_1 = i_1 \circ f$.

Since $E_0 + E_1$, i_0, and i_1 make up a coproduct in the category of sets and partial functions, f must be unique.

We define f as $f((j,e)) = f_j(e)$ and prove it to be a morphism. Since $(E, F, \mapsto, \sharp, \rhd) = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0) + (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1)$, we know $f : (E, F, \mapsto, \sharp, \rhd) \to (E_2, F_2, \mapsto_2, \sharp_2, \rhd_2)$ is an RBES-morphism, and by definition $\lambda((e,j)) = \lambda_j(e) = \lambda_2(f_j(e))$. □

C.2 Proposition 5.1.8

Proof. Clearly $\mathcal{E}$ is an LRBES, and for all $i \in \omega$, $\mathcal{E}_i \leq \mathcal{E}$.

We therefore know that $\mathcal{E}$ is an upper bound of the chain. We now show that $\mathcal{E}$ is the least upper bound of the chain. Given an upper bound $\mathcal{E}'$, $E \subseteq E'$, $F = E \cap F'$, and if $X \mapsto e^*$ then, for all $n \in \omega$, if $e \in E_n$ then $X \cap E_n \mapsto_n e^*$, meaning since $\mathcal{E}_n \leq \mathcal{E}'$ there exists $X'_n \subseteq E'$ such that $X'_n \mapsto e^*$ and $X'_n \cap E_n = X_n$, and since for all $n' \geq n$, $\mathcal{E}_{n'} \leq \mathcal{E}'$ and $e \in E_{n'}$, $X'_n \cap E_{n'} \mapsto_{n'} e^*$. This means clearly $X'_n \cap E = \bigcup_{n \in \omega} X_n = X$. And for $e \in E$, if $X' \mapsto' e'^*$, then for all $n \in \omega$ such that $e \in E_n$ $X' \cap E_n \mapsto_n e^*$, meaning $X' \cap E \mapsto e^*$. Similar arguments apply to $\sharp$, $\rhd$, λ, and Act.

Therefore, clearly $\mathcal{E} \leq \mathcal{E}'$, and $\mathcal{E}$ is the least upper bound of the chain. □

C.3 Proposition 5.1.14

Proof. We say that $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ and prove this by induction on P:

- Suppose $P = 0$. Then $\mathcal{E}$ is empty, and therefore causal.
- Suppose $P = P_0 + P_1$, $[\![P_i]\!] = \langle (E_i, F_i, \mapsto_i, \sharp_i, \rhd_i, \lambda_i, \mathsf{Act}_i), \mathsf{Init}_i, k_i \rangle$, $e \in E$ and $e' \in F$. Then if $e \rhd \underline{e'}$, then there exists $i \in \{0,1\}$ such that either $e \rhd_i e'$ or $e \in E_i$ and $e' \in F_{1-i}$. By induction, $e \rhd_i e'$ this means there exists an $X_i \subseteq E_i$ such that $X_i \mapsto_i e$ and $e' \in X_i$. As $X_i \mapsto_i e$, we get $X_i \mapsto e$. And $E_i \times E_{1-i} \subseteq \sharp$.

  If there exists an $X \subseteq E$ such that $X \mapsto e$ and $e' \in X$, then there exists an $i \in \{0,1\}$ such that $X \mapsto_i e$. That by induction we get $e \rhd_i \underline{e'}$, implying $e \rhd \underline{e'}$.

  $X \mapsto \underline{e'}$ if and only if there exists an $i \in \{0,1\}$ such that $X \mapsto_i \underline{e'}$. By induction, this means $e' \in X$.
- Suppose $P = \alpha.P'$, $[\![P']\!] = \langle (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}'), \mathsf{Init}', k' \rangle$, $e \in E$ and $e' \in F$. Then if $e \rhd \underline{e'}$ then either $e \rhd' \underline{e'}$, or $e' = e_\alpha$ and $e \in E'$. If $e \rhd' \underline{e'}$, then by induction there exists an $X \subseteq E'$ such that $X \mapsto' e$ and $e' \in X$, and $X \mapsto e$. If $e' = e_\alpha$ and $e \in E'$ then we know $\{e_\alpha\} \mapsto e$.

If there exists an $X \subseteq E$ such that $X \mapsto e$ and $e' \in X$, then either $X \mapsto' e$, or $X = \{e_\alpha\}$ and $e \in E'$. If $X \mapsto' e$, then by induction we get $e \rhd' \underline{e'}$, and therefore $e \rhd \underline{e'}$. If $X = \{e_\alpha\}$ and $e \in E'$, then we know $e \rhd \underline{e_\alpha}$.

$X \mapsto \underline{e'}$ if and only if $X \mapsto' \underline{e'}$ or $e' = e_\alpha$ and $X = \{e_\alpha\}$. By induction, if $X \mapsto' \underline{e'}$ then $e' \in X$.

- Suppose $P = \alpha[m].P'$. Then the proof is similar to the previous case.
- Suppose $P = P_0 \mid P_1$, $[\![P_i]\!] = \langle (E_i, F_i, \mapsto_i, \sharp_i, \rhd_i, \lambda_i, \mathsf{Act}_i), \mathsf{Init}_i, k_i \rangle$, $e \in E$ and $e' \in F$. Then if $e \rhd \underline{e'}$, then there exists an $i \in \{0,1\}$, such that $\pi_i(e) \rhd_i \pi_i(e')$. By induction, this means there exists an $X_i \subseteq E_i$ such that $X_i \mapsto_i \pi_i(e)$ and $\pi_i(e') \in X_i$. This means $\{e'' \in E \mid \pi_i(e'') \in X_i\} \mapsto e$, and $e' \in \{e'' \in E \mid \pi_i(e'') \in X_i\}$.

  If there exists an $X \subseteq E$ such that $X \mapsto e$ and $e' \in X$, then there exists an $i \in \{0,1\}$ and $X_i \subseteq E_i$ such that $X_i \mapsto_i \pi_i(e)$ and $X = \{e'' \in E \mid \pi_i(e'') \in X_i\}$, meaning $\pi_i(e') \in X_i$. By induction we get $\pi_i(e) \rhd_i \underline{\pi_i(e')}$, and therefore $e \rhd \underline{e'}$.

  $X \mapsto \underline{e'}$ if and only if there exists $i \in \{0,1\}$ and $X_i \subseteq E_i$ such that $X_i \mapsto_i \pi_i(e')$ and $X = \{e'' \in E \mid \pi_i(e'') \in X_i\}$. By induction, since $X_i \mapsto_i \pi_i(e')$ we know $\pi_i(e') \in X_i$, meaning clearly $e' \in X$.
- Suppose $P = P' \setminus A$, $[\![P']\!] = \langle (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}'), \mathsf{Init}', k' \rangle$, $e \in E$ and $e' \in F$. Then $X \mapsto e'^*$ if and only if $X \mapsto' e'^*$ and $e \rhd \underline{e'}$ if and only if $e \rhd' \underline{e'}$. The rest of the case follows from induction.
- Suppose $P = P'[f]$. Then the result follows from induction. □

C.4 Proposition 5.1.15

Proof. We say that $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$ and do a case analysis on the structural congruence rules:

$P = X \mid Y$ and $P' = Y \mid X$: Products are unique up to isomorphism and

$$f(e) = \begin{cases} (e_Y, e_X) & \text{if } e = (e_X, e_Y) \\ (e_Y, *) & \text{if } e = (*, e_Y) \\ (*, e_X) & \text{if } e = (e_X, *) \end{cases} \quad \text{clearly fulfils the conditions other conditions.}$$

$P = X \mid (Y \mid Z)$ and $P' = (X \mid Y) \mid Z$: Products are associative up to isomorphism, and $f((e_X, (e_Y, e_Z)) = ((e_X, e_Y), e_Z)$ clearly fulfills the other conditions.

$P = P' \mid 0$: If $f((e, *)) = e$, then this clearly holds.

$P = X + Y$ and $P' = Y + X$: Coproducts are unique up to isomorphism, and $f(e) = e$ clearly fulfil the other conditions.

$P = (X + Y) + Z$ and $P' = (X + Y) + Z$: Coproducts are associative up to isomorphism, and $f(e) = e$ clearly fulfils the other conditions.

$P = P' + 0$: Clearly $\mathcal{E}_P = (E' \cup \emptyset, F' \cup \emptyset, \mapsto' \cup \emptyset, \sharp' \cup \emptyset, \rhd' \cup \emptyset, \lambda', \mathsf{Act}' \cup \emptyset)$, $\mathsf{Init} = \mathsf{Init}'$, and $k = k'$.

$P = A\langle \tilde{a} \rangle$ and $P' = P_A\{^{\tilde{a}}/_{\tilde{b}}\}$ where $A\langle \tilde{b} \rangle = P_A$: Follows from the condition that all free and bound names are distinct. □

C.5 Lemma 5.1.17

Proof. We show this by induction on P.

- Suppose $P = 0$. Then $\mathsf{Init} = \emptyset$.
- Suppose $P = \alpha.P'$ and $[\![P']\!] = \langle \mathcal{E}', \mathsf{Init}', k' \rangle$. Then $\mathsf{Init} = \mathsf{Init}'$, and therefore Init is conflict-free.

- Suppose $P = \alpha[m].P'$ and $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k' \rangle$. Then $\mathsf{Init} = \mathsf{Init}' \cup \{e_\alpha\}$, Init' is conflict-free, and therefore Init is clearly conflict-free.

- Suppose $P = P_1 + P_2$ and $\{\!|P_i|\!\} = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i \rangle$. Then $\mathsf{Init} = \mathsf{Init}_1 \cup \mathsf{Init}_2$ and, since P is reachable from a standard process, either $Init_1 = \emptyset$ or $\mathsf{Init}_2 = \emptyset$, and both Init_1 and Init_2 are conflict-free. Therefore, Init is conflict-free.

- Suppose $P = P_1 \mid P_2$ and $\{\!|P_i|\!\} = \langle (E_i, F_i, \mapsto_i, \sharp_i, \rhd_i, \lambda_i, \mathsf{Act}_i), \mathsf{Init}_i, k_i \rangle$. Then, since P is reachable from a standard process, each key appears at most once in P_1 and once in P_2. Additionally, Init_1 is conflict-free and Init_2 is conflict-free, meaning $\mathsf{Init} = \{(e_0, *) \mid e_0 \in \mathsf{Init}_0 \text{ and } \nexists e_1 \in \mathsf{Init}_1 . \lambda_0(e_0) = \overline{\lambda_1(e_1)} \text{ and } k_0(e_0) = k_1(e_1)\} \cup \{(*, e_1) \mid e_1 \in \mathsf{Init}_1 \text{ and } \nexists e_0 \in \mathsf{Init}_0 . \lambda_0(e_0) = \overline{\lambda_1(e_1)} \text{ and } k_0(e_0) = k_1(e_1)\} \cup \{(e_0, e_1) \mid e_0 \in \mathsf{Init}_0, e_1 \in \mathsf{Init}_1, k_0(e_0) = k_1(e_1)\}$ is conflict-free.

- Suppose $P = P' \setminus A$ and $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}', k' \rangle$. Then Init' is conflict-free, and $\mathsf{Init} \subseteq \mathsf{Init}'$, meaning Init is conflict-free.

- Suppose $P = P'[f]$. Then $\mathsf{Init} = \mathsf{Init}'$, which is conflict-free. □

C.6 Theorem 5.1.18

Proof. We say that $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$, and prove the theorem by inductions on $P \xrightarrow{\mu[m]} P'$:

- Suppose $P = \alpha.Q$, $P' = \alpha[m].Q$, $\mu = \alpha$, and $\mathsf{std}(Q)$. Then there exist $\mathcal{E}_Q$ and e_α such that:
  - $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}, k \rangle$,
  - $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$,
  - $e_\alpha \notin E_Q$,
  - $E = E_Q \cup \{e_\alpha\}$,
  - $F = F_Q \cup \{e_\alpha\}$,
  - $X \mapsto e^*$ if $X \mapsto_Q e^*$ or $X = \{e_\alpha\}$ and $e^* = e \in E_Q$,
  - $\sharp = \sharp_Q$,
  - $\rhd = \rhd_Q \cup (E_Q \times \{\underline{e_\alpha}\})$,
  - $\mathsf{Act} = \mathsf{Act}_Q \cup \{\alpha\}$,
  - for all $e \in E$, $\lambda(e) = \begin{cases} \lambda_Q(e) & \text{if } e \in E_Q \\ \alpha & \text{if } e = e_\alpha \end{cases}$,
  - and if $\mathsf{Init} \neq \emptyset$ then for all $e \in E$, $\{e\} \mapsto e$ and for all $e \in F$, $e \rhd \underline{e}$.

  There also exist $\mathcal{E}'_Q$ and e'_α such that:
  - $\{\!|Q|\!\} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$, $E' = E'_Q \cup \{e'_\alpha\}$,
  - $\mathcal{E}'_Q = (E'_Q, F'_Q, \mapsto'_Q, \sharp'_Q, \rhd'_Q, \lambda'_Q, \mathsf{Act}'_Q)$,
  - $e'_\alpha \notin E'_Q$,
  - $F' = F'_Q \cup \{e'_\alpha\}$,
  - $X \mapsto' e^*$ if $X \mapsto'_Q e^*$ or $X = \{e'_\alpha\}$ and $e^* = e \in E'_Q$,
  - $\sharp' = \sharp'_Q$,
  - $\rhd' = \rhd'_Q \cup (E'_Q \times \{\underline{e'_\alpha}\})$,
  - $\mathsf{Act}' = \mathsf{Act}'_Q \cup \{\alpha\}$,

– for all $e \in E'$, $\lambda'(e) = \begin{cases} \lambda'_Q(e) & \text{if } e \in E'_Q \\ \alpha & \text{if } e = e'_\alpha \end{cases}$,

– $\mathsf{Init}' = \mathsf{Init}'_Q \cup \{e'_\alpha\}$,

– and $k'(e) = \begin{cases} k'_Q(e) & \text{If } e \in \mathsf{Init}'_Q \\ m & \text{If } e = e'_\alpha \end{cases}$.

As $\mathcal{E}_Q$ and $\mathcal{E}'_Q$ have been generated by the same process, we have isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and $g_Q : \mathcal{E}'_Q \to \mathcal{E}_Q$. We say that $f = f_Q[e_\alpha \mapsto e'_\alpha]$ and $g = g_Q[e'_\alpha \mapsto e_\alpha]$. It is easy to check that these are isomorphisms.

Since Init is conflict-free and $\sharp = \sharp_Q$, $X = \mathsf{Init} \cup \{e_\alpha\}$ is conflict free, and therefore a configuration of $C_{br}(\mathcal{E})$. And since no $X' \subseteq E$ exists such that $X' \mapsto e_\alpha$, we get $\mathsf{Init} \xrightarrow{\{e_\alpha\}} \{e_\alpha\}$, and clearly $\lambda(e_\alpha) = \alpha$ and $k(f(e)) = m$.

- Suppose that $P = \alpha[n].Q$, $P' = \alpha[n].Q'$, $Q \xrightarrow{\mu[m]} Q'$, and $m \neq n$. Then there exist $\mathcal{E}_Q$ and e_α such that:

  – $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$,

  – $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$

  – $e_\alpha \notin E_Q$,

  – $E = E_Q \cup \{e_\alpha\}$,

  – $F = F_Q \cup \{e_\alpha\}$,

  – $X \mapsto e^*$ if $X \mapsto_Q e^*$ or $X = \{e_\alpha\}$ and $e^* = e \in E_Q$,

  – $\sharp = \sharp_Q$,

  – $\rhd = \rhd_Q \cup (E_Q \times \{\underline{e_\alpha}\})$,

  – $\mathsf{Act} = \mathsf{Act}_Q \cup \{\alpha\}$,

  – for all $e \in E$, $\lambda(e) = \begin{cases} \lambda_Q(e) & \text{if } e \in E_Q \\ \alpha & \text{if } e = e_\alpha \end{cases}$,

  – $\mathsf{Init} = \mathsf{Init}_Q \cup \{e_\alpha\}$,

  – and $k(e) = \begin{cases} k_Q(e) & \text{If } e \in \mathsf{Init}_Q \\ m & \text{If } e = e_\alpha \end{cases}$.

  And there exist $\mathcal{E}'_Q$ and e'_α such that:

  – $\mathcal{E}'_Q = (E'_Q, F'_Q, \mapsto'_Q, \sharp'_Q, \rhd'_Q, \lambda'_Q, \mathsf{Act}'_Q)$,

  – $e'_\alpha \notin E'_Q$,

  – $\{\!|Q'|\!\} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$,

  – $E' = E'_Q \cup \{e'_\alpha\}$,

  – $F' = F'_Q \cup \{e'_\alpha\}$,

  – $X \mapsto' e^*$ if $X \mapsto'_Q e^*$ or $X = \{e'_\alpha\}$ and $e^* = e \in E'_Q$,

  – $\sharp' = \sharp'_Q$,

  – $\rhd' = \rhd'_Q \cup (E'_Q \times \{\underline{e'_\alpha}\})$,

  – $\mathsf{Act}' = \mathsf{Act}'_Q \cup \{\alpha\}$,

  – for all $e \in E'$, $\lambda'(e) = \begin{cases} \lambda'_Q(e) & \text{if } e \in E_{Q'} \\ \alpha & \text{if } e = e'_\alpha \end{cases}$,

- $\mathsf{Init}' = \mathsf{Init}'_Q \cup \{e'_\alpha\}$,
- and $k'(e) = \begin{cases} k'_Q(e) & \text{If } e \in \mathsf{Init}'_Q \\ m & \text{If } e = e'_\alpha \end{cases}$.

By induction, we get isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and $g_Q : \mathcal{E}'_Q \to \mathcal{E}_Q$ and a transition $\mathsf{Init}_Q \xrightarrow{\{e\}} X_Q$ in $C_{br}(\mathcal{E}_Q)$ such that $\lambda_Q(e) = \mu$, $k'_Q(f_Q(e)) = m$, and $f_Q(X_Q) = \mathsf{Init}'_Q$.

We define $f = f_Q[e_\alpha \mapsto e'_\alpha]$ and $g = g_Q[e'_\alpha \mapsto e_\alpha]$. Since Init_Q and X_Q are conflict-free in $\mathcal{E}_Q$, $\mathsf{Init}_Q \cup \{e_\alpha\} = \mathsf{Init}$ and $X_Q \cup \{e_\alpha\} = X$ are configurations of $C_{rb}(\mathcal{E})$, and clearly $\mathsf{Init} \xrightarrow{\{e\}} X$.

- Suppose $P = Q \mid R$, $P' = Q' \mid R$, $Q \xrightarrow{\mu[m]} Q'$, and $\mathsf{fsh}[m](R)$. Then there exist $\mathcal{E}_Q$ and $\mathcal{E}_R$ such that
  - $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$,
  - $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$,
  - $\{\!|R|\!\} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$,
  - $\mathcal{E}_R = (E_R, F_R, \mapsto_R, \sharp_R, \rhd_R, \lambda_R, \mathsf{Act}_R)$,
  - $E = \{(e, *) \mid e \in E_Q\} \cup \{(*, e) \mid e \in E_R\} \cup \{(e, e') \mid e \in E_Q, e' \in E_R, \lambda_Q(e) = \overline{\lambda_R(e')}\}$,
  - $F = \{(e, *) \mid e \in F_Q\} \cup \{(*, e) \mid e \in F_R\} \cup \{(e, e') \mid e \in F_Q, e' \in F_R, \lambda_0(e) = \overline{\lambda_1(e')}\}$,
  - there exist π_Q and π_R such that $\pi_i(e_Q, e_R) = e_i$ for $i \in \{Q, R\}$,
  - $X \mapsto e^*$ if there exists $i \in \{Q, R\}$ and $X_i \subseteq E_i$ such that $X_i \mapsto_i \pi_i(e)$ and $X = \{e' \in E \mid \pi_i(e') \in X_i\}$,
  - $e \sharp e'$ if there exists $i \in \{Q, R\}$ such that $\pi_i(e) \sharp_i \pi_i(e')$ or $\pi_i(e) = \pi_i(e') \neq \bot$,
  - $e \rhd \underline{e'}$ if there exists $i \in \{Q, R\}$ such that $\pi_i(e) \rhd_i \underline{\pi_i(e')}$,
  - $\mathsf{Act} = \mathsf{Act}_Q \cup \mathsf{Act}_R \cup \{\tau\}$,
  - $\lambda(e) = \begin{cases} \tau & \text{if } e = (e_Q, e_R) \\ \lambda_Q(e_Q) & \text{if } e = (e_Q, *) \\ \lambda_1(e_R) & \text{if } e = (*, e_R) \end{cases}$,
  - $\mathsf{Init} = \{(e_Q, *) \mid e_Q \in \mathsf{Init}_Q \text{ and } \nexists e_R \in \mathsf{Init}_R . \lambda_Q(e_Q) = \overline{\lambda_R(e_R)} \text{ and } k_Q(e_Q) = k_R(e_R)\} \cup \{(*, e_R) \mid e_R \in \mathsf{Init}_R \text{ and } \nexists e_Q \in \mathsf{Init}_Q . \lambda_Q(e_Q) = \overline{\lambda_R(e_R)} \text{ and } k_Q(e_Q) = k_R(e_R)\} \cup \{(e_Q, e_R) \mid e_Q \in \mathsf{Init}_Q, e_R \in \mathsf{Init}_R, k_Q(e_Q) = k_R(e_R)\}$,
  - and $k(e) = \begin{cases} k_Q(e_Q) & \text{If } e = (e_Q, *) \\ k_R(e_R) & \text{If } e = (*, e_R) \\ k_Q(e_Q) & \text{If } e = (e_Q, e_R) \end{cases}$.

We also have $\langle \mathcal{E}', \mathsf{Init}', k' \rangle$ similarly made up of some $\langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and $\langle \mathcal{E}'_R, \mathsf{Init}'_R, k'_R \rangle$ such that $\{\!|Q'|\!\} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and $\{\!|R|\!\} = \langle \mathcal{E}'_R, \mathsf{Init}'_R, k'_R \rangle$.

We clearly have isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}_{Q'}$, $g_Q : \mathcal{E}'_Q \to \mathcal{E}_Q$, $f_R : \mathcal{E}_R \to \mathcal{E}'_R$, and $g_R : \mathcal{E}'_R \to \mathcal{E}_R$ and a transition $\mathsf{Init}_Q \xrightarrow{\{e_Q\}} X_Q$ of $C_{rb}(\mathcal{E}_Q)$ such that $\lambda_Q(e) = \mu$, $k'_Q(f_Q(e_Q)) = m$, and $f_Q(X_Q) = \mathsf{Init}'_Q$.

Since Init is conflict-free and X_Q is conflict-free in $\mathcal{E}_Q$, clearly $\mathsf{Init} \cup \{(e_Q, *)\} = X$ is conflict-free, and $\mathsf{Init} \xrightarrow{(e_Q, *)} X$.

We define our isomorphisms as

$$f(e) = \begin{cases} (f_Q(e'), *) & \text{if } e = (e', *) \\ (*, f_R(e')) & \text{if } e = (*, e') \\ (f_Q(e'), f_R(e'')) & \text{if } e = (e', e'') \end{cases}$$

and

$$g(e) = \begin{cases} (g_Q(e'), *) & \text{if } e = (e', *) \\ (*, g_R(e')) & \text{if } e = (*, e') \\ (g_Q(e'), g_R(e'')) & \text{if } e = (e', e'') \end{cases}$$

And, since $\mathsf{fsh}[m](R)$, $f(X) = \mathsf{Init}'$. And the rest of the proof is straightforward.

- Suppose $P = Q \mid R$, $P' = Q' \mid R'$, $Q \xrightarrow{\alpha[m]} Q'$, $R \xrightarrow{\overline{\alpha}[m]} R'$, and $\mu = \tau$. Then there exist $\mathcal{E}_Q$ and $\mathcal{E}_R$ such that
  - $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$,
  - $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$,
  - $\{\!|R|\!\} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$,
  - $\mathcal{E}_R = (E_R, F_R, \mapsto_R, \sharp_R, \rhd_R, \lambda_R, \mathsf{Act}_R)$,
  - $E = \{(e, *) \mid e \in E_Q\} \cup \{(*, e) \mid e \in E_R\} \cup \{(e, e') \mid e \in E_Q, e' \in E_R, \lambda_Q(e) = \overline{\lambda_R(e')}\}$,
  - $F = \{(e, *) \mid e \in F_Q\} \cup \{(*, e) \mid e \in F_R\} \cup \{(e, e') \mid e \in F_Q, e' \in F_R, \lambda_0(e) = \overline{\lambda_1(e')}\}$,
  - there exist π_Q and π_R such that $\pi_i(e_Q, e_R) = e_i$ for $i \in \{Q, R\}$,
  - $X \mapsto e^*$ if there exists $i \in \{Q, R\}$ and $X_i \subseteq E_i$ such that $X_i \mapsto_i \pi_i(e)$ and $X = \{e' \in E \mid \pi_i(e') \in X_i\}$,
  - $e \sharp e'$ if there exists $i \in \{Q, R\}$ such that $\pi_i(e) \sharp_i \pi_i(e')$ or $\pi_i(e) = \pi_i(e') \neq \bot$,
  - $e \rhd \underline{e'}$ if there exists $i \in \{Q, R\}$ such that $\pi_i(e) \rhd_i \underline{\pi_i(e')}$,
  - $\mathsf{Act} = \mathsf{Act}_Q \cup \mathsf{Act}_R \cup \{\tau\}$,
  - $\lambda(e) = \begin{cases} \tau & \text{if } e = (e_Q, e_R) \\ \lambda_Q(e_Q) & \text{if } e = (e_Q, *) \\ \lambda_1(e_R) & \text{if } e = (*, e_R) \end{cases}$,
  - $\mathsf{Init} = \{(e_Q, *) \mid e_Q \in \mathsf{Init}_Q \text{ and } \nexists e_R \in \mathsf{Init}_R . \lambda_Q(e_Q) = \overline{\lambda_R(e_R)} \text{ and } k_Q(e_Q) = k_R(e_R)\} \cup \{(*, e_R) \mid e_R \in \mathsf{Init}_R \text{ and } \nexists e_Q \in \mathsf{Init}_Q . \lambda_Q(e_Q) = \overline{\lambda_R(e_R)} \text{ and } k_Q(e_Q) = k_R(e_R)\} \cup \{(e_Q, e_R) \mid e_Q \in \mathsf{Init}_Q, e_R \in \mathsf{Init}_R, k_Q(e_Q) = k_R(e_R)\}$,
  - and $k(e) = \begin{cases} k_Q(e_Q) & \text{If } e = (e_Q, *) \\ k_R(e_R) & \text{If } e = (*, e_R) \\ k_Q(e_Q) & \text{If } e = (e_Q, e_R) \end{cases}$.

  We also have $\langle \mathcal{E}', \mathsf{Init}', k' \rangle$ similarly made up of some $\langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and $\langle \mathcal{E}_{R'}, \mathsf{Init}_{R'}, k_{R'} \rangle$ such that $\{\!|Q'|\!\} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and $\{\!|R'|\!\} = \langle \mathcal{E}_{R'}, \mathsf{Init}_{R'}, k_{R'} \rangle$.

  By induction, we have isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$, $g_Q : \mathcal{E}'_Q \to \mathcal{E}_Q$, $f_R : \mathcal{E}_R \to \mathcal{E}_{R'}$, and $g_R : \mathcal{E}_{R'} \to \mathcal{E}_R$ and transitions $\mathsf{Init}_Q \xrightarrow{\{e_Q\}} X_Q$ of $C_{rb}(\mathcal{E}_Q)$ such that $\lambda_Q(e) = \alpha$, $k'_Q(f_Q(e_Q)) = m$, and $f_Q(X_Q) = \mathsf{Init}'_Q$, and $\mathsf{Init}_R \xrightarrow{\{e_R\}} X_R$ of $C_{rb}(\mathcal{E}_R)$ such that $\lambda_R(e) = \overline{\alpha}$, $k_{R'}(f_R(e_R)) = m$, and $f_R(X_R) = \mathsf{Init}_{R'}$.

  We define our isomorphisms as

$$f(e) = \begin{cases} (f_Q(e'), *) & \text{if } e = (e', *) \\ (*, f_R(e')) & \text{if } e = (*, e') \\ (f_Q(e'), f_R(e'')) & \text{if } e = (e', e'') \end{cases}$$

  and

$$g(e) = \begin{cases} (g_Q(e'), *) & \text{if } e = (e', *) \\ (*, g_R(e')) & \text{if } e = (*, e') \\ (g_Q(e'), g_R(e'')) & \text{if } e = (e', e'') \end{cases}$$

We know that Init, X_Q, and X_R are conflict-free, so the only way $\mathsf{Init} \cup \{(e_Q, e_R)\}$ has conflict is if Init_Q or Init_R an event with the key m, which we know from Lemma 5.2 of [80] is not possible. The rest of the proof is straightforward.

- Suppose $P = Q + R$, $P' = Q' + R$, $Q \xrightarrow{\mu[m]} Q'$, and $\mathsf{std}(R)$. Then there exist $\mathcal{E}_Q$ and $\mathcal{E}_R$ such that:
  - $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$,
  - $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$,
  - $\{\!|R|\!\} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$,
  - $\mathcal{E}_R = (E_R, F_R, \mapsto_R, \sharp_R, \rhd_R, \lambda_R, \mathsf{Act}_R)$,
  - $E = E_Q \cup E_R$,
  - $F = F_Q \cup F_R$,
  - $X \mapsto e^*$ if there exists $i \in \{Q, R\}$ such that $X \mapsto_i e^*$,
  - $\sharp = \sharp_Q \cup \sharp_R \cup (E_Q \times E_R) \cup (E_R \times E_Q)$,
  - $\rhd = \rhd_Q \cup \rhd_R \cup (E_Q \times \underline{F_R}) \cup (E_R \times \underline{F_Q})$,
  - $\mathsf{Act} = \mathsf{Act}_Q \cup \mathsf{Act}_R$,
  - for all $e \in E$, $i \in \{Q, R\}$, $\lambda(e) = \lambda_i(e)$ if $e \in E_i$,
  - $\mathsf{Init} = \mathsf{Init}_Q \cup \mathsf{Init}_R$,
  - for $i \in \{Q, R\}$, $k(e) = k_i(e)$ if $e \in \mathsf{Init}_i$,
  - if $\mathsf{Init}_Q \neq \emptyset$ and $\mathsf{Init}_Q \neq \emptyset$ then for all $e \in E$, $\{e\} \mapsto e$ and for all $e \in F$, $e \rhd \underline{e}$.

  We also have $\langle \mathcal{E}', \mathsf{Init}', k' \rangle$ similarly made up of some $\langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and $\langle \mathcal{E}'_R, \mathsf{Init}'_R, k'_R \rangle$ such that $\{\!|Q'|\!\} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and $\{\!|R|\!\} = \langle \mathcal{E}'_R, \mathsf{Init}'_R, k'_R \rangle$.

  We clearly have isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$, $g_Q : \mathcal{E}'_Q \to \mathcal{E}_Q$, $f_R : \mathcal{E}_R \to \mathcal{E}'_R$, and $g_R : \mathcal{E}'_R \to \mathcal{E}_R$ and a transition $\mathsf{Init}_Q \xrightarrow{\{e\}} X$ of $C_{rb}(\mathcal{E}_Q)$ such that $\lambda_Q(e) = \mu$, $k'_Q(f_Q(e)) = m$, and $f(X) = \mathsf{Init}'_Q$. We define our isomorphisms

$$f(e) = \begin{cases} f_Q(e) & \text{if } e \in E_Q \\ f_R(e) & \text{if } e \in E_R \end{cases}$$

  and

$$g(e) = \begin{cases} g_Q(e) & \text{if } e \in E'_Q \\ g_R(e) & \text{if } e \in E'_R \end{cases}$$

  Since $\mathsf{std}(R)$, $\mathsf{Init}_R = \emptyset$, and therefore $\mathsf{Init} = \mathsf{Init}_Q$, which is conflict-free end therefore a configuration. We then get $\mathsf{Init} \xrightarrow{\{e\}} X$ in $C_{rb}(\mathcal{E})$, and the rest follows.

- Suppose $P = Q \setminus A$, $P' = Q' \setminus A$, $Q \xrightarrow{\mu[m]} Q'$, and $\mu \notin A \cup \overline{A}$. Then there exists $\mathcal{E}_Q$ such that:
  - $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$
  - $\mathcal{E} = \mathcal{E}_Q \upharpoonright \{e \mid \lambda_Q(e) \notin A \cup \overline{A}\}$
  - $\mathsf{Init} = \mathsf{Init}_Q \cap \{e \mid \lambda_Q(e) \notin A \cup \overline{A}\}$
  - $k = k_Q \upharpoonright \{e \mid \lambda_Q(e) \notin A \cup \overline{A}\}$

  And there exists $\mathcal{E}'_Q$ such that:
  - $\{\!|Q'|\!\} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$
  - $\mathcal{E}' = \mathcal{E}'_Q \upharpoonright \{e \mid \lambda'_Q(e) \notin A \cup \overline{A}\}$

– $\mathsf{Init}' = \mathsf{Init}'_Q \cap \{e \mid \lambda'_Q(e) \notin A \cup \overline{A}\}$

– $k' = k'_Q \upharpoonright \{e \mid \lambda'_Q(e) \notin A \cup \overline{A}\}$

By inductions we have isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and $g : \mathcal{E}'_Q \to \mathcal{E}_Q$ and a transition $\mathsf{Init}_Q \xrightarrow{\{e_Q\}} X_Q$ or $C_{rb}(\mathcal{E}_Q)$ such that $\lambda_Q(e) = \mu$, $k'_Q(f_Q(e_Q)) = m$, and $f_Q(X_Q) = \mathsf{Init}'_Q$. We define our isomorphisms as $f_Q \upharpoonright \{e \mid \lambda'_Q(e) \notin A \cup \overline{A}\}$ and $g_Q \upharpoonright \{e \mid \lambda'_Q(e) \notin A \cup \overline{A}\}$. And since $\lambda(e) \notin A \cup \overline{A}$, the rest of the proof is straightforward.

- Suppose $P = Q[f']$, $P' = Q'[f']$, $Q \xrightarrow{\nu[m]} Q'$, and $f'(\nu) = \mu$. Then there exist λ_Q and Act_Q such that

  – $[\![Q]\!] = \langle (E, F, \mapsto, \sharp, \rhd, \lambda_Q, \mathsf{Act}_Q), \mathsf{Init}, k\rangle$,

  – $\mathsf{Act} = f'(\mathsf{Act}_Q)$

  – and $\lambda = f' \circ \lambda_Q$.

  And there exist λ'_Q and Act'_Q such that

  – $[\![Q']\!] = \langle (E', F', \mapsto', \sharp', \rhd', \lambda'_Q, \mathsf{Act}'_Q), \mathsf{Init}', k'\rangle$,

  – $\mathsf{Act}' = f'(\mathsf{Act}'_Q)$

  – and $\lambda = f' \circ \lambda'_Q$.

  By induction, we get isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and $g_Q : \mathcal{E}'_Q \to \mathcal{E}_Q$ and a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ in $C_{rb}(\mathcal{E}_Q)$ such that $\lambda_Q(e) = \nu$, $k'(f'(e)) = m$, and $f(X) = \mathsf{Init}'$. We define our isomorphisms $f = f_Q$ and $g = g_Q$, and the rest of the proof is straightforward.

- Suppose $P \equiv Q$, $P' \equiv Q'$, and $Q \xrightarrow{\mu[m]} Q'$. Then the result follows from induction and Proposition 5.1.15. □

C.7 Theorem 5.1.19

Proof. We say that $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$, and prove the theorem by induction on P.

- Suppose $P = 0$. Then $E = \emptyset$, and therefore no transitions exist in $C_{br}(\mathcal{E})$.

- Suppose $P = \alpha.Q$. Then $\{e_\alpha\} \mapsto e'$ for all $e' \in E \setminus \{e_\alpha\}$, meaning by definition $e = e_\alpha$. In addition, since P is reachable, clearly $\mathsf{std}(P)$ meaning $\mathsf{Init} = \emptyset$. This means we get $P \xrightarrow{\alpha[m]} \alpha[m].Q$ for some fresh m, and the isomorphisms are similar to this case in the proof of Theorem 5.1.18.

- Suppose $P = \alpha[n].Q$ and $[\![Q]\!] = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q\rangle$. Then $e_\alpha \in \mathsf{Init}$, and clearly $\mathsf{Init}_Q \xrightarrow{e} X_Q$, meaning there exists a a key m and a transition $Q \xrightarrow{\lambda(e)[m]} Q'$, such that $[\![Q']\!] = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q\rangle$ and there exist isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and $g_Q : \mathcal{E}'_Q \to \mathcal{E}_Q$ such that $k'_Q(f_Q(e)) = m$ and $f_Q(X_Q) = \mathsf{Init}'_Q$. If $m \neq n$, then $P \xrightarrow{\lambda(e)[m]} \alpha[m].P'_Q$. Otherwise, we can chose a fresh m and still get a transition. We define our isomorphisms as $f = f_Q[e_\alpha \mapsto e'_\alpha]$ and $g = g_Q[e'_\alpha \mapsto e_\alpha]$ and the rest of the proof is straightforward.

- Suppose $P = P_0 + P_1$, $[\![P_0]\!] = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $C_{br}(\mathcal{E}_0) = (E_0, F_0, \mathsf{C}_0, \to_0)$, $[\![P_1]\!] = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$, and $C_{br}(\mathcal{E}_1) = (E_1, F_1, \mathsf{C}_1, \to_1)$. Then either $\mathsf{Init}_0 \xrightarrow{e}_0 X_0$ and $\mathsf{Init}_1 = \emptyset$, or $\mathsf{Init}_1 \xrightarrow{e}_1 X_1$ and $\mathsf{Init}_0 = \emptyset$.

  If $\mathsf{Init}_0 \xrightarrow{e}_0 X_0$, then there exists a a key m and a transition $P_0 \xrightarrow{\lambda_0(e)[m]} P'_0$, such that $\{\!|P'_0|\!\} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$ and there exist isomorphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$ and $g_0 : \mathcal{E}'_0 \to \mathcal{E}_0$ such that $k'_0(f_0(e)) = m$ and $f_0(X_0) = \mathsf{Init}'_0$.

Then, since $\mathsf{Init}_1 = \emptyset$ means $\mathsf{std}(P_1)$, $P \xrightarrow{\lambda(e)[m]} P'_0 + P_1$, and the isomorphisms are similar to this case in the proof of Theorem 5.1.18.

If $\mathsf{Init}_1 \xrightarrow{e}_1 X_1$, then the proof is similar.

- Suppose $P = P_0 \mid P_1$, $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$, $C_{br}(\mathcal{E}_0) = (E_0, F_0, \mathsf{C}_0, \rightarrow_0)$, $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$, $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$, and $C_{br}(\mathcal{E}_1) = (E_1, F_1, \mathsf{C}_1, \rightarrow_1)$. Then either $e = (e_0, *)$, $e = (*, e_1)$, or $e = (e_0, e_1)$.

  If $e = (e_0, *)$, then whenever $X'_0 \mapsto_0 e_0$, we get $\{e' \in E \mid \pi_0(e') \in X'_0\} \mapsto e$. And whenever $\pi_0(e') \sharp_0 \pi_0(e)$, we get $e' \sharp e$. This means Init_0 is conflict-free, $\pi_0(X)$ is conflict-free, and $\mathsf{Init}_0 \xrightarrow{e_0}_0 \pi_0(X)$. There therefore exists a a key m and a transition $P_0 \xrightarrow{\lambda_0(e_0)[m]} P'_0$, such that $\{\!|P'_0|\!\} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \rangle$ and there exist isomorphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$ and $g_0 : \mathcal{E}'_0 \to \mathcal{E}_0$ such that $k'_0(f_0(e_0)) = m$ and $f_0(\pi_0(X)) = \mathsf{Init}'_0$.

  We chose an m, which is fresh for P_1, and we get $P \xrightarrow{\lambda_0(e_0)[m]} P'_0 \mid P_1$. We define our isomorphisms

$$f(e') = \begin{cases} (f_0(e'_0), *) & \text{if } e' = (e'_0, *) \\ (*, e'_1) & \text{if } e' = (*, e'_1) \\ (f_0(e'_0), e'_1) & \text{if } e' = (e'_0, e'_1) \end{cases}$$

  and

$$g(e') = \begin{cases} (g_0(e'_0), *) & \text{if } e' = (e'_0, *) \\ (*, e'_1) & \text{if } e' = (*, e'_1) \\ (g_0(e'_0), e'_1) & \text{if } e' = (e'_0, e'_1) \end{cases}$$

  Since $\mathcal{E}' = \mathcal{E}'_0 \times \mathcal{E}_1$, these are isomorphisms, and the rest of the case is straightforward.

  If $e = (e_0, *)$, the argument is similar.

  If $e = (e_0, e_1)$, then for $i \in \{0, 1\}$, whenever $X'_i \mapsto_i e_i$, we get $\{e' \in E \mid \pi_i(e') \in X'_i\} \mapsto e$. And whenever $\pi_i(e') \sharp_0 \pi_i(e)$, we get $e' \sharp e$. This means Init_i is conflict-free, $\pi_i(X)$ is conflict-free, and $\mathsf{Init}_i \xrightarrow{e_0}_i \pi_i(X)$. There therefore exists a a key m_i and a transition $P_i \xrightarrow{\lambda_i(e_i)[m_i]} P'_i$, such that $\{\!|P'_i|\!\} = \langle \mathcal{E}'_i, \mathsf{Init}'_i, k'_i \rangle$ and there exist isomorphisms $f_i : \mathcal{E}_0 \to \mathcal{E}'_i$ and $g_i : \mathcal{E}'_i \to \mathcal{E}_i$ such that $k'_i(f_i(e_i)) = m_i$ and $f_i(\pi_i(X)) = \mathsf{Init}'_0$.

  We say that $m_0 = m_1$ is a fresh m, and then since $\lambda_0(e_0) = \overline{\lambda_1(e_1)}$ and $\lambda(e) = \tau$, we get $P \xrightarrow{\lambda(e)[m]} P'_0 \mid P'_1$. We define our isomorphisms

$$f(e') = \begin{cases} (f_0(e'_0), *) & \text{if } e' = (e'_0, *) \\ ((*, f_1(e'_1)) & \text{if } e' = (*, e'_1) \\ (f_0(e'_0), f_1(e'_1)) & \text{if } e' = (e'_0, e'_1) \end{cases}$$

  and

$$g(e') = \begin{cases} (g_0(e'_0), *) & \text{if } e' = (e'_0, *) \\ ((*, g_1(e'_1)) & \text{if } e' = (*, e'_1) \\ (g_0(e'_0), g_1(e'_1)) & \text{if } e' = (e'_0, e'_1) \end{cases}$$

  Since $\mathcal{E}' = \mathcal{E}'_0 \times \mathcal{E}'_1$, these are isomorphisms, and the rest of the case is straightforward.

- Suppose $P = Q \setminus A$, $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}, k \rangle$, and $C_{br}(\mathcal{E}_Q) = (E_Q, F_Q, \mathsf{C}_Q, \rightarrow_Q)$. Then $\lambda(e) \notin A \cup \overline{A}$ and $\mathsf{Init} \xrightarrow{e}_Q X$, meaning there exists a key n and a transition $Q \xrightarrow{\lambda_Q(e)} Q'$ such that $\{\!|Q'|\!\} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$, and there exist isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and $g_Q : \mathcal{E}'_Q \to \mathcal{E}_Q$ such that $f_Q \circ k'_Q = k_Q[e \mapsto n]$ and $f_Q(X) = \mathsf{Init}'_Q$.

This means $P \xrightarrow{\lambda_Q(e)} Q' \setminus A$ and the morphisms $f \upharpoonright E$ and $g \upharpoonright \{e' \in E'_Q \mid \lambda'_Q(e') \notin A \cup \overline{A}\}$ clearly fulfil the remaining conditions.

- Suppose $P = Q[f]$, $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}, k \rangle$, and $C_{br}(\mathcal{E}_Q) = (E_Q, F_Q, \mathsf{C}_Q, \rightarrow_Q)$. Clearly $\mathsf{Init} \xrightarrow{e}_Q X$, and $f(\lambda_Q(e)) = \lambda(e)$, and the result follows. □

C.8 Lemma 5.2.8

Proof. We prove this by induction on the size of C.

Suppose $C = \emptyset$. Then $P_{\star C} = P$.

Suppose $P_{\star C'} \rightarrow^* P$ and $C = C' \cup \{n\}$ for some n such that if $m \leq_P n$ then $m \notin C'$. Then if there does not exist an action α and a tag γ such that $\alpha_\gamma[n]$ occurs in P, we get $P_{\star C} = P_{\star C'} \rightarrow^* P$. If there exists a process P', an action α and tag γ such that $\alpha_\gamma[n].P'$ is a subprocess of P then all past actions of P' are in C', meaning $P'_{\star C} = \mathsf{rt}(P')$, and since $m \leq_P n \Rightarrow m \notin C'$, we get $P_{\star C} \xrightarrow{\alpha_\gamma[n]} P_{\star C'} \rightarrow^* P$.

□

C.9 Theorem 5.2.12

Proof. To get P'' we apply first $\overset{\mathsf{start\ roll}\ \gamma_i}{\rightsquigarrow}$ for every $m_i \in T$ and then $\overset{\mathsf{roll}\ m_i}{\rightsquigarrow}$ for every $m_i \in T$ to P. We show that this is the correct P''.

Let P_r be P with every $\mathsf{rolling}\ \gamma$ replaced with $\mathsf{roll}\ \gamma$. Then $P_r \rightsquigarrow^* P''_r$, where P''_r is P'' with every $\mathsf{rolling}\ \gamma$ replaced with $\mathsf{roll}\ \gamma$ using the same rules as $P \rightsquigarrow^* P''$.

By the loop theorem we get $P''_r \rightarrow^* P_r$. And since $\phi(P_r) = \phi(P)$ and $\phi(P''_r) = \phi(P'')$, we can translate this computation into CCSK: $\phi(P'') \rightarrow^*_{CCSK} \phi(P)$. From the loop lemma of CCSK, this gives us $\phi(P) \rightsquigarrow^*_{CCSK} \phi(P'')$. And we get $\phi(P'') \not\rightsquigarrow_T$.

We then only need to show that if $\phi(P) \rightsquigarrow^*_T P' \not\rightsquigarrow_T$, then $P' = \phi(P'')$. Since they both reverse all the keys causally dependent on keys in T, this follows from Proposition 5.16 of [80]. □

C.10 Lemma 5.3.4

Proof. Be definition $E'_0 \subseteq E'_1$ and $F'_0 = F'_1 \cap E'_0$

We then prove that $X \mapsto'_0 e^*$ if and only if $X' \mapsto'_1 e^*$, $X = X' \cap E'_0$, and $e^* \in E'_0 \cup \underline{F'_0}$:

- If $X \mapsto'_0 e$ then either $X \mapsto_0 e$, or $X = \{e_\alpha\}$, $e \in E_0$, and $\lambda_0(e) \neq \mathsf{roll}\ \gamma'$.
  - If $X \mapsto_0 e$ then there exists some $X_1 \subseteq E_1$ such that $X_1 \cap E_0 = X_0$ and $X_1 \mapsto_1 e$, meaning $X_1 \mapsto'_1 e$ and $X_1 \cap E'_0 = X_0$.
  - If $X = \{e_\alpha\}$, $e \in E_0$, and $\lambda_0(e) \neq \mathsf{roll}\ \gamma'$ then $e \in E_1$ and $\lambda_1(e) = \lambda_0(e) \neq \mathsf{roll}\ \gamma'$, meaning $\{e_\alpha\} \mapsto'_1 e$.
- If $X \mapsto'_1 e$ and $e \in E'_0$ then either $X \mapsto_1 e$, or $X = \{e_\alpha\}$, $e \in E_1$, and $\lambda_1(e) \neq \mathsf{roll}\ \gamma'$.
  - If $X \mapsto_1 e$ and $e \in E'_0$ then $X \cap E_0 = X \cap E'_0 \mapsto_0 e$, meaning $X \cap E'_0 \mapsto_0 e$.
  - If $e \in E'_0$, $X = \{e_\alpha\}$, $e \in E_1$, and $\lambda_1(e) \neq \mathsf{roll}\ \gamma'$, then $\lambda_0(e) = \lambda_1(e) \neq \mathsf{roll}\ \gamma$, meaning $\{e_\alpha\} \mapsto'_0 e$.
- If $X \mapsto'_0 \underline{e}$ then either $X = \{e\}$, or $e = e_\alpha$ and $X = \{e' \in E_0 \mid \{e' \mid \lambda_0(e') = \mathsf{roll}\ \gamma$ and $e' \in X' \mapsto e'' \Rightarrow \lambda(e'') = \mathsf{start\ roll}\ \gamma\}\}$, or $\lambda_0(e) \in \{\mathsf{roll}\ \gamma' \mid \gamma'$ is a tag$\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'}$ or $\beta_{\gamma'}[n]$ occurs in $\alpha_\gamma.P_0\}$ and $X \mapsto_0 \underline{e}$, or $\lambda_0(e) \notin \{\mathsf{roll}\ \gamma' \mid \gamma'$ is a tag$\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'}$ or $\beta_{\gamma'}[n]$ occurs in $\alpha_\gamma.P_0\}$, $\{e\} \neq X' \mapsto_0 \underline{e}$, and $X = X' \cup \{e' \mid \lambda_0(e') = \mathsf{roll}\ \gamma\}$.
  - If $X = \{e\}$ then by definition $X \mapsto'_1 e$.

- If $e = e_\alpha$ and $X = \{e' \in E_0 \mid \lambda_0(e') = \mathsf{roll}\ \gamma\}$ then $X = \{e' \in E_1 \mid \lambda_1(e') = \mathsf{roll}\ \gamma\} \cap E_0$ and $\{e' \in E_1 \mid \lambda_1(e') = \mathsf{roll}\ \gamma\} \mapsto'_1 e$.
- If $\lambda_0(e) \in \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_0\}$ and $X \mapsto_0 \underline{e}$ then $\lambda_1(e) \in \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_1\}$ and there exists X_1 such that $X_1 \cap E_0 = X$ and $X_1 \mapsto_1 \underline{e}$, meaning $X_1 \mapsto'_1 \underline{e}$.
- If $\lambda_0(e) \notin \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_0\}$, $\{e\} \neq X' \mapsto_0 \underline{e}$, and $X = X' \cup \{e' \mid \lambda_0(e') = \mathsf{roll}\ \gamma\}$ then there exists $X'_1 \subseteq E_1$ such that $X'_1 \cap E_0 = X'$ and $X'_1 \mapsto_1 e$. This means $X'_1 \cup \{e' \in E_1 \mid \lambda_1(e') = \mathsf{roll}\ \gamma\} \mapsto'_1 e$, and clearly $\{e' \in E_1 \mid \lambda_1(e') = \mathsf{roll}\ \gamma\} \cap E_0 = \{e' \in E_0 \mid \lambda_0(e') = \mathsf{roll}\ \gamma\}$, meaning $(X'_1 \cup \{e' \in E_1 \mid \lambda_1(e') = \mathsf{roll}\ \gamma\}) \cap E'_0 = X$.

- If $X \mapsto'_1 \underline{e}$ and $e \in E'_0$ then either $X = \{e\}$, or $e = e_\alpha$ and $X = \{e' \in E_1 \mid \lambda_1(e') = \mathsf{roll}\ \gamma\}$, or $\lambda_1(e) \in \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_1\}$ and $X \mapsto_1 \underline{e}$, or $\lambda_1(e) \notin \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_1\}$, $\{e\} \neq X' \mapsto_1 \underline{e}$, and $X = X' \cup \{e' \mid \lambda_1(e') = \mathsf{roll}\ \gamma\}$.
  - If $X = \{e\}$ then by definition $X \mapsto'_0 e$.
  - If $e = e_\alpha$ and $X = \{e' \in E_1 \mid \lambda_1(e') = \mathsf{roll}\ \gamma\}$ then $\{e' \in E_0 \mid \lambda_0(e') = \mathsf{roll}\ \gamma\} \mapsto'_0 \underline{e}$ and therefore $X \cap E'_0 = \{e' \in E_0 \mid \lambda_0(e') = \mathsf{roll}\ \gamma\}$.
  - If $\lambda_1(e) \in \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_1\}$ and $X \mapsto_1 \underline{e}$ then $\lambda_0(e) \in \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_0\}$, and $X \cap E_0 \mapsto_0 \underline{e}$, meaning $X \cap E_0 \mapsto'_0 \underline{e}$.
  - If $\lambda_1(e) \notin \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_1\}$, $\{e\} \neq X' \mapsto_1 \underline{e}$, and $X = X' \cup \{e' \mid \lambda_1(e') = \mathsf{roll}\ \gamma\}$ then $X' \cap E_0 \mapsto_0 \underline{e}$, meaning $(X' \cap E_0) \cup \{e' \mid \lambda_0(e') = \mathsf{roll}\ \gamma\} \mapsto'_0 \underline{e}$, and therefore $X \cap E_0 = (X' \cap E_0) \cup \{e' \mid \lambda_0(e') = \mathsf{roll}\ \gamma\}$.

We then prove that $e \rhd'_0 e'^*$ if and only if $e \rhd'_1 e'^*$, $e \in E'_0$, and $e' \in E'_0 \cap \underline{F'_0}$.

- If $e \rhd'_0 e'^*$ then either $\lambda_0(e) \in \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_0\}$ and $e'^* = \underline{e_\alpha}$, or $e = e_\alpha$, $e'^* = \underline{e'}$ and $\lambda_0(e') = \mathsf{roll}\ \gamma$, or $\lambda_0(e) = \mathsf{roll}\ \gamma$ and $e'^* = e_\alpha$, or $\lambda_0(e) = \mathsf{roll}\ \gamma$, $e'^* = e'$, and $\lambda_0(e') \notin \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_0\}$.
  - If $\lambda_0(e) \in \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_0\}$ and $e'^* = \underline{e_\alpha}$, then $\lambda_1(e) \in \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_1\}$, and therefore $e \rhd'_1 \underline{e_\alpha}$.
  - If $e = e_\alpha$, $e'^* = \underline{e'}$ and $\lambda_0(e') = \mathsf{roll}\ \gamma$ then $\lambda_1(e') = \mathsf{roll}\ \gamma$, and therefore $e_\alpha \rhd'_1 e'$.
  - If $\lambda_0(e) = \mathsf{roll}\ \gamma$ and $e'^* = e_\alpha$, then $\lambda_1(e) = \mathsf{roll}\ \gamma$, and therefore $e \rhd'_1 e_\alpha$.
  - If $\lambda_0(e) = \mathsf{roll}\ \gamma$, $\lambda_0(e') \notin \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\} \cup \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_0\}$, and $e'^* = e'$, then we get $\lambda_1(e) = \mathsf{roll}\ \gamma$ and $\lambda_1(e') \notin \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P_0\} \cup \{\mathsf{roll}\ \gamma' \mid \gamma' \text{ is a tag}\}$, meaning $e \rhd'_1 e'$.
- If $e \rhd'_1 e'^*$ for $e, e' \in E_0$ then the argument is similar.

By definition $\lambda'_0 = \lambda'_1 \upharpoonright_{E'_0}$ and $\mathsf{Act} = \mathsf{ran}(\lambda'_0)$. □

C.11 Lemma 5.3.6

Proof. We first prove that $E'_0 \subseteq E'_1$. For all $e \in E'_0$ we know either $e \in E_0 \times_* E_2$ and $\lambda'_0(e) \in \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$, or $e = (X, e')$ for some $e' \in E_0 \times_* E_2$ and $X \in \mathsf{causes}(e')$. If $e \in E_0 \times_* E_2$ and $\lambda'_0(e) \in \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$ then by definition $e \in E'_1$. If $e = (X, e')$ then if $e' = (e_0, *)$ then for each $(e'_0, e'_2) \in X$ there exists X_0 such that $e'_0 \in X_0$ and $X_0 \mapsto_0 e_0$. This means there exists $X_1 \subseteq E_1$ such that $X_1 \mapsto_1 e_0$ and $X_0 = X_1 \cap E_0$. In addition, for any $X'_1 \subseteq E_1$ such that $X'_1 \mapsto_1 e_0$, we have $X'_1 \cap E_0 \mapsto_0 e_0$, and therefore $(X'_1 \times E_2) \cap X \neq \emptyset$. We therefore

get $e \in E_1'$. If $e' = (*, e_2)$ then e_2's causes are the same in $\mathcal{E}_1'$ and therefore $e' \in E_1'$. If $e' = (e_0, e_2)$ then the argument is a combination of the first two cases.

By definition $F_0' = E_0' \cap F_1'$.

We then prove that $X \mapsto_0' e^*$ if and only if $X' \mapsto_1' e^*$, $X = X' \cap E_0'$, and $e^* \in E_0' \cup \underline{F_0'}$:

- If $X \mapsto_0' e$ then either $e = (X', e')$, $X = \{(X'', e'') \mid X'' \subseteq X'\}$, and $e'' \in X'$, or $e = (e_0, e_2)$ and there exists X' such that $X' \mapsto_{0\times 2} e$ and $X = \{e' \mid (\pi_0(e'), \pi_2(e')) \in X'\}$.
  - If $e = (X', e')$, $X = \{(X'', e'') \mid X'' \subseteq X'\}$, and $e'' \in X'$ then clearly $X \mapsto_1' e$.
  - If $e = (e_0, e_2)$ and there exists X' such that $X' \mapsto_{0\times 2} e$ and $X = \{e' \mid (\pi_0(e'), \pi_2(e')) \in X'\}$ then $X' \mapsto_{1\times 2} e$, and therefore $\{e' \mid (\pi_0(e'), \pi_2(e')) \in X'\} = \{e' \mid (\pi_1(e'), \pi_2(e')) \in X'\} \cap E_0'$.
- If $X \mapsto_1' e$ and $e \in E_0'$ then either $e = (X', e')$, $X = \{(X'', e'') \mid X'' \subseteq X'\}$, and $e'' \in X'$, or $e = (e_1, e_2)$ and there exists X' such that $X' \mapsto_{1\times 2} e$ and $X = \{e' \mid (\pi_1(e'), \pi_2(e')) \in X'\}$.
  - If $e = (X', e')$, $X = \{(X'', e'') \mid X'' \subseteq X'\}$, and $e'' \in X'$ then since $e \in E_0$, for all $e''' \in X'$, $\pi_1(e''') \in E_0$, meaning $X \subseteq E_0'$, and $\mapsto_0' e$.
  - If $e = (e_1, e_2)$ and there exists X' such that $X' \mapsto_{1\times 2} e$ and $X = \{e' \mid (\pi_1(e'), \pi_2(e')) \in X'\}$ then $e_1 \in E_0 \cup \{*\}$, and $X' \cap (E_0 \times_* E_2) \mapsto_{0\times 2} e$, meaning $X \cap E_0' = \{e' \mid (\pi_0(e'), \pi_2(e')) \in X' \cap (E_0 \times_* E_2)\} \mapsto e$.
- If $X \mapsto_0' \underline{e}$ then $X = \bigcup \left\{ X'' \;\middle|\; \begin{array}{c} \exists i \in \{0,2\}, X_i \in E_i . X_i \mapsto \underline{\pi_i(e)} \text{ or } \exists e_\times \in X' . X_i \mapsto \underline{\pi_i(e_\times)} \\ \text{and } e' \in X'' \text{ iff } \pi_i(e') \in X_i \end{array} \right\}$ and $e = (X', (e_0, e_2))$, or $e = (e_0, e_1)$ and there exists X' such that $X' \mapsto_{\times_0} e$ and $X = \{e' \mid (\pi_0(e'), \pi_1(e')) \in X'\}$, or $X = \{e\}$.
  - If $X = \{e\}$ then by definition $X \mapsto_1' \underline{e}$.
  - If $e = (X', (e_0, e_2))$ and $X = \left\{ X'' \;\middle|\; \begin{array}{c} \exists i \in \{0,2\}, X_i \in E_i . X_i \mapsto_i \underline{\pi_i(e)} \text{ or } \exists e_\times \in X' . X_i \mapsto_i \underline{\pi_i(e_\times)} \\ \text{and } e' \in X'' \text{ iff } \pi_i(e') \in X_i \end{array} \right\}$ then (1) for each X_0 such that $X_0 \mapsto_0 \underline{e_0}$, there exists X_1 such that $X_1 \mapsto_1 \underline{e_0}$ and $X_1 \cap E_0 = X_0$, and for each X_0 such that there exists $(e_0', e_2') \in X'$, such that $X_0 \mapsto_0' \underline{e_0}$, there exists X_1 such that $X_1 \mapsto_1 \underline{e_0'}$ and $X_1 \cap E_0 = X_0$, meaning
    $$\bigcup \left\{ X'' \;\middle|\; \begin{array}{c} \exists i \in \{1,2\}, X_i \in E_i . X_i \mapsto_i \underline{\pi_i(e)} \text{ or } \exists e_\times \in X' . X_i \mapsto_i \underline{\pi_i(e_\times)} \\ \text{and } e' \in X'' \text{ iff } \pi_i(e') \in X_i \end{array} \right\} \cap E_0 = X$$
  - If $e = (e_0, e_1)$ and there exists X' such that $X' \mapsto_{0\times 2} e$ and $X = \{e' \mid (\pi_0(e'), \pi_1(e')) \in X'\}$, then by Lemma 5.1.10, there exists X'' such that $X' = X'' \cap (E_0 \times_* E_2)$ and $X'' \mapsto_{1\times 2} e$, meaning $\{e' \mid (\pi_1(e'), \pi_1(e')) \in X''\} \mapsto_1' e$.
- If $X \mapsto_1' \underline{e}$ and $e \in E_0'$ then $X = \{e\}$, or $e = (X', (e_1, e_2))$ and $X = \bigcup \{X'' \mid \exists i \in \{0,1\} . \pi_i(X'') \mapsto \underline{e_i}$ or $\exists e' \in X' . \pi_i(X'') \mapsto_i \pi_i(\underline{e'})\}$. If $X = \{e\}$ then by definition $X \mapsto_0' \underline{e}$. If $e = (X', (e_1, e_2))$ and $X = \bigcup \left\{ X'' \;\middle|\; \begin{array}{c} \exists i \in \{1,2\}, X_i \in E_i . X_i \mapsto_i \underline{\pi_i(e)} \text{ or } \exists e_\times \in X' . X_i \mapsto_i \underline{\pi_i(e_\times)} \\ \text{and } e' \in X'' \text{ iff } \pi_i(e') \in X_i \end{array} \right\}$ then for each X_1 such that $X_1 \mapsto \underline{e_1}$, we know $X_1 \cap E_0 \mapsto \underline{e_1}$, and for each X_1 such that there exists $(e_1', e_2') \in X'$ such that $X_1 \mapsto_1' \underline{e_1}$, since $e \in E_0'$, $e_1' \in E_0$, meaning $X_1 \cap E_0 \mapsto \underline{e_1}$. Therefore $X \cap E_0' \mapsto_0' e_1$.

We then prove that $e \rhd_0' e'^*$ if and only if $e \rhd_1' e'^*$, $e \in E_0'$, and $e' \in E_0' \cap \underline{F_0'}$.

If $e \rhd_0' e'^*$ then there exists $i \in \{0,2\}$ such that either (1) $\pi_i(e) \rhd_i \pi_i(e')^*$, or (2) $\pi_i(e) = \pi_i(e') \neq \bot$, or (3) $e'^* = e'$, $e \neq e'$, and $e \in X \mapsto \underline{e'}$, or (4) and there exist γ, γ' such that $\lambda(e) = \mathsf{roll}\ \gamma$ and $\lambda(e') = \mathsf{roll}\ \gamma'$. In all these cases it is clear that the same conditions will apply in $\mathcal{E}_1'$.

Similar logic applies if $e \rhd_1' e'^*$ and $e, e' \in E_0'$.

And finally, by definition, $\lambda_0' = \lambda_1' \upharpoonright_{E_0'}$ and $\mathsf{Act} = \mathsf{ran}(\lambda_0')$. □

C.12 Lemma 5.3.9

Proof. We prove this by structural induction in P.

- Suppose $P = 0$. Then there are no events and the lemma is trivially true.
- Suppose $P = \mathsf{roll}\ \gamma$. Then $e \in X \mapsto e'$ means $e = e_s$ and $e' = e_r$, and therefore $\lambda(e') = \mathsf{roll}\ \gamma$.
- Suppose $P = \mathsf{rolling}\ \gamma$. Then the argument is the same as the previous case.
- Suppose $P = \alpha_{\gamma'}.P'$. Then $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}, k\rangle$ and either $X \mapsto' e'$ or $X = \{e_\alpha\}$. If $X \mapsto' e'$ then $e' \neq e_\alpha$ and by induction if $e' \not\triangleright' \underline{e}$ then $\lambda(e') \in \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$. If $X = \{e_\alpha\}$ then, $e' \rhd e_\alpha$ unless $\lambda(e') \in \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$.
- Suppose $P = \alpha_{\gamma'}.[m]P'$. Then the argument is the same as the previous case.
- Suppose $P = P_0 + P_1$. Then $e = (i, e_i)$, $e' = (i, e_i')$, $e_i \mapsto_i e_i'$, and $e_i' \not\triangleright_i \underline{e_i}$, meaning $\lambda_i(e_i') \in \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$, and therefore $\lambda(e') \in \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$.
- Suppose $P = P_0 \mid P_1$. Then if $e' = (Y', e'')$, $e = (Y, e''')$ and $e''' \in Y'$, meaning there exists $i \in \{0, 1\}$ such that $\pi_i(e) \in X_i \mapsto_i \pi_i(e')$. By induction we get that, if $e_i' \not\triangleright_i \underline{e_i}$, then there exists γ such that $\lambda_i(e_i') \in \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$, meaning $e'_{1-i} = *$ and $\lambda(e') \in \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$, and if $e_i' \rhd_i \underline{e_i}$ then $e' \rhd \underline{e}$. If $e' \in E_\times$ then $\lambda(e') \in \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$
- Suppose $P = P' \setminus A$. Then the result follows from induction.
- Suppose $P = A\langle \tilde{a}, \tilde{\gamma}\rangle$ and $A(\tilde{b}, \tilde{\delta}) = P_A$. Then the result holds if it holds for P_A. □

C.13 Lemma 5.3.10

Proof. We prove this by structural induction on P.

- Suppose $P = 0$. Then there are no events and the lemma is trivially true.
- Suppose $P = \mathsf{roll}\ \gamma$. Then no X, X', e, e' exist such that $X \mapsto e \in X' \mapsto e'$.
- Suppose $P = \mathsf{rolling}\ \gamma$. Then the argument is the same as the previous case.
- Suppose $P = \alpha_{\gamma'}.P'$. Then $\{\!|P'|\!\} = \langle \mathcal{E}', \mathsf{Init}, k\rangle$, $X' \mapsto' e'$, and either $X = \{e_\alpha\}$, or $X \mapsto' e$. If $X = \{e_\alpha\}$, then $X \mapsto e'$ whenever $\lambda(e') \neq \mathsf{roll}\ \gamma'$. If $X \mapsto' e$ and $X' \mapsto' e'$ then by induction, $X \mapsto e'$.
- Suppose $P = \alpha_{\gamma'}.[m]P'$. Then the argument is the same as the previous case.
- Suppose $P = P_0 + P_1$. Then there exists an $i \in \{0, 1\}$ such that $e = (i, e_i)$, $e' = (i, e_i')$, $\{e_i'' \mid (i, e_i'') \in X'\} \mapsto_i e_i'$, and $\{e_i'' \mid (i, e_i'') \in X\} \mapsto_i e_i$, meaning by induction $\{e_i'' \mid (i, e_i'') \in X\} \mapsto_i e_i'$, and therefore $X \mapsto e'$.
- Suppose $P = P_0 \mid P_1$. Then $e' = (Y', (e_0', e_1'))$ or there exists a γ such that $\lambda(e') = \mathsf{roll}\ \gamma$. If $e' = (Y', (e_0', e_1'))$ then $e = (Y, (e_0, e_1))$ and $(e_0, e_1) \in Y'$, meaning there exists $i \in \{0, 1\}$ such that $e_i \in X_i \mapsto_i e_i'$. Similarly, $X = \{(Y'', e'') \mid (Y'', e'') \in E\}$ for some $e'' \in Y$. Since $Y \in \mathsf{cause}(e)$ and $e \in Y' \in \mathsf{cause}(e')$, there exists $Y'' \in \mathsf{cause}(e')$ such that $Y \subseteq Y''$. This means $X \mapsto e$.
- Suppose $P = P' \setminus A$. Then the result follows from induction.
- Suppose $P = A\langle \tilde{a}, \tilde{\gamma}\rangle$ and $A(\tilde{b}, \tilde{\delta}) = P_A$. Then the result holds if it holds for P_A. □

C.14 Lemma 5.3.11

Proof. We prove this by structural induction on P.

- Suppose $P = 0$, then $E = \emptyset$ and the case is trivial.
- Suppose $P = \mathsf{roll}\ \gamma$. Then $e' = e_s$ and $e = e_r$.
- Suppose $P = \mathsf{rolling}\ \gamma$. Then $e' = e_s$ and $e = e_r$.
- Suppose $P = \alpha_\gamma.P'$. Then by induction, if $X \mapsto_{P'} e$, then there exists an e' such that $X = \{e'\}$. If $X \not\mapsto_{P'} e$, then $X = \{e_\alpha\}$.
- Suppose $P = \alpha_\gamma[m].P'$. Then by induction, if $X \mapsto_{P'} e$, there exists an e' such that $X = \{e'\}$. If $X \not\mapsto_{P'} e$, then $X = \{e_\alpha\}$.
- Suppose $P = P_0 + P_1$. Then, by induction if $e = (i, e_i)$ and $X_i \mapsto_i e_i$, then $X_i = e_i'$, and $e' = (i, e_i')$.
- Suppose $P = P_0 \mid P_1$. Then either $e = (Y, e_\times)$ or there exists a γ such that $\lambda(e) = \mathsf{roll}\ \gamma$. If $e = (Y, e_\times)$ then $X = \{(Y'', e'') \mid (Y'', e'') \in E\}$ for some $e'' \in Y$. We therefore need to show that given an event $e'' \in Y$, there exists exactly one $Y'' \in \mathsf{cause}(e'')$ such that $Y'' \subseteq Y$. This follows naturally from items 2 and 3 of Definition 5.3.1.
- Suppose $P = P' \setminus A$. Then the result follows from the definition of ρ (Definition 5.3.3).
- Suppose $P = A\langle \tilde{a}, \tilde{\gamma} \rangle$ and $A(\tilde{b}, \tilde{\delta}) = P_A$. Then the lemma holds if it holds for P_A. □

C.15 Lemma 5.3.12

Proof. We prove this by structural induction on P:

- Suppose $P = 0$, then $E = \emptyset$ and the case is trivial.
- Suppose $P = \mathsf{roll}\ \gamma$. Then $e' = e_r$ and $e = e_s$.
- Suppose $P = \mathsf{rolling}\ \gamma$. Then $e' = e_r$ and $e = e_s$.
- Suppose $P = \alpha_\gamma.P'$. Then either $e = e_\alpha$ and $X = \{e'' \mid \lambda'_{P'}(e'') = \mathsf{roll}\ \gamma\}$, or $\lambda_{P'}(e) = \mathsf{roll}\ \gamma'$ and $X \mapsto_{P'} \underline{e}$, or $\lambda'_{P'}(e'') \neq \mathsf{roll}\ \gamma'$, $\{e\} \neq X' \mapsto_{P'} e$, and $X = X' \cup \{e'' \mid \lambda'_{P'}(e'') = \mathsf{roll}\ \gamma\}$.

  In either case, it is clear that $e' \rhd \underline{e}$.
- Suppose $P = \alpha_\gamma[m].P'$. Then the argument is similar to the previous case.
- Suppose $P = P_0 + P_1$. Then, by induction if $e = (i, e_i)$ and $e_i' \in X_i \mapsto_i \underline{e_i}$, then $e_i' \rhd \underline{e_i}$, and $e' = (i, e_i')$, meaning $e' \rhd \underline{e}$.
- Suppose $P = P_0 \mid P_1$. Then the lemma holds by definition.
- Suppose $P = P' \setminus A$. Then the result follows from induction.
- Suppose $P = A\langle \tilde{a}, \tilde{\gamma} \rangle$ and $A(\tilde{b}, \tilde{\delta}) = P_A$. Then the result holds if it holds for P_A. □

C.16 Lemma 5.3.13

Proof. We prove this by structural induction in P:

- Suppose $P = 0$, then $E = \emptyset$ and the case is trivial.
- Suppose $P = \mathsf{roll}\ \gamma$. Then $X = \{e_r\}$ and $e = e_s$.
- Suppose $P = \mathsf{rolling}\ \gamma$. Then $X = \{e_r\}$ and $e = e_s$.
- Suppose $P = \alpha_\gamma.P'$. Then either $e = e_\alpha$ and $X = \{e'' \mid \lambda'_{P'}(e'') = \mathsf{roll}\ \gamma\}$, or $\lambda_{P'}(e) = \mathsf{roll}\ \gamma'$ and $X \mapsto_{P'} \underline{e}$, or $\lambda'_{P'}(e'') \neq \mathsf{roll}\ \gamma'$, $\{e\} \neq X' \mapsto_{P'} e$, and $X = X' \cup \{e'' \mid \lambda'_{P'}(e'') = \mathsf{roll}\ \gamma\}$.

  In either case, it is clear that there exists only one X.
- Suppose $P = \alpha_\gamma[m].P'$. Then the argument is similar to the previous case.
- Suppose $P = P_0 + P_1$. Then, by induction if $e = (i, e_i)$ then there exists at most one X_i such that $e'_i \in X_i \mapsto_i \underline{e_i}$, meaning $\{i\} \times X_i \mapsto e$.
- Suppose $P = P_0 \mid P_1$. Then either $e = (X', e')$, in which case the lemma holds by definition, or there exists X' such that $X' \mapsto_\times e$ and $X = \{e' \mid (\pi_0(e'), \pi_1(e')) \in X'\}$. By induction, since there exists an $i \in \{0, 1\}$ such that $\pi_i(e) = \bot$, there can only exist one such X'.
- Suppose $P = P' \setminus A$. Then the result follows from induction.
- Suppose $P = A\langle \tilde{a}, \tilde{\gamma} \rangle$ and $A(\tilde{b}, \tilde{\delta}) = P_A$. Then the result holds if it holds for P_A. □

C.17 Lemma 5.3.14

Proof. We prove this by structural induction in P:

- Suppose $P = 0$, then $E = \emptyset$ and the case is trivial.
- Suppose $P = \mathsf{roll}\ \gamma$. Then there does not exist any e such that $\lambda(e) = \mu$.
- Suppose $P = \mathsf{rolling}\ \gamma$. Then there does not exist any e such that $\lambda(e) = \mu$.
- Suppose $P = \alpha_\gamma.P'$. Then either $X \mapsto_{P'} e$ or $X = \{e_\alpha\}$ and $e \in E_{P'}$.

  If $X \mapsto_{P'} e$ then there exists an $X_{P'}$ such that $X_{P'} \mapsto_{P'} \underline{e}$ and $X' = X_{P'} \cup \{e'' \mid \lambda_{P'}(e'') = \mathsf{roll}\ \gamma\}$. This means there exists $X'_{P'}$ such that $X'_{P'} \mapsto_{P'} \underline{e'}$, $X_{P'} \subseteq X'_{P'}$, and $X'_{P'} \cup \{e'' \mid \lambda_{P'}(e'') = \mathsf{roll}\ \gamma\} = X'' \mapsto \underline{underlinee}$.

  If $X = \{e_\alpha\}$ and $e \in E_{P'}$ then $X' = \{e'' \mid \lambda_{P'}(e'') = \mathsf{roll}\ \gamma\}$ and there exists an $X_{P'}$ such that $X_{P'} \mapsto_{P'} \underline{e}$ and $X'' = X_{P'} \cup \{e'' \mid \lambda_{P'}(e'') = \mathsf{roll}\ \gamma\}$, meaning clearly $X' \subseteq X''$.
- Suppose $P = \alpha_\gamma[m].P'$. Then the argument is similar to the previous case.
- Suppose $P = P_0 + P_1$. Then, if $e = (i, e_i)$ and $e' = (i, e'_i)$ then there exists X'_i such that $X' = \{i\} \times X'_i$ and $X'_i \mapsto \underline{e'_i}$, meaning there exists $X''_i \supseteq X'_i$ such that $X''_i \mapsto \underline{e_i}$ and therefore $\{i\} \times X''_i = X'' \mapsto \underline{e}$.
- Suppose $P = P_0 \mid P_1$. Then either $e = (Y, e_\times)$, or there exists X' such that $X' \mapsto_\times e$ and $X = \{e' \mid (\pi_0(e'), \pi_1(e')) \in X'\}$.

  If $e = (Y, e_\times)$ then $e' = (Y', e'_\times)$ and $Y' \cup \{e'_\times\} \subseteq Y$, and

$$X' = \bigcup \left\{ X''' \;\middle|\; \begin{array}{l} \exists i \in \{0,1\}, X_i \in E_i . X_i \mapsto_i \underline{\pi_i(e')} \text{ or } \exists e''_\times \in Y' . X_i \mapsto_i \underline{\pi_i(e''_\times)} \\ \text{and } e'' \in X''' \text{ iff } \pi_i(e'') \in X_i \end{array} \right\}.$$

We define $X'' = \bigcup \left\{ X''' \;\middle|\; \begin{array}{l} \exists i \in \{0,1\}, X_i \in E_i . X_i \mapsto_i \underline{\pi_i(e)} \text{ or } \exists e''_\times \in Y . X_i \mapsto_i \underline{\pi_i(e''_\times)} \\ \text{and } e'' \in X''' \text{ iff } \pi_i(e'') \in \underline{X_i} \end{array} \right\}$ and show that $X' \subseteq X''$. By definition, since $e_\times \in Y'$, whenever $X_i \mapsto \pi_i(e')$ we get $\pi(e'') \in X_i$ iff $e'' \in X''$. And if there exists $e''_\times \in Y'$ such that $X_i \mapsto \pi_i(e''_\times)$ then, since $Y' \subseteq Y$, $e''_\times \in Y$ and therefore $\pi(e'') \in X_i$ iff $e'' \in X''$.

- Suppose $P = P' \setminus A$. Then the lemma follows from induction.
- Suppose $P = A\langle \tilde{a}, \tilde{\gamma} \rangle$. Then the lemma holds if it holds for P_A. □

C.18 Proposition 5.3.15

Proof. We say that $\mathcal{E} = (E, F, \mapsto, \triangleright, \lambda, \mathsf{Act})$ and $\mathcal{E}' = (E', F', \mapsto', \triangleright', \lambda', \mathsf{Act}')$ and do a case analysis on the structural congruence rules:

$P = Q \mid R$ and $P' = R \mid Q$: Then there exist $\mathcal{E}_Q$ and $\mathcal{E}_R$ such that for $i \in \{Q, R\}$, $[\![P_i]\!] = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i \rangle$ and $\langle \mathcal{E}, \mathsf{Init}, k \rangle$ is composed of them as defined in the event structure semantics.

And there exist $\mathcal{E}'_Q$ and $\mathcal{E}'_R$ such that for $i \in \{Q, R\}$, $[\![P_i]\!] = \langle \mathcal{E}'_i, \mathsf{Init}'_i, k'_i \rangle$ and $\langle \mathcal{E}', \mathsf{Init}', k' \rangle$ is composed of them as defined in the event structure semantics.

And by induction we have isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and $f_R : \mathcal{E}_R \to \mathcal{E}'_R$ fulfilling the conditions.

We first define a helper function

$$f'(e) = \begin{cases} (f_R(e_R), f_Q(e_Q)) & \text{if } e = (e_Q, e_R) \\ (f_R(e_R), *) & \text{if } e = (*, e_R) \\ (*, f_Q(e_Q)) & \text{if } e = (e_Q, *) \end{cases}$$

and then our isomorphism

$$f(e) = \begin{cases} (\{f'(e'') \mid e'' \in X\}, f'(e')) & \text{if } e = (X, e') \\ f'(e) & \text{otherwise} \end{cases}$$

Since the definition of parallel compositions treat both parts the same way, this clearly fulfils the conditions.

$P = P_0 \mid (P_1 \mid P_2)$ and $P' = (P_0 \mid P_1) \mid P_2$: Then there exist $\mathcal{E}_0$, $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_{12}$ such that $[\![P_0]\!] = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $[\![P_1]\!] = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$, $[\![P_2]\!] = \langle \mathcal{E}_2, \mathsf{Init}_2, k_2 \rangle$, $\langle \mathcal{E}_{12}, \mathsf{Init}_{12}, k_{12} \rangle$ is composed of $\langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$ and $\langle \mathcal{E}_2, \mathsf{Init}_2, k_2 \rangle$ as described in the parallel rule, and $\langle \mathcal{E}, \mathsf{Init}, k \rangle$ is composed of $\langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$ and $\langle \mathcal{E}_{12}, \mathsf{Init}_{12}, k_{12} \rangle$ as described in the parallel rule.

And there exist $\mathcal{E}'_0$, $\mathcal{E}'_1$, $\mathcal{E}'_2$, and $\mathcal{E}_{01}$ such that $[\![P_0]\!] = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \rangle$, $[\![P_1]\!] = \langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1 \rangle$, $[\![P_2]\!] = \langle \mathcal{E}'_2, \mathsf{Init}'_2, k'_2 \rangle$, $\langle \mathcal{E}_{01}, \mathsf{Init}_{01}, k_{01} \rangle$ is composed of $\langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$ and $\langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$ as described in the parallel rule, and $\langle \mathcal{E}', \mathsf{Init}', k' \rangle$ is composed of $\langle \mathcal{E}_Z, \mathsf{Init}_Z, k_Z \rangle$ and $\langle \mathcal{E}_{01}, \mathsf{Init}_{01}, k_{01} \rangle$ as described in the parallel rule. And there exist isomorphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$, $f_1 : \mathcal{E}_1 \to \mathcal{E}'_1$, and $f_2 : \mathcal{E}_2 \to \mathcal{E}'_2$ satisfying the conditions of the proposition.

We define a helper function $f_{01}((e_0, e_1)) = (f_0(e_0), f_1(e_1))$ if $e_0 \in E_0$ and $e_1 \in E_1$ and define the morphism

$$f(e) = \begin{cases} (f_{01}((e_0, e_1)), f_2(e_2)) & \text{if } e = (e_0, (e_1, e_2)) \\ (Y, ((Y', f_{01}((e_0, e_1))), f_2(e_2))) & \text{if } e = (X, (e_0, (X', (e_1, e_2)))), \\ & Y' = \{ f_{01}((e'_0, e'_1)) \mid \exists e'_2, X'' . (e'_0, (X'', (e'_1, e'_2))) \in X \\ & \text{and } e'_0 \in X_0 \in \mathsf{cause}(e_0) \text{ or } e'_1 \in X_1 \in \mathsf{cause}(e_1) \}, \\ & \text{and } Y = \{ ((f_{01}(Y''), f_{01}((e'_0, e'_1))), e'_2) \in E_{01\times 2} \mid \\ & \exists X'' . (e'_0, (X'', (e'_1, e'_2))) \in X \text{ and} \\ & (f_{01}(Y''), f_{01}((e'_0, e'_1))) \in Y' \} \end{cases}$$

We first show that for any $e = (X, (e_0, (X', (e_1, e_2))))$, there exists at most one possible $f(e) \in E'$: Since causes must be conflict free, there can at most exist one e'_2 and X'' for each e'_0 and e'_1 such that $(e'_0, (X'', (e'_1, e'_2))) \in X$, meaning there can only exist one Y' and Y fulfilling the conditions.

We then show that for any $e = (X, (e_0, (X', (e_1, e_2))))$, there exists $f(e') = (Y, ((Y', (e'_0, e'_1)), e'_2)) \in E'$: By induction, $e'_0 \in E'_0$, $e'_1 \in E'_1$, and $e'_2 \in E'_2$, so we show that $(Y', (e'_0, e'_1)) \in E_{01}$. We know there exists $X_1 \in \mathsf{cause}(e_1)$ such that $X_1 \subseteq \pi_1(X') = \pi_1(\pi_{12}(X))$, and there exists $X_0 \in \mathsf{cause}(e_0)$ such that $X_0 \subseteq \pi_0(X)$. And since for all $e \in Y'$, either $e'_0 \in X_0 \in \mathsf{cause}(e_0)$ or $e'_Y \in X_Y \in \mathsf{cause}(e_1)$, we get that $Y' \in \mathsf{cause}((f_0(e_0), f_1(e_1)))$, and therefore $(Y', (f_0(e_0), f_1(e_1))) \in E_{01}$. And for similar reasons $Y \in \mathsf{cause}(((Y', (f_0(e_0), f_1(e_1))), f_2(e_2)))$, meaning $(Y, ((Y', (f_0(e_0), f_1(e_1))), f_2(e_2))) \in E'$.

We then show that for any $e' = (X, ((X', (e'_0, e'_1)), e'_2)) \in E'$, there exists $e = (Y, (e_0, (Y', (e_1, e_2)))) \in E$ such that $f(e) = e'$. By induction, there exist e_0, e_1, e_2 such that $f_0(e_0) = e'_0$, $f_1(e_1) = e'_1$, and $f_2(e_2) = e'_2$. We also know there exist $X_0 \in \mathsf{cause}(e'_0)$, $X_1 \in \mathsf{cause}(e'_1)$, and $X_2 \in \mathsf{cause}(e'_2)$ such that (1) whenever $((X'', (e''_0, e''_1)), e''_2)) \in X$, either $e''_0 \in X_0$ or $e''_1 \in X_1$ or $e''_2 \in X_2$, and for each $(e'''_0, e'''_1) \in X''$, there exists $X''' \subseteq X''$ and e'''_2 such that $((X''', (e'''_0, e'''_1)), e'''_2) \in X$; and (2) whenever $e_i \in X_i$, there exists $((X'', (e''_0, e''_1)), e''_2)) \in X$ such that $e_i \in \{e''_0, e''_1, e''_2\}$.

For $i \in \{0, 1, 2\}$, since f_i is an isomorphism, $f_i^{-1}(X_i) \in \mathsf{cause}(e_i)$, meaning if we set

$$Y' = \{(f_1^{-1}(e''_1), f_2^{-1}(e''_2)) \mid ((X'', (e''_0, e''_1)), e''_2)) \in X \text{ and } e''_1 \in X_1 \text{ or } e''_2 \in X_2\}$$

and

$$Y = \{(f_0^{-1}(e''_0), (Y'', (f_1^{-1}(e''_1), f_2^{-1}(e''_2)))) \mid \exists X''.((X'', (e''_0, e''_1)), e''_2) \in X \text{ and } (Y'', (f_1^{-1}(e''_1), f_2^{-1}(e''_2))) \in Y'\}$$

we have $e = (Y, (e_0, (Y', (e_1, e_2)))) \in E$ and $f(e) = e'$.

We then show that f is a morphism, meaning for $e, e' \in E$:

- By definition $\lambda(e) = \lambda'(f(e))$.
- If $f(e) = f(e')$ then either (1) $e = (e_0, (e_1, e_2)) = e'$, or (2) $e = (X, (e_0, (Y, (e_1, e_2))))$, $e = (X', (e_0, (Y', (e_1, e_2))))$, and $(e''_0, (Y'', (e''_1, e''_2))) \in X$ if and only if there exists $(e''_0, (Y''', (e''_1, e''_2))) \in X'$. However, since $Y'', Y''' \in \mathsf{cause}(e''_1, e''_2)$, either $Y'' = Y'''$, or there exist $y'' \in Y''$ and $y''' \in Y'''$ such that $y'' \sharp_{12} y'''$. And in addition, there exist $e'''_0, e''''_0, Y_{y''}$, and $Y_{y'''}$ such that $(e'''_0, (Y_{y''}, y'')) \in X$ and $(e''''_0, (Y_{y'''}, y''')) \in X'$. Since X and x' must be conflict-free, $X = X'$.
- If $X \mapsto f(e)^*$, then either $e^* = (e_0, (e_1, e_2))$, $e^* = (Y, (e_0, (Y', (e_1, e_2))))$, $e^* = \underline{(e_0, (e_1, e_2))}$, or $e^* = \underline{(Y, (e_0, (Y', (e_1, e_2))))}$.

  If $e = (e_0, (e_1, e_2))$ then there exists $i \in 0, 12$ and X_i such that $X_i \mapsto_i \pi_i(e)$, and $X = \{e'' \mid \pi_i(e'') \in X_i\}$, meaning if $i = 0$, then $\{e'' \mid \pi_0(e'') \in X_i\} \mapsto (e_0, e_1)$ and therefore $\{e'' \mid \pi_0(\pi_{01}(e'')) \in X_i\} \mapsto ((e_0, e_1), e_2)$, and therefore $f(\{e'' \mid \pi_0(\pi_{01}(e'')) \in X_i\}) = X$. If $i = 12$ then there exists $j \in 1, 2$ and X_j such that $X_j \mapsto_j \pi_j(e)$, and $X = \{e'' \mid \pi_j(\pi_{12}(e'')) \in X_j\}$, and by similar logic if $j = 1$ then $\{e'' \mid \pi_1(\pi_{01}(e'')) \in X_j\} \mapsto ((e_0, e_1), e_2)$, and $f(\{e'' \mid \pi_1(\pi_{01}(e'')) \in X_j\}) = X$ and if $j = 2$ then $\{e'' \mid \pi_2(e'') \in X_j\} \mapsto ((e_0, e_1), e_2)$, and $f(\{e'' \mid \pi_2(e'') \in X_j\}) = X$.

  If $e = (Y, (e_0, (Y', (e_1, e_2))))$ and $f(e) = (Z, ((Z', (f_0(e_0), f_1(e_1))), f_2(e_2)))$ then there exists $e' = ((Z'', (e'_0, e'_1)), e'_2) \in Z$ such that $X = \{(X', e') \mid X' \subseteq X\}$ and the result follows.

$P = P' \mid 0$: Then there exists $\mathcal{E}$ and $\mathcal{E}'$ such that $[\![P']\!] = \langle \mathcal{E}', \mathsf{Init}', k' \rangle$, $[\![P]\!] = \langle \mathcal{E}, \mathsf{Init}, k \rangle$, and $\mathcal{E}$ is composed of $\mathcal{E}'$ and the empty LREBES, $\mathcal{E}_0$ as described in the parallel composition rule.

We define $f(e) = \begin{cases} e' & \text{if } e = (X, (e', *)) \\ e' & \text{if } e = (e', *) \end{cases}$

And show that $f : \mathcal{E} \to \mathcal{E}'$ is a morphism, meaning for all $e_0, e_1 \in E$:

- Clearly $\lambda(e_0) = \lambda'(f(e_0))$.

- If $f(e_0) = f(e_1)$ then there exists e such that either $e_0 = (e, *) = e_1$ or there exist X_0 and X_1 such that $e_0 = (X_0(e, *))$ and $e_1 = (X_1(e, *))$. However, by Lemma 5.3.11, we know that whenever $X'_0 \mapsto e$, X'_0 contains exactly one event, e'_0. Since that event cannot synchronise with anything from E_0, e'_0 must be in every possible cause of e, and similarly for the causes of e''_0, meaning e can only have one cause in $\mathcal{E}'||\mathcal{E}_0$, and therefore $e_0 = e_1$.
- For $X' \subseteq E'$, if $X' \mapsto' f(e_0)^*$ then $f(e_0) = e'_0$ and either $e_0 = (e'_0, *)$ or $e_0 = (X_0, (e'_0, *))$.
  If $e^*_0 = (e'_0, *)$, then $\{e \mid \exists e' \in X'.e = (X, (e', *)) \text{ or } e = (e', *)\} \mapsto e_0$. Clearly $\{e \mid \exists e' \in X'.e = (X, (e', *)) \text{ or } e = (e', *)\} = \{e \mid f(e) \in X'\}$
  If $e^*_0 = (X_0, (e'_0, *))$ then by Lemma 5.3.11 there exists e such that $X' = \{e\}$. Clearly this requires that $(e, *) \in X_0$, which means $\{(X'_0, (e, *)) \mid X'_0 \subseteq X_0\} \mapsto e_0$, and clearly $f(\{(X'_0, (e, *)) \mid X'_0 \subseteq X_0\}) = \{e\}$.
  If $e^*_0 = \underline{(e'_0, *)}$ then $\{e \mid e = (X, (e', *)) \text{ or } e = (', *) \text{ for } e' \in X'\} \mapsto e^*_0$.
  If $e^*_0 = \overline{(X_0, (e'_0, *))}$ then $\bigcup\{X'' \mid \exists X''' \in E'.X''' \mapsto' \underline{e'_0} \text{ or } \exists (e', *) \in X_0.X'' \mapsto' \underline{e'}$, and $e'' \in X''$ iff $f(e'') \in X'''\} \mapsto e^*_0$, by Lemmas 5.3.13 and 5.3.14, we know that for all $e \in X_0$, if $X'' \mapsto \underline{e}$, then $X'' \subseteq X'$, meaning $X' = \bigcup\{X'' \mid \exists X''' \in E'.X''' \mapsto' \underline{e'_0} \text{ or } \exists (e', *) \in X_0.X'' \mapsto' \underline{e'}$, and $e'' \in X''$ iff $f(e'') \in X'''\}$.
- If $f(e_0) \rhd f(e_1)^*$ then by definition, $e_0 \rhd e_1 *$.

We then prove f is bijective: We already showed above, that f is injective, and it is clear that it is also surjective.

In order to show f is an isomorphism, we therefore only need to show that f^{-1} is a morphism, meaning for $e'_0, e'_1 \in E'$:

- Again, clearly $\lambda(f^{-1}(e'_0)) = \lambda'(e'_0)$.
- If $f^{-1}(e'_0) = f^{-1}(e'_1)$ then we already know f is a bijection, so $e'_0 = e'_1$.
- For $X \subseteq E$, if $X \mapsto f^{-1}(e'_0)^*$ then $f^{-1}(e'_0) = e_0$ and either $e_0 = (e'_0, *)$ or $e_0 = (X_0, (e'_0, *))$.
  If $e^*_0 = (e'_0, *)$, then $\{e \mid (e, *) \in X \text{ or } \exists X'.(X', (e, *)) \in X\} \mapsto e'_0$.
  If $e^*_0 = (X_0, (e'_0, *))$ then by Lemma 5.3.11 we know there exists an e such that $X = \{e\}$. This means there exists X' such that $e = (X', (e', *))$ and $(e', *) \in X_0$, meaning $\{e'\} \mapsto' e'_0$.
  If $e^*_0 = \underline{(e'_0, *)}$ then either $X = \{e_0\}$, in which case by definition $\{e'_0\} \mapsto \underline{e'_0}$, or there exists an X' such that $\overline{X' \mapsto e_0}$ and $X = \{e \mid \exists e' \in X'.e = (e', *) \text{ or } e = (X'', (e', *))\}$.
  If $e^*_0 = \overline{(X_0, (e'_0, *))}$ then either $X = \{e_0\}$, in which case by definition $\{e'_0\} \mapsto \underline{e'_0}$, or $X = \bigcup\{X'' \mid f(X'') \mapsto' \underline{e'_0} \text{ or } \exists (e', *) \in X_0.f(X'') \mapsto' \underline{e'}\}$. Clearly any of these X''s can be used to fulfil the condition.
- If $f^{-1}(e'_0) \rhd f^{-1}(e'_1)^*$ then either (1) $e'_0 \rhd e'^*_1$, (2) $e'_0 = e'_1$ and $f^{-1}(e'_0) \neq f^{-1}(e'_1)^*$, (3) $e'^*_1 = e'_1$, $f^{-1}(e'_0) \neq f^{-1}(e'_1)^*$, and $f^{-1}(e'_0) \in X \mapsto \overline{f^{-1}(e'_1)}$, or (4) $e'^*_1 = e'_1$ and there exist γ_0 and γ_1 such that $\lambda(f^{-1}(e'_0)) = \mathsf{roll}\ \gamma_0$ and $\lambda(f^{-1}(e_1)) = \mathsf{roll}\ \gamma_1$.
  In case 1, the condition is trivially fulfilled. Case 2 will never occur. In case 3, as shown above, $e'_0 \in f(X) \mapsto \underline{e'_1}$, and by Lemma 5.3.12, this means $e'_0 \rhd e'_1$. In case 4, since the e'_0 and e'_1 must both have been caused by a roll at the end of a subprocess, they were either in parallel or different option in a choice, and in either case clearly $e'_0 \rhd' e'_1$.

And from Lemma 5.3.10 and the definition of Init, we see that $f(\mathsf{Init}) = \mathsf{Init}'$ and $f \circ k' = k$.

$P = X + Y$ and $P' = Y + X$: Selection works the same in Roll-CCSK as in CCSK, so this case is the same as in Proposition 5.1.15.

$P = (X + Y) + Z$ and $P' = (X + Y) + Z$: Selection works the same in Roll-CCSK as in CCSK, so this case is the same as in Proposition 5.1.15.

$P = P' + 0$: Selection works the same in Roll-CCSK as in CCSK, so this case is the same as in Proposition 5.1.15.

$P = Q \setminus A$, $P' = Q' \setminus A$, and $Q \equiv Q'$: Then $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$, and $\{\!|Q'|\!\} = \langle \mathcal{E}_{Q'}, \mathsf{Init}_{Q'}, k_{Q'} \rangle$, there exist an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}_{Q'}$ such that $f_Q(\mathsf{Init}_Q) = \mathsf{Init}_{Q'}$ and for all $e \in \mathsf{Init}_Q$, $k_Q(e) = k_{Q'}(f_Q(e))$, and by applying the restriction we get $\langle \mathcal{E}, \mathsf{Init}, k \rangle = \langle \mathcal{E}_Q \upharpoonright \rho(A \cup \overline{A}), \mathsf{Init}_Q \cap \rho(A \cup \overline{A}), k_Q \upharpoonright \rho(A \cup \overline{A}) \rangle$ and $\langle \mathcal{E}', \mathsf{Init}', k' \rangle = \langle \mathcal{E}_{Q'} \upharpoonright \rho(A \cup \overline{A}), \mathsf{Init}_{Q'} \cap \rho(A \cup \overline{A}), k_{Q'} \upharpoonright \rho(A \cup \overline{A}) \rangle$.

We now show that $e \in \rho(A \cup \overline{A})$ if and only if $f(e) \in \rho(A \cup \overline{A})$.

For any $e \in E_Q$, $\lambda_Q(e) \in A \cup \overline{A}$ iff $\lambda_{Q'}(f(e)) \in A \cup \overline{A}$. We show that for any $X \subseteq E_Q$, $X \in \mathsf{causes}(e)$ if and only if $f(X) \in \mathsf{cause}(f(e))$ by induction in the size of X.

If $X = \emptyset$ then there does not exist $x \subseteq E_Q$ such that $x \mapsto_Q e$, and by definition of an morphism, there cannot exist $x' \subseteq E_{Q'}$ such that $x' \mapsto f(e)$, meaning $\emptyset \in \mathsf{cause}(e)$. And since f is an isomorphism the same argument can be used for f^{-1}.

If X contains n events, and for all events e' and $X' \in \mathsf{cause}(e')$ such that X' contains less that n events, $X' \subseteq \rho(A \cup \overline{A})$ if and only if $f(X') \in \mathsf{cause}(f(e'))$ then whenever $x' \mapsto_{Q'} f(e)$, there exists $x \subseteq E_Q$ such that $x \mapsto_Q e$ and $f(x) \subseteq x'$, meaning there exists e'' such that $x \cap X = \{e''\}$, and $x' \cap f(X) \supseteq \{f(e'')\}$. And by induction if $X'' \mapsto e'' \in X$ then $X'' \subset X$ and therefore $f(X'') \in \mathsf{causes}(e'')$. And since X is conflict-free, $f(X)$ is conflict free. And since f is an isomorphism the same argument can be used for f^{-1}.

$P = A\langle \tilde{a}, \tilde{\gamma} \rangle$ and $P' = P_A\{^{\tilde{a},\tilde{\gamma}}/_{\tilde{b},\tilde{\delta}}\}$ where $A\langle \tilde{b}, \tilde{\delta} \rangle = P_A$: imilar to the corresponding case of Proposition 5.1.15.

□

C.19 Theorem 5.3.16

Proof. We prove this by induction on the transition $P \xrightarrow{\mu[m]} P'$:

- Suppose $P = \alpha_\gamma.Q$, $P' = \alpha_\gamma[m].Q$, $\mu = \alpha$, and $\mathsf{std}(Q)$. Then there exist $\mathcal{E}_Q$ and e_α such that $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}, k \rangle$ and $\langle \mathcal{E}, \mathsf{Init}, k \rangle$ is constructed based on this as described in the prefix rule.

  And there exist $\mathcal{E}_{Q'}$ and e'_α such that $\{\!|Q|\!\} = \langle \mathcal{E}_{Q'}, \mathsf{Init}_{Q'}, k_{Q'} \rangle$ and $\langle \mathcal{E}', \mathsf{Init}', k' \rangle$ is constructed from this a described in the past prefix rule.

  By induction, there must exist isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}_{Q'}$ and $g_Q : \mathcal{E}_{Q'} \to \mathcal{E}_Q$, and we define $f = f_Q[e_\alpha \mapsto e'_\alpha]$ and $g = g_Q[e_\alpha \mapsto e'_\alpha]$, which are clearly isomorphisms.

  Since $\mathsf{std}(Q)$, meaning $\mathsf{Init} = \emptyset$, and since no X exists such that $X \mapsto e_\alpha$, $\mathsf{Init} \xrightarrow{e_\alpha} \{e_\alpha\}$, and the rest of the conditions are satisfied.

- Suppose that $P = \alpha_\gamma[n].Q$, $P' = \alpha_\gamma[n].Q'$, $Q \xrightarrow{\mu[m]} Q'$, and $n \neq m$.

  Then there exist $\mathcal{E}_Q$ and e_α such that $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$ and $\langle \mathcal{E}, \mathsf{Init}, k \rangle$ is constructed based on this as described in the past prefix rule.

  And there exist $\mathcal{E}_{Q'}$ and e'_α such that $\{\!|'Q|\!\} = \langle \mathcal{E}_{Q'}, \mathsf{Init}_{Q'}, k_{Q'} \rangle$ and $\langle \mathcal{E}', \mathsf{Init}', k' \rangle$ is constructed from this a described in the past prefix rule.

  By induction, we get isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}_{Q'}$ and $g_Q : \mathcal{E}_{Q'} \to \mathcal{E}_Q$ and a transition $\mathsf{Init}_Q \xrightarrow{\{e\}} X_Q$ in $\mathcal{C}_{re}(\mathcal{E}_Q)$ such that $\lambda_Q(e) = \mu$, $k_{Q'}(f_Q(e)) = m$, and $f_Q(X_Q) = \mathsf{Init}_{Q'}$.

  We define $f = f_Q[e_\alpha \mapsto e'_\alpha]$ and $g = g_Q[e'_\alpha \mapsto e_\alpha]$. Since Init_Q and X_Q are conflict-free in $\mathcal{E}_Q$, $\mathsf{Init}_Q \cup \{e_\alpha\} = \mathsf{Init}$ and $X_Q \cup \{e_\alpha\} = X$ are configurations of $\mathcal{C}_{re}(\mathcal{E})$, and clearly $\mathsf{Init} \xrightarrow{\{e\}} X$.

- Suppose $P = P_0 \mid P_1$, $P' = P'_0 \mid P_1$, $P_0 \xrightarrow{\mu[m]} P'_0$, and $\mathsf{fsh}[m](P_1)$. Then there exist $\mathcal{E}_0$ and $\mathcal{E}_1$ such that for $i \in \{0, 1\}$, $\{\!|P_i|\!\} = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i \rangle$, and $\langle \mathcal{E}, \mathsf{Init}, k \rangle$ is constructed as described in the parallel composition rule.

And there exist $\mathcal{E}'_0$ and $\mathcal{E}'_1$ such that $[\![P'_0]\!] = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \rangle$, $[\![P_1]\!] = \langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1 \rangle$, and $\langle \mathcal{E}', \mathsf{Init}', k' \rangle$ is constructed as described in the parallel composition rule.

We have isomorphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$, $g_0 : \mathcal{E}'_0 \to \mathcal{E}_0$, $f_1 : \mathcal{E}_1 \to \mathcal{E}'_1$, and $g_1 : \mathcal{E}'_1 \to \mathcal{E}_1$, and there exists a transition $\mathsf{Init}_0 \xrightarrow{e_\mu} X$ in $C_{re}(\mathcal{E}_0)$ such that $\lambda_0(e_\mu) = \mu$, $k'_0(f_0(e)) = m$ and $f_0(X) = \mathsf{Init}'_0$.

We define functions

$$f'(e) = \begin{cases} (f_0(e_0), *) & \text{if } e = (e_0, *) \\ (*, f_1(e_1)) & \text{if } e = (*, e_1) \\ (f_0(e_0), f_1(e_1)) & \text{if } e = (e_0, e_1) \end{cases}$$

and

$$g'(e) = \begin{cases} (g_0(e_0), *) & \text{if } e = (e_0, *) \\ (*, g_1(e_1)) & \text{if } e = (*, e_1) \\ (g_0(e_0), g_1(e_1)) & \text{if } e = (e_0, e_1) \end{cases}$$

and our isomorphisms as:

$$f(e) = \begin{cases} (f'(X), f'(e')) & \text{if } e = (X, e') \\ f'(e) & \text{otherwise} \end{cases}$$

and

$$g(e) = \begin{cases} (g'(X), g'(e')) & \text{if } e = (X, e') \\ g'(e) & \text{otherwise} \end{cases}$$

.

It is clear that $f \circ g = I_{\mathcal{E}}$ and $g \circ f = I_{\mathcal{E}'}$

We show that $f : \mathcal{E} \to \mathcal{E}'$ is a morphism, meaning for all $e, e' \in E$:

- By definition $\lambda(e) = \lambda'(f(e))$
- If $f(e) = f(e')$ then since f_0 and f_1 are injective, $e = e'$.
- For $X' \subseteq E'$, if $X' \mapsto' f(e)^*$ then either $e^* = (Y, e_\times$ and there exists $e_{\times'} \in f(Y)$ such that $X' = \{(Y', e_{\times'}) \mid Y' \subseteq f'(Y)\}$, or $e^* \in E_\times$ and there exists X'' such that $X'' \mapsto'_\times f(e)$ and $X' = \{e' \mid (\pi'_0(e'), \pi'_1(e')) \in X''\}$, or $e^* = \underline{(Y, e_\times)}$ and $X' = \{f(e)\}$, or $e^* = \underline{(Y, e_\times)}$ and

$$X' = \bigcup \left\{ X'' \;\middle|\; \begin{array}{l} \exists i \in \{0,1\}, X_i \in E'_i \,.\, X'_i \mapsto'_i \underline{\pi'_i(f(e))} \text{ or } \exists e_{\times'} \in f(Y) \,.\, X'_i \mapsto'_i \underline{\pi'_i(e_{\times'})} \\ \text{and } e'' \in X'' \text{ iff } \pi'_i(e'') \in X'_i \end{array} \right\}$$

  or $e^* \in \underline{E_\times}$ and there exists X'' such that $X'' \mapsto'_\times \underline{f(e)}$ and $X' = \{e' \mid (\pi'_0(e'), \pi'_1(e')) \in X''\}$.

  If $e^* = (Y, e_\times)$ and there exists $e_{\times'} \in f(Y)$ such that $X' = \{(Y', e_{\times'}) \mid Y' \subseteq f(Y)\}$ then there exists an $e'_{times} \in Y$ such that $f(e'_\times) = e_{\times'}$ and clearly $\{(Y', e'_\times) \mid Y' \subseteq Y\} \mapsto (Y, e_\times)$.

  If $e^* \in E_\times$ and there exists X'' such that $X'' \mapsto'_\times f(e)$ and $X' = \{e' \mid (\pi'_0(e'), \pi'_1(e')) \in X''\}$ then by induction and since $||$ is an REBES product, there exists $X''' \subseteq E_\times$ such that $X''' \mapsto_\times e$, $f(X''') \subseteq X''$, and if $e' \in X'''$ then $f(e') \neq \bot$. This means $\{e' \mid (\pi_0(e'), \pi_1(e')) \in X'''\} \mapsto e$.

  If $e^* = \underline{(Y, e_\times)}$ and $X' = \{f(e)\}$, or $e^* = \underline{(Y, e_\times)}$ and

$$X' = \bigcup \left\{ X'' \;\middle|\; \begin{array}{l} \exists i \in \{0,1\}, X'_i \in E'_i . X'_i \mapsto'_i \underline{\pi'_i(f(e))} \text{ or } \exists e_{\times'} \in f(Y) . X'_i \mapsto'_i \underline{\pi'_i(e_{\times'})} \\ \text{and } e'' \in X'' \text{ iff } \pi'_i(e'') \in X'_i \end{array} \right\}$$

  then by induction, since $g = f^{-1}$ is a morphism for each $X_i \mapsto_i \underline{\pi_i(e)}$, $f(X_i) \subset X'_i \mapsto'_i \underline{\pi'_i(f(e))}$, and for each $X_i \mapsto_i \underline{\pi_i(e'_\times)} \in \underline{Y}$, $f(X_i) \subset X'_i \mapsto'_i \underline{\pi'_i(f(e'_\times))}$ meaning

$$\left\{ X'' \;\middle|\; \begin{array}{l} \exists i \in \{0,1\}, X_i \in E_i \,.\, X_i \mapsto_i \underline{\pi_i(e)} \text{ or } \exists e'_\times \in Y \,.\, X_i \mapsto_i \underline{\pi_i(e'_\times)} \\ \text{and } e'' \in X'' \text{ iff } \pi'_i(e'') \in X_i \end{array} \right\}$$
$$\subseteq \left\{ X'' \;\middle|\; \begin{array}{l} \exists i \in \{0,1\}, X_i \in E'_i . X'_i \mapsto'_i \underline{\pi'_i(f(e))} \text{ or } \exists e_{\times'} \in f(Y) \,.\, X'_i \mapsto'_i \underline{\pi'_i(e_{\times'})} \\ \text{and } e'' \in X'' \text{ iff } \pi'_i(e'') \in X'_i \end{array} \right\}$$

If $e^* \in \underline{E_\times}$ and there exists X'' such that $X'' \mapsto'_\times \underline{f(e)}$ and $X' = \{e' \mid (\pi'_0(e'), \pi'_1(e')) \in X''\}$ then by induction and because $||$ is an REBES product, there exists $X''' \subseteq E_\times$ such that $f(X''') \subseteq X''$ and $X''' \mapsto_\times \underline{e}$. This means $\{e' \mid (\pi_0(e'), \pi_1(e')) \in X'''\} \mapsto \underline{e}$.

– If $f(e) \rhd' f(e')^*$ then there exists $i \in \{0,1\}$ such that either $\pi'_i(f(e)) \rhd'_i \pi'_i(f(e'))^*$, or $\pi'_i(f(e)) = \pi'_i(f(e')) \neq \bot$, and $f(e) \neq f(e')$, or $f(e')^* = f(e')$, $f(e) \neq f(e')$, and $f(e) \in X \mapsto' \underline{f(e')}$, or $f(e')^* = f(e')$ and there exist γ, γ' such that $\lambda'(f(e)) = \mathsf{roll}\ \gamma$ and $\lambda'(f(e')) = \mathsf{roll}\ \gamma'$.

If $\pi'_i(f(e)) \rhd'_i \pi'_i(f(e'))^*$ then by induction $\pi_i(e) \rhd_i \pi_i(e')^*$, meaning $e^* \rhd e$.

If $\pi'_i(f(e)) = \pi'_i(f(e')) \neq \bot$, and $f(e) \neq f(e')$ then $\pi_i(e) = \pi_i(e') \neq \bot$ and $e \neq e'$, meaning $e \rhd e'^*$.

If $f(e')^* = f(e')$, $f(e) \neq f(e')$, and $f(e) \in X \mapsto' \underline{f(e')}$ then, since, by similar arguments to the previous case, $g(X) \mapsto \underline{f(e')}$, and $e \in g(X)$, $e \rhd e^*$.

If $f(e')^* = f(e')$ and there exist γ, γ' such that $\lambda'(f(e)) = \mathsf{roll}\ \gamma$ and $\lambda'(f(e')) = \mathsf{roll}\ \gamma'$, then $\lambda(e) = \mathsf{roll}\ \gamma$ and $\lambda(e') = \mathsf{roll}\ \gamma'$, meaning $e \rhd e'^*$.

By similar arguments, g is a morphism too.

We now show that there exists an $(Y, (e_\mu, *)) \in E$ such that $\{e' \mid (\pi_0(e'), \pi_1(e')) \in Y\} \subseteq \mathsf{Init}$. Since $\mathsf{Init}_0 \xrightarrow{\{e_\mu\}}$, for every $X_0 \mapsto_0 e_\mu$, $X_0 \cap \mathsf{Init}_0 = X_0 = \{e_0\}$, and if $X'_0 \mapsto_0 e_0$ then by Lemma 5.3.10, $X'_0 \mapsto_0 e_\mu$, and therefore $X'_0 \cap \mathsf{Init}_0 \neq \emptyset$. Therefore there must exist one $(Y, (e_\mu, *)) \in E$ such that $\{e' \mid (\pi_0(e'), \pi_1(e')) \in Y\} \subseteq \mathsf{Init}$. We use this $(Y, (e_\mu, *))$ as our e and show that $\mathsf{Init} \xrightarrow{\{e\}}$: Since $Y \subseteq \mathsf{Init}$, for every $X \mapsto e$, $X \cap \mathsf{Init} \neq \emptyset$. And if $e' \rhd e$ then it must that either $\pi_0(e') \rhd_0 e_\mu$, in which case $\pi_0(e') \notin \mathsf{Init}_0$, and therefore $e' \notin \mathsf{Init}$, or $\pi_0(e') = e_\mu$ and $e \neq e'$, in which case, since $\mathsf{Init}_0 \xrightarrow{\{e_\mu\}}$, $e' \notin \mathsf{Init}_0$, and therefore $e' \notin \mathsf{Init}$, or $e' \in X \mapsto \underline{e}$ and $e' \neq e$, in which case $\pi_0(e') \in X_0 \mapsto e_\mu$ or $\pi_0(e') \in X_0 \mapsto \pi_0(e'')$ for $e'' \in Y$, and by Lemmas 5.3.12 and 5.3.14, $\pi_0(e') \rhd e_\mu$, meaning $\pi_0(e') \notin \mathsf{Init}_0$, and $e' \notin I$.

We therefore have $\mathsf{Init} \xrightarrow{\{e\}} I \cup \{e\}$, and $\lambda(e) = \lambda_0(e_\mu) = \mu$ and $f \circ k' = k[e \mapsto m]$, and since $f_0(\mathsf{Init}_0 \cup \{e_\mu\}) = \mathsf{Init}'_0$ and $f_1(\mathsf{Init}_1) = \mathsf{Init}'_1$, and there only exists one $(Y, (e_\mu, *)) \in E$ such that $\{e' \mid (\pi_0(e'), \pi_1(e')) \in Y\} \subseteq \mathsf{Init}$, $f(\mathsf{Init} \cup \{e\}) = \mathsf{Init}'$.

- Suppose $P = P_0 \mid P_1$, $P' = P'_0 \mid P'_1$, $P_0 \xrightarrow{\alpha[m]} P'_0$, $P_1 \xrightarrow{\overline{\alpha}[m]} P'_1$, and $\mu = \tau$.

  Then the construction of $\langle \mathcal{E}, \mathsf{Init}, k\rangle$ and $\langle \mathcal{E}', \mathsf{Init}', k'\rangle$ and the isomorphisms are similar to the previous case. And by induction we have transitions $\mathsf{Init}_0 \xrightarrow{\{e_0\}}$ and $\mathsf{Init}_1 \xrightarrow{\{e_1\}}$ fulfilling the conditions.

  For similar reasons to the previous case there exists exactly one $(Y, (e_0, e_1)$ such that $\{e' \mid (\pi_0(e'), \pi_1(e')) \in Y\} \subseteq \mathsf{Init}$, and we use this $(Y, (e_0, e_1))$ as e, and the rest of the proof follows similarly.

- Suppose $P = P_0 + P_1$, $P' = P'_0 + P_1$, $P_0 \xrightarrow{\mu[m]} P'_0$, and $\mathsf{std}(P_1)$. Then the rule for selection is the same in Roll-CCSK as in CCSK, and the case is therefore identical to Theorem 5.1.18.

- Suppose $P = Q \setminus A$, $P' = Q' \setminus A$, $Q \xrightarrow{\mu[m]} Q'$, and $\mu \notin A \cup \overline{A}$. Then there exist $\mathcal{E}_Q$ and $\mathcal{E}'_Q$ such that $[\![Q]\!] = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k\rangle$, $[\![Q']\!] = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q\rangle$, and $\mathcal{E}$ and $\mathcal{E}'$ are constructed from $\mathcal{E}_Q$ and $\mathcal{E}'_Q$ as described in the restriction rule, and there exist isomorphisms $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and $g : \mathcal{E}'_Q \to \mathcal{E}_Q$ and a transition $\mathsf{Init}_Q \xrightarrow{\{e_Q\}}$ where $\lambda_Q(e_Q) = \mu$, $f_Q \circ k'_Q = k_Q[e_q \mapsto m]$, and $f_Q(\mathsf{Init}_Q \cup \{e_Q\}) = \mathsf{Init}'_Q$.

  Since there exists a standard process P'' such that $P'' \to^* P$, there cannot exist $e' \in \mathsf{Init}$ such that $\lambda(e') \in A \cup \overline{A}$ or for all $x \in \mathsf{cause}(e')$, there exists $e'' \in x$ such that $\lambda(e'') \in A \cup \overline{A}$, meaning $\mathsf{Init} \cap \rho(A \cup \overline{A}) = \mathsf{Init}$, and, since $e_Q \in \rho(A \cup \overline{A})$, $\mathsf{Init} \xrightarrow{e_Q}$.

- Suppose $P = Q[f]$, $P' = Q'[f]$, $Q \xrightarrow{\nu[m]} Q'$, and $f'(\nu) = \mu$. Then the rule for functions is the same in Roll-CCSK as in CCSK, and the case is therefore identical to Theorem 5.1.18.

- Suppose $P \equiv Q$, $P' \equiv Q'$, and $Q \xrightarrow{\mu[m]} Q'$. Then the result follows from induction and Proposition 5.3.15. □

C.20 Theorem 5.3.17

Proof. We prove this by induction on P.

- Suppose $P = 0$. Then $E = \emptyset$, and no transitions exist in $C_{br}(\mathcal{E})$.
- Suppose $P = \mathsf{roll}\ \gamma$. Then there does not exist $e \in E$ such that $\lambda(e) = \mu$.
- Suppose $P = \mathsf{rolling}\ \gamma$. Then there does not exist $e \in E$ such that $\lambda(e) = \mu$.
- Suppose $P = \alpha_\gamma.P''$. Then $\{e_\alpha\} \mapsto e'$ for all $e' \in E \setminus \{e_\alpha\}$ such that $\lambda(e) = \mu$, meaning by definition $e = e_\alpha$. In addition, by Lemma 5.3.9, whenever $e' \in \mathsf{Init}$, $\lambda(e') \in \{\mathsf{roll}\ \gamma', \mathsf{start\ roll}\ \gamma'\}$ meaning $\mathsf{std}(P)$. This means we get $P \xrightarrow{\alpha[m]} \alpha[m].P''$ for some fresh m, and the isomorphisms are similar to this case in the proof of Theorem 5.1.18.
- Suppose $P = \alpha[n].P''$ and $\{\!|P''|\!\} = \langle \mathcal{E}'', \mathsf{Init}'', k'' \rangle$. Then $e_\alpha \in \mathsf{Init}$, and clearly $\mathsf{Init}'' \xrightarrow{e} X''$, meaning there exists a a key m and a transition $P'' \xrightarrow{\lambda(e)[m]} P'''$, such that $\{\!|P'''|\!\} = \langle \mathcal{E}''', \mathsf{Init}''', k''' \rangle$ and there exist isomorphisms $f'' : \mathcal{E}'' \to \mathcal{E}'''$ and $g'' : \mathcal{E}''' \to \mathcal{E}''$ such that $k'''(f''(e)) = m$ and $f''(X'') = \mathsf{Init}'''$. If $m \neq n$, then $P \xrightarrow{\lambda(e)[m]} \alpha[m].P'''$. Otherwise, we can chose a fresh m and still get a transition. We define our isomorphisms as $f = f''[e_\alpha \mapsto e'_\alpha]$ and $g = g''[e'_\alpha \mapsto e_\alpha]$ and the rest follows.
- Suppose $P = P_0 + P_1$. Then the proof is similar to the same case in CCSK, as the choice semantics are the same.
- Suppose $P = P_0 \mid P_1$, $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $C_{br}(\mathcal{E}_0) = (E_0, F_0, \mathsf{C}_0, \to_0)$, $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$, and $C_{br}(\mathcal{E}_1) = (E_1, F_1, \mathsf{C}_1, \to_1)$. Then either $e = (Y, (e_0, *))$, $e = (Y, (*, e_1))$, or $e = (Y, (e_0, e_1))$.

  If $e = (Y, (e_0, *))$, then whenever $X'_0 \mapsto_0 e_0$, there exists $e' \in Y$ such that $\pi_0(e') \in X_0$ and $\{e'\} \mapsto e$. And whenever $\pi_0(e') \rhd_0 \pi_0(e)$, we get $e' \rhd e$. This means Init_0 is conflict-free, $\pi_0(X)$ is conflict-free, and $\mathsf{Init}_0 \xrightarrow{e_0}_0 \pi_0(X)$. There therefore exists a a key m and a transition $P_0 \xrightarrow{\lambda_0(e_0)[m]} P'_0$, such that $\{\!|P'_0|\!\} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \rangle$ and there exist isomorphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$ and $g_0 : \mathcal{E}'_0 \to \mathcal{E}_0$ such that $k'_0(f_0(e_0)) = m$ and $f_0(\pi_0(X)) = \mathsf{Init}'_0$.

  We chose an m, which is fresh for P_1, and we get $P \xrightarrow{\lambda_0(e_0)[m]} P'_0 \mid P_1$. We define our isomorphisms similarly to the corresponding case in Theorem 5.3.16, and the proof of them being isomorphisms is similar.

  If $e = (Y, (*, e_1))$, the argument is similar.

  If $e = (Y, (e_0, e_1))$, then for $i \in \{0, 1\}$, whenever $X'_i \mapsto_i e_i$, there exists $e' \in Y$ such that $\pi_i(e') \in X'_i$ and $\{e'\} \mapsto e$. And whenever $\pi_i(e') \sharp_i \pi_i(e)$, we get $e' \sharp e$. This means Init_i is conflict-free, $\pi_i(X)$ is conflict-free, and $\mathsf{Init}_i \xrightarrow{e_0}_i \pi_i(X)$. There therefore exists a a key m_i and a transition $P_i \xrightarrow{\lambda_i(e_i)[m_i]} P'_i$, such that $\{\!|P'_i|\!\} = \langle \mathcal{E}'_i, \mathsf{Init}'_i, k'_i \rangle$ and there exist isomorphisms $f_i : \mathcal{E}_0 \to \mathcal{E}'_i$ and $g_i : \mathcal{E}'_i \to \mathcal{E}_i$ such that $k'_i(f_i(e_i)) = m_i$ and $f_i(\pi_i(X)) = \mathsf{Init}'_0$.

  We say that $m_0 = m_1$ is a fresh m, and then since $\lambda_0(e_0) = \overline{\lambda_1(e_1)}$ and $\lambda(e) = \tau$, we get $P \xrightarrow{\lambda(e)[m]} P'_0 \mid P'_1$. We define our isomorphisms similarly to the corresponding case in Theorem 5.3.16, and the proof of them being isomorphism is similar to that case and the rest follows.
- Suppose $P = P'' \setminus A$, $\{\!|P''|\!\} = \langle \mathcal{E}'', \mathsf{Init}, k \rangle$, and $C_{br}(\mathcal{E}'') = (E'', F'', \mathsf{C}'', \to'')$. Then $\lambda(e) \notin A \cup \overline{A}$ and there exists at least one $Y \in \mathsf{cause}(e)$ such that if $e' \in Y$ then $\lambda(e') \notin (A \cup \overline{A})$. And since P is reachable, for all $e' \in \mathsf{Init}$, $\lambda(e') \notin (A \cup \overline{A})$. We therefore know $\mathsf{Init}'' = \mathsf{Init} \xrightarrow{e}{}'' X$, meaning there exists a key

n and a transition $P'' \xrightarrow{\lambda''(e)} P'''$ such that $\{\!|P'''|\!\} = \langle \mathcal{E}''', \mathsf{Init}''', k''' \rangle$, and there exist isomorphisms $f' : \mathcal{E}'' \to \mathcal{E}'''$ and $f' : \mathcal{E}''' \to \mathcal{E}''$ such that $f' \circ k''' = [e \mapsto n]$ and $f'(X) = \mathsf{Init}'''$.

This means $P \xrightarrow{\lambda''(e)} P''' \setminus A$ and the morphisms $f \upharpoonright E$ and $g \upharpoonright E''' \cap \rho(A \cup \overline{A})$ clearly fulfil the remaining conditions.

- Suppose $P = P''[f]$, $\{\!|P''|\!\} = \langle \mathcal{E}'', \mathsf{Init}, k \rangle$. Then the case is similar to the corresponding case of Theorem 5.1.19. □

C.21 Theorem 5.3.25

Proof. We prove this through induction on the derivation of $P \overset{\mathsf{roll}\ n}{\rightsquigarrow} P'$:

(act ROLL): Suppose $P = \alpha_\gamma[n].R$, $R \overset{\mathsf{roll}\ n}{\rightsquigarrow} R'$, and $P' = \alpha_\gamma.R_{\star\{n' | n \leq_P n'\}}$, with $\{\!|R|\!\} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$ and $\left\{\!\left| R_{\star\{n' | n \leq_P n'\}} \right|\!\right\} = \langle \mathcal{E}_{R'}, \mathsf{Init}_{R'}, k_{R'} \rangle$. Then $\{n' \mid n \leq_P n'\}$ is n and all n's for which $\beta_{\gamma'}[n']$ occurs in P. It is clear from the semantic rules that this means $\mathsf{Init}' = \{e \mid \lambda(e) \in \{\mathsf{start\ roll}\ \gamma' \mid \mathsf{rolling}\ \gamma' \text{ occurs in } P \text{ and } \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[m] \text{ occurs in } P\}$ and there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$.

In addition we have $e, e_r \in E$ such that $\lambda(e) = \alpha$, $k(e) = n$, $\lambda(e_r) = \mathsf{roll}\ \gamma$, $\mathsf{N}(e_r) = \mathsf{roll}\ n$, and $\{e_r\} \mapsto \underline{e}$. By Lemmas 5.3.9, 5.3.13, and 5.3.14. we then get $\mathsf{Init} \xrightarrow{\{e_r\}} X_0 \xrightarrow{\{\underline{e_0}\}} X_1 \cdots \xrightarrow{\{\underline{e_n}\}} X_{n+1} \xrightarrow{\{\underline{e_r}\}} X_{\mathsf{done}}$ fulfilling the conditions.

(par ROLL): Suppose $P = Q \mid R$, $P' = (Q \mid R)_{\star\{n' | n \leq_P n'\}}$, $Q \overset{\mathsf{roll}\ \gamma}{\rightsquigarrow} Q'$, $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$, $\{\!|R|\!\} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $\langle \mathcal{E}, \mathsf{Init}, k \rangle$ is constructed from $\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$ and $\langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$ as described in the semantics, $\left\{\!\left| Q_{\star\{n' | n \leq_P n'\}} \right|\!\right\} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$, $\left\{\!\left| R_{\star\{n' | n \leq_P n'\}} \right|\!\right\} = \langle \mathcal{E}'_R, \mathsf{Init}'_R, k'_R \rangle$, and $\langle \mathcal{E}', \mathsf{Init}', k' \rangle$ is constructed from $\langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and $\langle \mathcal{E}'_R, \mathsf{Init}'_R, k'_R \rangle$ as described in the semantics.

It is clear from the semantics that there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$.

In addition, by induction we have $\mathsf{Init}_Q \xrightarrow{\{e_r\}} X_{(0,Q)} \xrightarrow{\{\underline{e_0}\}} X_{(1,Q)} \cdots \xrightarrow{\{\underline{e_m}\}} X_{(m+1,Q)} \xrightarrow{\{\underline{e_r}\}} X_{(done,Q)}$, $\{e_0, e_1, \ldots e_n\} = \{e' \mid n \leq_Q k_Q(e')\}$, and there exists an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $f(X_{(done,Q)}) = \mathsf{Init}'_Q$.

From this we get that $(e_r, *) \in E$, and for each $X \mapsto (e_r, *)$, we have an $X_Q \mapsto e_r$ such that $X = \{e \in E \mid \pi_Q(e) \in X_Q\}$, and therefore $X \cap \mathsf{Init} \neq \emptyset$. Additionally, if $e \triangleright (e_r, *)$, then either $\pi_Q(e) \triangleright e_r$, or $\lambda(e) = \mathsf{roll}\ \gamma'$, meaning $e \notin \mathsf{Init}$. We therefore get $\mathsf{Init} \xrightarrow{\{(e_r,*)\}}$. Since by Lemma 5.3.20 $\nexists e' \in I.\lambda(e') = \mathsf{roll}\ \gamma'$, we get that by Lemma 5.3.21 for e_i, $0 \leq i \leq n$, whenever $X_i \mapsto e_i$, either $X_i = \{e_i\}$, or $e_r \in X_i$, meaning for any $e \in E$ such that $\pi_Q(e) \in \{e_0, e_1, \ldots e_n\}$, whenever $X \mapsto e$, either $X = \{e\}$ or $(e_r, *) \in X$. The rest follows from Lemmas 5.3.14 and 5.3.22.

(prop ROLL Key 1): Suppose $P = \beta'_\gamma[m].R$, $P' = \beta'_\gamma[m].R'$, $m' \neq n$, $\{\!|R|\!\} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $\{\!|R'|\!\} = \langle \mathcal{E}_{R'}, \mathsf{Init}_{R'}, k_{R'} \rangle$, and $R \overset{\mathsf{roll}\ n}{\rightsquigarrow} R'$. Then by induction $\mathsf{Init}_R \xrightarrow{\{e_r\}} X_0 \xrightarrow{\{\underline{e_0}\}} X_1 \cdots \xrightarrow{\{\underline{e_n}\}} X_{n+1} \xrightarrow{\{\underline{e_r}\}} X_{\mathsf{done}}$, and there exists an isomorphism $f_R : \mathcal{E}_R \to \mathcal{E}_{R'}$ fulfilling the conditions. Then it is clear from the semantics that the result holds using the isomorphism $f = f_R[e_\alpha \mapsto e'_\alpha]$.

(prop ROLL Key 2): Suppose $P = \beta'_\gamma.R$, $P' = \beta'_\gamma.R'$, $\{\!|R|\!\} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $\{\!|R'|\!\} = \langle \mathcal{E}_{R'}, \mathsf{Init}_{R'}, k_{R'} \rangle$, and $R \overset{\mathsf{roll}\ n}{\rightsquigarrow} R'$. Then $\mathsf{Init}_R \xrightarrow{\{e_r\}} X_0 \xrightarrow{\{\underline{e_0}\}} X_1 \cdots \xrightarrow{\{\underline{e_n}\}} X_{n+1} \xrightarrow{\{\underline{e_r}\}} X_{\mathsf{done}}$, and there exists an isomorphism $f_R : \mathcal{E}_R \to \mathcal{E}_{R'}$ such that $f_R(X_d) = \mathsf{Init}_{R'}$. Then it is clear from the semantics that the result holds using the isomorphism $f = f_R[e_\alpha \mapsto e'_\alpha]$.

(prop ROLL Key 3): Suppose $P = P_0 + P_1$, $P' = P'_0 + P_1$, $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$ $\{\!|P'_0|\!\} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \rangle$, and $P_0 \overset{\mathsf{roll}\ n}{\rightsquigarrow} P'_0$. Then by induction $\mathsf{Init}_0 \xrightarrow{\{e_{(r,0)}\}} X_{(0,0)} \xrightarrow{\{e_{(0,0)}\}} X_{(1,0)} \cdots \xrightarrow{\{e_{(n,0)}\}} X_{(n+1,0)} \xrightarrow{\{e_{(r,0)}\}} X_{(done,0)}$, and there exists an isomorphism $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$ fulfilling the conditions. Then, since P is consistent, $\mathsf{std}(P_1)$, and therefore $\{0\} \times \mathsf{Init}_0 \xrightarrow{\{(0,e_{(r,0)})\}} \{0\} \times X_{(0,0)} \xrightarrow{\{(0,e_{(0,0)})\}} \{0\} \times X_{(1,0)} \cdots \xrightarrow{\{(0,e_{(n,0)})\}} \{0\} \times X_{(n+1,0)} \xrightarrow{\{(0,e_{(r,0)})\}} \{0\} \times X_{(done,0)}$, and it is straightforward to check that the rest holds.

(prop ROLL Key 4): Suppose $P = R \setminus A$, $P' = R' \setminus A$, $\{\!|R|\!\} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $\{\!|R'|\!\} = \langle \mathcal{E}_{R'}, \mathsf{Init}_{R'}, k_{R'} \rangle$, and $R \overset{\mathsf{roll}\ n}{\rightsquigarrow} R'$. Then by induction $\mathsf{Init}_R \xrightarrow{\{e_r\}} X_0 \xrightarrow{\{e_0\}} X_1 \cdots \xrightarrow{\{e_n\}} X_{n+1} \xrightarrow{\{e_r\}} X_{\mathsf{done}}$, and there exists an isomorphism $f_R : \mathcal{E}_R \to \mathcal{E}_{R'}$ such that $f_R(X_d) = \mathsf{Init}_{R'}$. Then, since P is consistent, if $\alpha_\gamma[n]$ occurs in R, $\alpha \notin A \cup \overline{A}$, and by Theorem 5.3.16, whenever $e \in \mathsf{Init}_R$, there exists $X \in \mathsf{cause}(e)$ such that $X \subseteq \rho(A \cup \overline{A})$, and the result follows.

(prop ROLL Key 5): Suppose $P = R[f]$, $P' = R'[f]$, $\{\!|R|\!\} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $\{\!|R'|\!\} = \langle \mathcal{E}_{R'}, \mathsf{Init}_{R'}, k_{R'} \rangle$, $\gamma' \neq \gamma$, and $R \overset{\mathsf{roll}\ n}{\rightsquigarrow} R'$. Then by induction $\mathsf{Init}_R \xrightarrow{\{e_r\}} X_0 \xrightarrow{\{e_0\}} X_1 \cdots \xrightarrow{\{e_n\}} X_{n+1} \xrightarrow{\{e_r\}} X_{\mathsf{done}}$, and there exists an isomorphism $f_R : \mathcal{E}_R \to \mathcal{E}_{R'}$ fulfilling the conditions, and the result follows.

(prop ROLL Key 6): Suppose $P \equiv Q$, $Q \overset{\mathsf{roll}\ n}{\rightsquigarrow} Q'$, and $Q' \equiv P$. Then the result follows from Proposition 5.3.15.

□

Appendix D

Proofs from Chapter 6

D.1 Proposition 6.1.5

Proof. 1. We perform induction on $P \xrightarrow{\mu[n]} Q$:

(a) Suppose $P = a(x).P'$, $\mu = a(b)$, $\mathsf{std}(P')$, $Q = a(b)[n].Q'$, and $Q' = Q[x := b_{[n]}]$. Then, since $x \notin n(Q')$, $Q \overset{a(b)}{\rightsquigarrow} P$.

(b) Suppose $P = \overline{a}(x).P'$, $\mu = \overline{a}(x)$, $\mathsf{std}(P')$, $Q = \overline{a}(x)[n].P'$. Then clearly $Q \overset{\overline{a}(x)}{\rightsquigarrow} P$.

(c) Suppose $P = \alpha[m].P'$, $P' \xrightarrow{\mu[n]} Q'$, $Q = \alpha[m].Q'$, $n \neq m$, and if $\mu = \overline{a}(x)$ then $x \notin n(\alpha)$. Then by induction $Q' \overset{\mu[n]}{\underset{P}{\rightsquigarrow}}{}'$, and clearly $Q \overset{\mu[n]}{\rightsquigarrow} P$.

(d) Suppose $P = P_0|P_1$, $P_0 \xrightarrow{\mu[n]} Q_0$, $\mathsf{fsh}[n](P_1)$, $Q = Q_0|P_1$, and if $\mu = \overline{a}(x)$ then $x \notin \mathsf{fn}(P_1)$. Then by induction, $Q_0 \overset{\mu[n]}{\rightsquigarrow} P_0$, and therefore $Q \overset{\mu[n]}{\rightsquigarrow} P$.

(e) Suppose $P = P_0|P_1$, $P_0 \xrightarrow{a(x)[n]} Q_0$, $P_1 \xrightarrow{\overline{a}(x)[n]} Q_1$, $\mu = \tau$, and $Q = (\nu x)(Q_0|Q_1)$. Then by induction $Q_0 \overset{a(x)}{\rightsquigarrow} P_0$ and $Q_1 \overset{\overline{a}(x)}{\rightsquigarrow} P_1$, meaning clearly $Q \overset{\mu[n]}{\rightsquigarrow} P$.

(f) Suppose $P = P_0 + P_1$, $P_0 \xrightarrow{\mu[n]} Q_0$, $\mathsf{std}(P_1)$, and $Q = Q_0 + P_1$. Then by induction $Q_0 \overset{\mu[n]}{\rightsquigarrow} P_0$, meaning $Q \overset{\mu[n]}{\rightsquigarrow} P$.

(g) Suppose $P = (\nu x)P'$, $P' \xrightarrow{\mu[n]} Q'$, $x \notin n(\mu)$, and $Q = (\nu x)Q'$. Then by induction $Q' \overset{\mu[n]}{\rightsquigarrow} P'$, and we get $Q \overset{\mu[n]}{\rightsquigarrow} P$.

(h) Suppose $P \equiv P'$, $P' \xrightarrow{\mu[n]} Q'$, and $Q \equiv Q'$. Then by induction $Q' \overset{\mu[n]}{\rightsquigarrow} P'$, and therefore $Q \overset{\mu[n]}{\rightsquigarrow} P$.

2. We prove this by induction on $P \overset{\mu[n]}{\rightsquigarrow} Q$:

(a) Suppose $P = a(b)[n].P'$, $\mu = a(b)$, $\mathsf{std}(P')$, $x \notin n(P')$, $Q' = P'[b_{[n]} := x]$, and $Q = a(x).Q'$. Then clearly $Q \xrightarrow{\mu[n]} P$.

(b) Suppose $P = \overline{a}(x)[n].P'$, $\mu = \overline{a}(x)$, $\mathsf{std}(P')$, $Q = \overline{a}(x).P'$. Then clearly $Q \xrightarrow{\overline{a}(x)} P$.

(c) Suppose $P = \alpha[m].P'$, $P' \overset{\mu[n]}{\rightsquigarrow} Q'$, $m \neq n$, and $Q = \alpha[n].Q'$. Then by induction, $Q' \xrightarrow{\mu[n]} P'$, and since P is forwards-reachable, if $\mu = \overline{a}(x)$ then $x \notin n(\alpha)$. This means $Q \xrightarrow{\mu[n]} P$.

(d) Suppose $P = P_0|P_1$, $P_0 \overset{\mu[n]}{\rightsquigarrow} Q_0$, $\mathsf{fsh}[n](P_1)$, $Q = Q_0|P_1$, and if $\mu = \overline{a}(x)$ then $x \notin \mathsf{fn}(P_1)$. Then by induction $Q_0 \xrightarrow{\mu[n]} P_0$, and clearly $Q \xrightarrow{\mu[n]} P$.

(e) Suppose $P = (\nu x)(P_0|P_1)$, $\mu = \tau$, $P_0 \overset{a(x)[n]}{\rightsquigarrow} Q_0$, $P_1 \overset{\overline{a}(x)[n]}{\rightsquigarrow} Q_1$, and $Q = Q_0|Q_1$. Then by induction $Q_0 \xrightarrow{a(x)} P_0$ and $Q_1 \xrightarrow{\overline{a}(x)} P_1$, meaning clearly $Q \xrightarrow{\mu[n]} P$.

(f) Suppose $P = P_0 + P_1$, $P_0 \overset{\mu[n]}{\rightsquigarrow} Q_0$, $\mathsf{std}(P_1)$, and $Q = Q_0 + P_1$. Then by induction $Q_0 \xrightarrow{\mu[n]} P_0$, meaning $Q \xrightarrow{\mu[n]} P$.

(g) Suppose $P = (\nu x)P'$, $P' \overset{\mu[n]}{\rightsquigarrow} Q'$, $x \notin n(\mu)$, and $Q = (\nu x)Q'$. Then by induction $Q' \xrightarrow{\mu[n]} P'$, and we get $Q \xrightarrow{\mu[n]} P$.

(h) Suppose $P \equiv P'$, $P' \overset{\mu[n]}{\rightsquigarrow} Q'$, and $Q \equiv Q'$. Then by induction $Q' \xrightarrow{\mu[n]} P'$, and therefore $Q \xrightarrow{\mu[n]} P$.

□

D.2 Proposition 6.1.6

Proof. We use structural induction on P to prove both these at once:

1. Suppose $P = 0$ or $P = \alpha.P'$. Then P cannot do any backwards transitions.

2. Suppose $P = \alpha[n'].P'$. Then either $\mathsf{std}(P')$ and $n = m = n'$, or $Q = a(b)[n'].Q'$, $R = a(b)[n'].R'$, $P' \overset{\mu[m]}{\rightsquigarrow} Q'$, and $P' \overset{\mu'[n]}{\rightsquigarrow} R'$, meaning by induction there exists S' such that $Q' \overset{\mu'[n]}{\rightsquigarrow} S'$ and $R' \overset{\mu[m]}{\rightsquigarrow} S'$. We say that $S = \alpha[n].S'$, and the property holds.

3. Suppose $P = P_0 + P_1$, then either $\mathsf{std}(P_0)$, $P_1 \overset{\mu[m]}{\rightsquigarrow} Q_1$, $P \overset{\mu'[n]}{\rightsquigarrow} R_1$, $Q = P_0 + Q_1$, and $R = P_0 + R_1$, or $\mathsf{std}(P_1)$, $P_0 \overset{\mu[m]}{\rightsquigarrow} Q_0$, $P \overset{\mu'[n]}{\rightsquigarrow} R_0$, $Q = Q_0 + P_1$, and $R = R_0 + P_1$. In the first case, by induction there exists an S_1 such that $Q_1 \overset{\mu'[n]}{\rightsquigarrow} S_1$ and $R_1 \overset{\mu[m]}{\rightsquigarrow} S_1$, and we define $S = P_0 + S_1$, and the property holds. The second case is similar.

4. Suppose $P = (\nu x)P'$. Then either (1) $P' \overset{\mu[m]}{\rightsquigarrow} Q'$ and $x \notin n(\mu)$ and $Q = (\nu x)Q'$ or (2) $P' = P_0|P_1$, $P_i \overset{a(x)[m]}{\rightsquigarrow} Q_i$, $P_{1-i} \overset{\overline{a}(x)[m]}{\rightsquigarrow} Q_{i-1}$, $\mu = \tau$, and $Q = Q_0|Q_1$, and either (a) $P' \overset{\mu'[n]}{\rightsquigarrow} R'$ and $x \notin n(\mu')$ and $R = (\nu x)R'$ or (b) $P' = P_0|P_1$, $P_i \overset{a(x)[n]}{\rightsquigarrow} R_i$, $P_{1-i} \overset{\overline{a}(x)[n]}{\rightsquigarrow} R_{i-1}$, $\mu' = \tau$, and $R = R_0|R_1$.

   In case 1a, by induction there exists S' such that $Q' \overset{\mu'[n]}{\rightsquigarrow} S'$ and $R' \overset{\mu[m]}{\rightsquigarrow} S'$, and we define $S = (\nu x)S'$, and the theorem holds.

   In case 1b, there exists P_j such that $P_j \overset{\mu[m]}{\rightsquigarrow} Q_j$, and $\mathsf{fsh}[m](P_{1-j})$, and if $\mu = \overline{a}(x)$ then $x \notin \mathsf{fn}(P_1)$. If $j = i$ then by induction there exists an S_i such that $Q_j \overset{\mu'[n]}{\rightsquigarrow} S_i$ and $R_i \overset{a(x)[m]}{\rightsquigarrow} S_i$, and we define $S = S_i|R_{1-i}$, and the property holds. If $I = 1 - j$, the argument is similar.

   Case 2a is similar to case 1b.

   Case 2b cannot occur because we cannot have more than one past action outputting the same name according to Lemma 6.1.4.

5. Suppose $P = P_0|P_1$. Then there exists an i such that either $P_i \overset{\mu[m]}{\rightsquigarrow} Q_i$ and $P_i \overset{\mu'[n]}{\rightsquigarrow} R_i$ and $Q = Q_i|P_{1-i}$ and $R = R_i|P_{1-i}$, or $P_i \overset{\mu[m]}{\rightsquigarrow} Q_i$ and $P_{1-i} \overset{\mu'[n]}{\rightsquigarrow} R_{1-i}$ and $Q = Q_i|P_{1-i}$ and $R = P_i|R_{1-i}$.

In the first case, there exists S_i such that $Q_i \overset{\mu'[n]}{\rightsquigarrow} S_i$ and $R_i \overset{\mu[m]}{\rightsquigarrow} S_i$, and we define $S = S_i|P_{1-i}$ and the property holds.

If the second case we define $S = Q_i|R_{1-i}$, and the property holds.

□

D.3 Proposition 6.1.7

Proof. We prove this by structural induction:

1. Suppose $P = 0$ or $P = \alpha.P'$. Then P cannot do any reverse transitions.

2. Suppose $P = \alpha[n].P'$. Then either $\mathsf{std}(P')$, meaning $\mu = \mu' = \alpha$, $n = m$, and $Q \equiv R$, or $P' \overset{\mu[m]}{\rightsquigarrow} Q'$, $P' \overset{\mu'[m]}{\rightsquigarrow} R'$, $Q = \alpha[n].Q'$, and $R = \alpha[n].R'$, and the result follows from induction.

3. Suppose $P = P_0 + P_1$. Then the result follows from induction.

4. Suppose $P = (\nu x)P'$. Then either (1) $P' \overset{\mu[m]}{\rightsquigarrow} Q'$ and $x \notin n(\mu)$ and $Q = (\nu x)Q'$ or (2) $P' = P_0|P_1$, $P_i \overset{a(x)[m]}{\rightsquigarrow} Q_i$, $P_{1-i} \overset{\overline{a}(x)[m]}{\rightsquigarrow} Q_{i-1}$, $\mu = \tau$, and $Q = Q_0|Q_1$, and either (a) $P' \overset{\mu'[m]}{\rightsquigarrow} R'$ and $x \notin n(\mu')$ and $R = (\nu x)R'$ or (b) $P' = P_0|P_1$, $P_i \overset{a(x)[m]}{\rightsquigarrow} R_i$, $P_{1-i} \overset{\overline{a}(x)[m]}{\rightsquigarrow} R_{i-1}$, $\mu' = \tau$, and $R = R_0|R_1$.

   In case 1a the result follows from induction.

   In case 1b P_j such that $P_j \overset{\mu[m]}{\rightsquigarrow} Q_j$, and $\mathsf{fsh}[m](P_{1-j})$, contradicting $P_{1-j} \xrightarrow{\alpha[m]} R_{1-j}$. Meaning this case cannot occur.

   Similar for case 2a.

   Case 2b follows from induction.

5. Suppose $P = P_0|P_1$. Then there exists an i such that either $P_i \overset{\mu[m]}{\rightsquigarrow} Q_i$ and $P_i \overset{\mu'[m]}{\rightsquigarrow} R_i$ and $Q = Q_i|P_{1-i}$ and $R = R_i|P_{1-i}$, or $P_i \overset{\mu[m]}{\rightsquigarrow} Q_i$ and $P_{1-i} \overset{\mu'[m]}{\rightsquigarrow} R_{1-i}$ and $Q = Q_i|P_{1-i}$ and $R = P_i|R_{1-i}$.

   In the first case the result follows from induction. In the second case $P_i \overset{\mu[m]}{\rightsquigarrow} Q_i$ requires $\mathsf{fsh}[m](P_{1-i})$, which contradicts $P_{1-i} \overset{\mu'[m]}{\rightsquigarrow} R_{1-i}$, meaning this case cannot occur.

□

D.4 Proposition D.4

Proof. We prove both simultaneously by structural induction in P.

- Suppose $P = 0$. Then P cannot perform any actions, and the results are trivial.
- Suppose $P = a(x).P'$. Then $\mu = a(b)$ for some b, $Q = a(b)[n].P'[x := b]$, $\mu' = a(c)$ for some c, $R = a(c)[m].P'[x := c]$, and since no subsequent action can change b and c, $b = c$, and the results hold.
- Suppose $P = \overline{a}(x).P'$. Then $\mu = \overline{a}(x)$ $Q = \overline{a}(x)[n].P'$, $\mu' = \overline{a}(x)$, $R = \overline{a}(x)[m].P'\overline{a}(x)$, and the results hold.
- Suppose $P = \alpha[n].P'$. Then the results follow from induction.
- Suppose $P = P_0 + P_1$. Then the results follow from induction, and the fact that both actions must use the same branch of the choice.

- Suppose $P = (\nu x)P'$. Then the result follows from induction.

- Suppose $P = P_0|P_1$. Then there exists an i such that either (1) $P_i \xrightarrow{\mu[m]} Q_i$ and $Q = Q_i|P_{1-i}$ or (2) $P_i \xrightarrow{a(x)[m]} Q_i$, $P_{1-i} \xrightarrow{\overline{a}(x)[m]} Q_{i-1}$, $\mu = \tau$, and $Q = Q_0|Q_1$ and there exists a j such that either (a) $P_j \xrightarrow{\mu'[n]} R_j$ and $R = R_j|P_{1-j}$ or (b) $P_j \xrightarrow{b(y)[n]} R_j$, $P_{1-j} \xrightarrow{\overline{b}(y)[n]} R_{j-1}$, $\mu' = \tau$, and $R = R_0|R_1$

  In Case 1a, either $i = j$ or $i \neq j$. If $i = j$ then the result follows from induction. If $i \neq j$, since we require $\mathsf{fsh}[m](P_{1-i})$ and $\mathsf{fsh}[n](P_{j-i})$, some where in $R \to^* T$ and $Q \to^* T$ respectively, $m \neq n$. For similar reasons, we cannot have that $\mu = \overline{a}(x)$ and $x \in n(\mu')$ or vice versa, meaning $Q \xrightarrow{\mu'[n]} S$ and $R \xrightarrow{\mu[m]} S$ and $S \to^*$.

  In Case 2b, suppose $i = j$. Then by induction, there exists S_i such that $Q_i \xrightarrow{b(y)[n]} S_i$, requiring $\mathsf{fsh}[n](R_{1-i})$ meaning $n \neq m$, and $R_i \xrightarrow{\mu[m]} S_i$, meaning we can define $S = S_i|R_{1-i}$, and requiring $\mathsf{fsh}[n](R_{1-i})$, and the result holds for that. If $i \neq j$ then the result is similar.

  Case 2a is similar.

  Case 2b follows from induction.

- Suppose $P = !P'$. Then $!P'|P' \xrightarrow{\mu[m]} Q$, and $!P'|P' \xrightarrow{\mu'[n]} R$, and this falls under the previous case.

□

D.5 Proposition 6.2.6

Proof. 1. We prove this by induction in $\mathbf{H} \vdash P \xrightarrow[u]{\alpha} \mathbf{H}' \vdash Q$:

[SCOPE] In this case $P = (\nu x)P'$ and $Q = (\nu x)Q'$, $x \notin n(\alpha)$, and by induction $\mathbf{H}' \vdash Q' \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P'$. From rule [SCOPE^{-1}] we therefore get $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.

[PAR$_i$] In this case $P = P_0|P_1$ and $Q = Q_0|Q_1$, $P_{1-i} = Q_{1-i}$, in $\alpha = \overline{a}(n)$ then $n \notin \mathsf{fn}(P_{1-i})$, and by induction $([\check{i}]\overline{H}', [\check{i}]\underline{H}', [\check{i}]H') \vdash Q_i \overset{\alpha}{\underset{u}{\rightsquigarrow}} ([\check{i}]\overline{H}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i$, meaning according to rule [PAR$_i^{-1}$], $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.

[COM$_i$] In this case $P = P_0|P_1$ and $Q = Q_0|Q_1$, $n \notin \mathsf{fn}(P_j)$, $\overline{H} = \overline{H}'$, $\underline{H} = \underline{H}'$, and $H' = H \cup \{((\alpha_0, \alpha_1), (0v_0, 1v_1))\}$. By induction and Lemma 6.2.5, we get transitions $([\check{i}]\overline{H}', [\check{i}]\underline{H}', [\check{i}]H') \vdash Q_i \overset{\alpha_i}{\underset{v_i}{\rightsquigarrow}} ([\check{i}]\overline{H}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i$ and $([\check{j}]\overline{H}', [\check{j}]\underline{H}', [\check{j}]H') \vdash Q_j \overset{\alpha_j}{\underset{v_j}{\rightsquigarrow}} ([\check{j}]\overline{H}, [\check{j}]\underline{H}, [\check{j}]H) \vdash P_j$. This means according to [COM^{-1}], $\mathbf{H}' \vdash Q \overset{\tau}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.

[STR] In this case $Q \equiv Q'$,$\mathbf{H}' \vdash Q' \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P'$, and $P' \equiv P$, and by rule [STR^{-1}], $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.

[OUT] In this Case $P = \sum_{i \in I} \alpha_i.P_i$, $Q = P_j$ and $\alpha = \overline{a}(n) = \alpha_j$ for some $j \in I$, and by [OUT^{-1}], $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.

[IN] Similar to [OUT].

2. Similar to previous.

□

D.6 Proposition 6.2.10

Proof. This proof is similar to Theorem 14 of [52]. Either we have a path l such that $u_i = l0u'_i$ and $u_{1-i} = l1u'_{1-i}$, or we have $\mu_i = \tau$.

In the first case, if $Q|R$ is the parallel composition at location l, then $([l\check{}0]\overline{H}, [l\check{}0]\underline{H}, [l\check{}0]H) \vdash Q \overset{\alpha_i}{\underset{u'_i}{\rightarrowtail}}$ and $([l\check{}1]\overline{H}, [l\check{}1]\underline{H}, [l\check{}1]H) \vdash Q \overset{\alpha_{1-i}}{\underset{u'_{1-i}}{\rightarrowtail}}$ and there does not exist n such that $D_i(n) = u_{1-i}$ or there does not exist n such that $D_{1-i}(n) = u_i$, and by $[\mathrm{PAR}_i]$ and $[\mathrm{PAR}_i^-1]$, this means $\mathbf{H}_0 \vdash P_0 \xrightarrow[u_1]{\alpha_1} \mathbf{H}'_0 \vdash P'$ and $\mathbf{H}_1 \vdash P_1 \xrightarrow[u_0]{\alpha_0} \mathbf{H}'_1 \vdash P'$ and by Lemma 15 of [52], $\mathbf{H}'_0 = \mathbf{H}'_1$. This applies to reverse transitions too.

If $\mu_i = \tau$ then $u_i = l_i\langle l_{i0}[P_0][Q_0], l_{i1}[P_1][Q_1]\rangle$ and we can split the transition into $l_{i0}[P_0][Q_0]$ and $l_{i1}[P_1][Q_1]$ at the parallel composition on location l and use similar logic. □

D.7 Lemma 6.2.12

Proof. If $\overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}}$ and $\overset{\alpha_1}{\underset{u_1,D_1}{\rightsquigarrow}}$ are not independent, then either u_0 and u_1 are not independent, or there exists n such that $D_i(n) = u_{1-i}$.

If u_0 and u_1 are not independent, then $u_0 = u_1$ and $\alpha_0 = \alpha_1$, $D_0 = D_1$, and $\mathbf{H}_0 \vdash P_0 \equiv \mathbf{H}_1 \vdash P_1$ follows from that.

If u_0 and u_1 are independent and exists n such that $D_i(n) = u_{1-i}$, then by $[\mathrm{PAR}_i^{-1}]$, we cannot have $\mathbf{H} \vdash P \overset{\alpha_{1-i}}{\underset{u_{1-i},D_{1-i}}{\rightsquigarrow}} \mathbf{H}_{1-i} \vdash P_{1-i}$. □

D.8 Lemma 6.3.4

Proof. We prove this by structural induction on P:

- Assume $P = 0$. Then $P' = P[x_1 := a_1][x_2 := a_2] \dots [x_k := a_k] = 0$ and $S(P[x_1 := a_1][x_2 := a_2] \dots [x_k := a_k], P, [n], x_1) = 0$.
- Assume $P = b(c).Q$ Then either $P' = d(e).Q'$, or $P' = d(e)[m].Q'$, for some d, e, m. We then get 4 cases: either $b = x_1$, $c = x_1$, $b = c = x_1$, or $b \neq x_1$ and $c \neq x_1$.

  Assume $b = x_1$ and $c \neq x_1$. Then $d = a_1$, and $S(P', P, [n], x_1) = d[n](c).S(Q', Q, [n], x_1)$, and the result follows from induction.

  Assume $c = x_1$ and $b \neq x_1$. Then, since c is bound, $P[x_1 := a_1] = P[x_1 := a_{1_{[n]}}] = P$, and the result follows.

  Assume $b = c = x_1$. Then $d = a_1$ and $Q[x_1 := a_1][x_2 := a_2] \dots [x_k := a_k] = Q[x_2 := a_2] \dots [x_k := a_k] = Q'$, and the result follows.
- Assume $P = \overline{b}(c).Q$. This is similar to the previous case.
- Assume $P = \sum_{i \in I} P_i$. Then the result follows trivially from induction.
- Assume $P = P_0|P_1$. Then either $P' = P'_0|P'_1$, or $P_0 \equiv !P_1$ and $P' = P'_0$.

  If $P' = P'_0|P'_1$ then the result follows trivially from induction.

  If $P_0 = !P_1$ and $P' = P_0$, then $P'' = S\left(!P'_0, P_0, [n], x\right)|S\left(P'_0, P_1, [n], x\right)$, and the result follows from induction.
- Assume $P = (\nu b)Q$. Then $P' = (\nu c)Q'$ and either $b = x_1$ or $b \neq x_1$.

  If $b = x_1$, then $P[x_1 := a_1] = P[x_1 := a_{1_{[n]}}] = P$.

  If $b \neq x_1$, then the result follows from induction.

- Assume $P = !Q$. Then either $P' = !Q'$, or $P' = P'_0|P'_1$.
  If $P' = !Q'$, the result follows trivially from induction.
  Otherwise the case is similar to the second case on parallel composition.

□

D.9 Theorem 6.3.8

Proof. We first show that if there exists a location u such that $\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$, then $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \xrightarrow[E]{\mu[m]} (\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$ by induction in the size of $\overline{H} \cup \underline{H} \cup H$ and the structure of P:
Assume $\mathbf{H} = (\emptyset, \emptyset, \emptyset)$. Then $E(\mathbf{H} \vdash P, P) = P$.

- Assume $P = a(x).Q$. Then $\mu = a(b)$, $u = [P][Q[x := b]]$, and $\mathbf{H}' \vdash P' = (\emptyset, \{(a(b), u)\}, \emptyset) \vdash Q[x := b]$. We then by Lemma 6.3.4 get $E(\mathbf{H}' \vdash P', P') = a(x)\,[[P][Q[x := b]]]\,.Q[x := b_{[[P][Q[x:=b]]]}]$, and the rest of the case follows naturally.
- Assume $P = \overline{a}(x).Q$. This case is similar to the previous.
- Assume $P = P_0|P_1$. Then either $u = iu'$, or $u = \langle 0u_0, 1u_1 \rangle$.
  If $u = 0u'$, then $(\emptyset, \emptyset, \emptyset) \vdash P_0 \xrightarrow[u']{\mu} \mathbf{H}'_0 \vdash P'_0$, $\mathbf{H}' \vdash P' = (0\overline{H'}_0, 0\underline{H'}_0, 0H'_0) \vdash P'_0|P_1$, and if $\mu = \overline{a}(b)$ then $b \notin \mathsf{fn}(P_1)$. By induction, $P_0 \xrightarrow{\mu[u']} E(\mathbf{H}'_0 \vdash P'_0, P_0)$, and therefore $P_0|P_1 \xrightarrow{\mu[u]} E(\mathbf{H}'_0 \vdash P'_0, P_0)|P_1 = E((0\overline{H'}_0, 0\underline{H'}_0, 0H'_0) \vdash P'_0|P_1, P'_0|P_1)$.
  If $u = 1u'$, the case is similar to $u = 0u'$.
  If $u = \langle 0u_0, 1u_1 \rangle$, then $(\emptyset, \emptyset, \emptyset) \vdash P_i \xrightarrow[u_i]{a(b)} \mathbf{H}'_i \vdash P'_i$ and $(\emptyset, \emptyset, \emptyset) \vdash P_{1-i} \xrightarrow[u_{1-i}]{\overline{a}(b)} \mathbf{H}_{1-i} \vdash P'_{1-i}$ for some $i \in \{0, 1\}$ and $b \notin \mathsf{fn}(P_i)$ and $\mathbf{H}' \vdash P' = (\emptyset, \emptyset, \{(a(b), \overline{a}(b), u)\}) \vdash P'_0|P'_1$. By induction, $E((\emptyset, \emptyset, \emptyset) \vdash P_i, P_i) \xrightarrow{a(b)[u_i]} E(\mathbf{H}'_i \vdash P'_i, P'_i)$ and $E((\emptyset, \emptyset, \emptyset) \vdash P_{1-i}, P_{1-i}) \xrightarrow{\overline{a}(b)[u_{1-i}]} E(\mathbf{H}'_{1-i} \vdash P'_{1-i}, P'_{i-1})$. Therefore $P_0|P_1 \xrightarrow{\tau[u]} E((\emptyset, \emptyset, \{(a(b), \overline{a}(b), \langle 0u_0, 1u_1 \rangle, m)\}) \vdash (\nu b)(P'_0|P'_1), (\nu b)(P'_0|P'_1))$
  $= (\nu b)E((\emptyset, \emptyset, \{(a(b), \overline{a}(b), \langle 0u_0, 1u_1 \rangle, m)\}) \vdash (P'_0|P'_1), (P'_0|P'_1))$.
- Assume $P = (\nu x)Q$. Then $(\emptyset, \emptyset, \emptyset) \vdash Q \xrightarrow[u]{\mu} \mathbf{H}' \vdash Q'$, $x \notin n(\mu)$, and $P' = (\nu x)Q'$. We then get by induction $Q \xrightarrow{\mu[u]} E(\mathbf{H}' \vdash Q', Q')$, and therefore $(\nu x)Q \xrightarrow{\mu[u]} (\nu x)E(\mathbf{H}' \vdash Q', Q') = E(\mathbf{H}' \vdash (\nu x)Q', (\nu x)Q')$.
- Assume $P = !Q$. Then $(\emptyset, \emptyset, \emptyset) \vdash !Q|Q \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$, and the rest follows from the parallel case.

If for any $(\mu', u') \in \overline{H} \cup \underline{H} \cup H$, if there exists a location u such that $\mathbf{H} - (\mu', u') \vdash P \xrightarrow[u]{\mu} \mathbf{H}'' \vdash P'$, then there exists a key m, such that $E(\mathbf{H} - (\mu', u') \vdash P, P) \xrightarrow{\mu[u]} E(\mathbf{H}'' \vdash P', P')$, then E only adds past actions and unused choice branches to the process, both of which one can easily propagate the action past.

We then show that if $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \xrightarrow{\mu[n]} Q$ then there exists a location, u and a πIH process, $\mathbf{H}' \vdash P'$, such that $\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$ and $P'' \equiv E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$. We again do this by induction on the number of extrusions in $\overline{H} \cup \underline{H} \cup H$, and the structure of P.

Assume $\overline{H} \cup \underline{H} \cup H = \emptyset$. Then $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) = P$. Since we are only proving operational correspondence up to structural congruence, we can discount any rules employing that.

- Assume $P = a(x).Q$. Then $\mu = a(b)$, we say that $u = [P][Q[x := b]]$, and $E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P') = a(b)[[P][Q[x := b]]].Q[x := b_{[[P][Q[x:=b]]]}]$. We see that, $\mathbf{H}' \vdash P' = (\emptyset, \{(a(b), [P][Q[x := b]], [P][Q[x := b]])\}, \emptyset) \vdash Q[x := b]$, and by Lemma 6.3.4 the result follows.

- Assume $P = \overline{a}(b).Q$. This case is similar to the previous.

- Assume $P = P_0|P_1$. Then $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) = Q_0 \mid Q_1$ and either $Q_i \xrightarrow{\mu[m]} Q_i'$ and $Q = Q_i'|Q_{1-i}$, or $Q_0 \xrightarrow{\alpha_0[m]} Q_0'$ and $Q_1 \xrightarrow{\alpha_1[m]} Q_1'$ and $\alpha_i = a(b)$ and $\alpha_{1-i} = \overline{a}(b)$ and $\mu = \tau$ and $Q = (\nu b)(Q_0'|Q_1')$.

  If $Q_i \xrightarrow{\mu[m]} Q_i'$ and $Q = Q_i'|Q_{1-i}$ then by induction there exists u_i such that $Q_i \xrightarrow[u_i]{\mu} P_i''$ and there exists $\mathbf{H}_i' \vdash P_i'$, such that $[\check{i}]\mathbf{H} \vdash P_i \overset{\mu}{\underset{u_i}{\rightsquigarrow}} \mathbf{H}_i' \vdash P_i'$ and $P_i'' \equiv E(\mathsf{lcopy}(\mathbf{H}_i') \vdash P_i', P_i')$. We therefore get $(\emptyset,\emptyset,\emptyset) \vdash P \xrightarrow[iu_i]{\mu} (i\overline{H_i}, i\underline{H_i}, iH_i) \vdash P_0'|P_1'$ with $P_{1-i}' = P_{1-i}$.

  If $Q_0 \xrightarrow{\alpha_0[m]} Q_0'$ and $Q_1 \xrightarrow{\alpha_1[m]} Q_1'$ and $\alpha_i = a(b)$ and $\alpha_{1-i} = \overline{a}(b)$ and $\mu = \tau$ and $Q = (\nu b)(Q_0'|Q_1')$, then by induction there exist u_i such that $Q_i \xrightarrow[u_i]{\alpha_i} P_i''$ and there exists $\mathbf{H}_i' \vdash P_i'$, such that $[\check{i}]\mathbf{H} \vdash P_i \overset{\mu}{\underset{u_i}{\rightsquigarrow}} \mathbf{H}_i' \vdash P_i'$ and $P_i'' \equiv E(\mathsf{lcopy}(\mathbf{H}_i') \vdash P_i', P_i')$ for $i \in \{0,1\}$. We say $u = \langle u_0, u_1 \rangle$ and get $(\emptyset,\emptyset,\emptyset) \vdash P \xrightarrow[u]{\mu} (\emptyset,\emptyset,((\alpha_0,\alpha_1),u,u) \vdash (\nu b)(P_0''|P_1''))$

- Assume $P = (\nu x)Q$. Then $x \notin n(\mu)$, $E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P') = (\nu x)Q'$ and $Q \xrightarrow{\mu[u]} Q'$. We therefore get $P' = (\nu x)Q''$, and by induction $\mathbf{H} \vdash Q \xrightarrow[u]{\mu} \mathbf{H}' \vdash Q'$, and therefore $\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$.

- Assume $P = !Q$. Then the transition must involve structural congruence, $!Q|Q \xrightarrow{\mu[u]} P'''$ for $P''' \equiv P''$, and the rest follows from the parallel case.

If for any $(\mu',u') \in \overline{H} \cup \underline{H} \cup H$, if there exists a key m, such that $E(\mathbf{H}-(\mu',u') \vdash P, P) \xrightarrow{\mu[m]} E(\mathbf{H}'' \vdash P', P')$, then there exists a location u such that $\mathbf{H} - (\mu',u') \vdash P \xrightarrow[u]{\mu} \mathbf{H}'' \vdash P'$, then having more past extrusions does not stop $\mathbf{H} - (\mu',u') \vdash P$ from performing any forwards actions and having more past actions does not allow $E(\mathbf{H} - (\mu',u') \vdash P, P)$ to perform additional forwards actions.

We then need to prove that if there exists a location u such that $\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash P'$, then there exists a key m, such that $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \overset{\mu[m]}{\rightsquigarrow} E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$.

This follows naturally from the above properties, and Propositions 6.1.5 and 6.2.6.

We finally need to prove that if $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \overset{\mu[n]}{\rightsquigarrow} Q$ then there exists a location, u, such that $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \overset{\mu[u]}{\rightsquigarrow} P''$, and there exists a πIH process, $\mathbf{H}' \vdash P'$, such that $\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash P'$ and $P'' \equiv E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$.

As we have proven the above properties, and Propositions 6.1.5, and 6.2.6, we only need to prove that there exists a πIH-calculus process $\mathbf{H}' \vdash P'$, such that $P'' \equiv E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$. Since none of the transition rules - forwards or reverse - in the πIK-calculus can create unguarded choice from guarded choice, and E only generates πI-calculus processes with guarded choice, we know P'' has guarded choice.

If P'' is a standard process, then $\mathbf{H}' \vdash P' = (\emptyset,\emptyset,\emptyset) \vdash P''$. Otherwise, by Theorems 6.1.9 and 6.2.14, P'' must be forwards reachable from a standard process P''' such that $P''' \equiv E((\emptyset,\emptyset,\emptyset) \vdash P''', P''')$, and by the above properties, $\mathbf{H}' \vdash P'$ exists.

□

D.10 Proposition 6.4.9

Proof. We say that $\mathcal{E} = (E, F, \mapsto, \sharp, \triangleright, \lambda, \mathsf{Act})$ and $\mathcal{E}' = (E', F', \mapsto', \sharp', \triangleright', \lambda', \mathsf{Act}')$ and do a case analysis on the Structural congruence rules:

$P = P_0|P_1$ **and** $P' = P_1|P_0$**:** In this case, products are unique up to isomorphism, and the function $f(e) = \begin{cases}(e_1, e_0) & \text{if } e = (e_0, e_1)\\ (e_1, *) & \text{if } e = (*, e_1)\\ (*, e_0) & \text{if } e = (e_0, *)\end{cases}$ clearly fulfils the conditions other conditions and remains a morphism after the enablings and preventions describing the link dependencies are added to the product.

$P = P_0|(P_1|P_2)$ **and** $P' = (P_0|P_1)|P_2$**:** We know that products are associative up to isomorphism, and the function $f((e_0, (e_1, e_2)) = ((e_0, e_1), e_2)$ clearly fulfils the other conditions and remains a morphism after the enablings and preventions describing the link dependencies are added to the product.

$P = P'|0$**:** If $f((e, *)) = e$, then this clearly holds.

$P = P_0 + P_1$ **and** $P' = P_1 + P_0$**:** Coproducts are unique up to isomorphism, and $f(i, e) = (1 - i, e)$ clearly fulfils the other conditions.

$P = P_0 + (P_1 + P_2)$ **and** $P' = (P_0 + P_1) + P_2$**:** We know that coproducts are associative up to isomorphism, and $f((e_0, (e_1, e_2))) = ((e_0, e_1), e_2)$ clearly fulfils the other conditions.

$P = P' + 0$**:** Clearly $f(0, e) = e$ is an isomorphism, $\mathsf{Init} = \{0\} \times \mathsf{Init}'$, and $k(0, e) = k'(e)$.

$P = !_{\mathbf{x}}Q$ **and** $P' = !_{\mathbf{x}\setminus\{x_0,\dots,x_k\}}Q|Q\{{}^{x_0,\dots,x_k}/_{a_0,\dots,a_k}\}$ **and** $\{x_0, \dots, x_k\} \subseteq \mathbf{x}$ **and** $\mathsf{bn}(P) = \{a_0, \dots, a_k\}$**:** This case follows from our requirement that all free and bound names are distinct.

□

D.11 Theorem 6.4.11

Proof. Let $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ and $\mathcal{E}' = (E', F', \mapsto', cf', \rhd', \lambda', \mathsf{Act}')$. We prove the theorem by induction on $P \xrightarrow{\mu[m]} P'$:

1. Suppose $P = a(x).Q$, $P' = a(x)[m].Q[x := b_{[n]}]$, $\mathsf{std}(Q)$, and $\mu = a(b)$. Then for all $n \in (\mathcal{N} \setminus \mathsf{sbn}(Q)) = (\mathcal{N} \setminus \mathsf{sbn}(Q[x := b_{[n]}]))$, we have $\{\!|Q[x := n]|\!\} = \langle \mathcal{E}_n, \mathsf{Init}_n, k_n\rangle$, $\left\{\!\left|Q[x := b_{[n]}][b_{[n]} := n]\right|\!\right\} = \langle \mathcal{E}'_n, \mathsf{Init}'_n, k'_n\rangle$, and an isomorphism $f_n : \mathcal{E}_n \to \mathcal{E}'_n$. We define our isomorphism

$$f((n, e_n)) = \begin{cases}(n, f_n(e_n)) & \text{if } e_n \in E_n\\ (n, e'_n) & \text{for } \{e'_n\} = \{e' \mid (n, e') \in E' \text{ and } e' \notin E'_n\} \text{ otherwise}\end{cases}$$

Since all bound names are different from all other bound and free names, $b \notin \mathsf{bn}(Q)$, and therefore there exists an $e \in E$ such that $\lambda(e) = a(b)$, and for all $e' \in E$ either $e' = e$, $e' \sharp e$, or $\{e\} \mapsto e'$. We therefore get $\mathsf{Init} = \emptyset \xrightarrow{\{e\}} X$ and $f(X) = \mathsf{Init}'$, and the rest of the conditions fulfilled.

2. Suppose $P = \overline{a}(x).Q$, $P' = \overline{a}(x)[m].Q$, $\mu = \overline{a}(x)$, and $\mathsf{std}(Q)$. This case is similar to the previous, without the choice of substitutions.

3. Suppose $P = \alpha[n].Q$, $P' = \alpha[n].Q'$, $Q \xrightarrow{\mu[m]} Q'$, $m \neq n$, and if $\mu = \overline{a}(x)$ then $x \notin n(\alpha)$. Then let $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q\rangle$ and $\{\!|Q'|\!\} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q\rangle$. We have an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and a transition $\mathsf{Init}_Q \xrightarrow{e_Q} X_Q$ such that $\lambda_Q(e_Q) = \mu$, $f \circ k'_Q = k_Q[e_Q \mapsto m]$, and $f(X_Q) = \mathsf{Init}'_Q$. We define our isomorphism

$$f((n, e_n)) = \begin{cases}(n, f_n(e_n)) & \text{if } e_n \in E_n\\ (n, e'_n) & \text{for } \{e'_n\} = \{e' \mid (n, e') \in E' \text{ and } e' \notin E'_n\} \text{ otherwise}\end{cases}$$

and $e = (x, e_Q)$ if $\alpha = a(x)$, and

$$f(e') = \begin{cases}f_Q(e') & \text{if } e' \in E_Q\\ e'' & \text{for } \{e''\} = \{e''' \mid e''' \in E' \text{ and } e''' \notin E'_Q\} \text{ otherwise}\end{cases}$$

and $e = e_Q$ if $\alpha = \overline{a}(x)$. These clearly fulfil the conditions.

4. Suppose $P = P_0|P_1$, $P' = P'_0|P_1$, $P_0 \xrightarrow{\mu[m]} P'_0$, $\mathsf{fsh}[n](P_1)$, and if $\mu = \overline{a}(x)$ then $x \notin \mathsf{fn}(P_1)$. Then let $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $\{\!|P'_0|\!\} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$, and $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$. We then have an isomorphism $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$ and transition $\mathsf{Init}_0 \xrightarrow{e_0} X_0$ such that $\lambda_0(e_0) = \mu$, $f_0 \circ k'_0 = k_0[e_0 \mapsto m]$, and $f_0(X_0) = \mathsf{Init}'_0$. We define our isomorphism

$$f(e') = \begin{cases} (f_0(e'_0), *) & \text{if } e' = (e'_0, *) \\ (*, e'_1) & \text{if } e' = (*, e'_1) \\ (f_0(e'_0), e'_1) & \text{if } e' = (e'_0, e'_1) \end{cases}$$

and $e = (e_0, *)$. Since $\mathsf{sbn}(P_0) = \mathsf{sbn}(P'_0)$ this is an isomorphism, and since all free and bound names are different, $no(\mu) \cap \mathsf{sbn}(P_1) = \emptyset$, implying $\mathsf{Init} \xrightarrow{e}$. The other conditions are clearly fulfilled.

5. Suppose $P = P_0|P_1$, $P' = (\nu x)(P'_0|P_1)$, $\mu = \tau$, $P_0 \xrightarrow{a(x)[m]} P'_0$, and $P_1 \xrightarrow{\overline{a}(x)[m]} P'_1$. Then let $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $\{\!|P'_0|\!\} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$, $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$, and $\{\!|P'_1|\!\} = \langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1\rangle$. Then we have isomorphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$ and $f_1 : \mathcal{E}_1 \to \mathcal{E}'_1$ and transitions $\mathsf{Init}_0 \xrightarrow{e_0} X_0$ and $\mathsf{Init}_1 \xrightarrow{e_1} X_1$ such that $\lambda_0(e_0) = a(x)$, $\lambda_1(e_1) = \overline{a}(x)$, $f_0 \circ k'_0 = k_0[e_0 \mapsto m]$, $f_1 \circ k'_1 = k_1[e_1 \mapsto m]$, $f_0(X_0) = \mathsf{Init}'_0$, and $f_1(X_1) = \mathsf{Init}'_1$. We then define our isomorphism

$$f(e') = \begin{cases} (f_0(e'_0), *) & \text{if } e' = (e'_0, *) \\ (*, f_1(e'_1)) & \text{if } e' = (*, e'_1) \\ (f_0(e'_0), f_1(e'_1)) & \text{if } e' = (e'_0, e'_1) \end{cases}$$

and $e = (e_0, e_1)$. Since $\mathsf{sbn}(P_0) = \mathsf{sbn}(P'_0)$ and the existence of $(f_0(e_0), f_1(e_1)) \in \mathsf{Init}'$ and $a(x)[m]$ and $\overline{a}(x)[m]$ prevents (νx) from affecting $\mathcal{E}'$, f is an isomorphism, and since $no(\tau) = \emptyset$, we have a transition $\mathsf{Init} \xrightarrow{e}$. The other conditions are clearly fulfilled.

6. Suppose $P = P_0 + P_1$, $P' = P'_0 + P_1$, $P_0 \xrightarrow{\mu[m]} P'_0$, and $\mathsf{std}(P_1)$. Then let $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0\rangle$, $\{\!|P'_0|\!\} = \langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0\rangle$, and $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1\rangle$. We then have an isomorphism $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$ and transition $\mathsf{Init}_0 \xrightarrow{e_0} X_0$ such that $\lambda_0(e_0) = \mu$, $f_0 \circ k'_0 = k_0[e_0 \mapsto m]$, and $f_0(X_0) = \mathsf{Init}'_0$. We define out isomorphism

$$f((i, e_i)) = \begin{cases} (0, f_0(e_0)) & \text{if i=0} \\ (1, e_1) & \text{if i=1} \end{cases}$$

and $e = (0, e_0)$. Isomorphism is preserved by the coproduct, and the remaining conditions are clearly fulfilled.

7. Suppose $P = (\nu x)Q$, $P' = (\nu x)Q'$, $Q \xrightarrow{\mu[m]} Q'$, and $x \notin n(\mu)$. Then let $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q\rangle$ and $\{\!|Q'|\!\} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q\rangle$. We have an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and a transition $\mathsf{Init}_Q \xrightarrow{e_Q} X_Q$ such that $\lambda_Q(e_Q) = \mu$, $f \circ k'_Q = k_Q[e_Q \mapsto m]$, and $f(X_Q) = \mathsf{Init}'_Q$. Either there exist past actions $b(a)[m]$ and $\overline{b}(a)[m]$ in P which are not guarded by a restriction (νa) in P or not. If such $b(a)[m]$ and $\overline{b}(a)[m]$ exist, then $\langle \mathcal{E}, \mathsf{Init}, k\rangle = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q\rangle$ and $\langle \mathcal{E}', \mathsf{Init}', k'\rangle = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q\rangle$, and the rest follows trivially. Otherwise restriction preserves morphisms, and clearly does not affect $e = e_Q$.

8. Suppose $P \equiv Q$, $P' \equiv Q'$, and $Q \xrightarrow{\mu[m]} Q'$. Then the result follows from induction and Proposition 6.4.9

□

D.12 Theorem 6.4.12

Proof. We prove this by structural induction on P:

- Suppose $P = 0$. Then $E = \emptyset$, and no transition $\mathsf{Init} \xrightarrow{\{e\}} X$ exists.
- Suppose $P = \overline{a}(x).Q$. Let $\{\!|Q|\!\}_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$, $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$, and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$. Then there exists e such that $E \setminus E_Q = \{e\}$, and for all $e' \in E$, if $e' \neq e$ then $\{e\} \mapsto e'$. Therefore this is the only possible e such that $\mathsf{Init} \xrightarrow{\{e\}}$. Additionally we have $\lambda(e) = \overline{a}(x)$ and $P \xrightarrow{\overline{a}(x)[m]} \overline{a}(x)[m].Q$ for any key m, and the rest of the case is straightforward.
- Suppose $P = a(x).Q$. Then there must exist $b \in \mathcal{N} \setminus \mathsf{sbn}(P)$ such that $\lambda(e) = a(b)$, and for all $e' \in E$ either $e = e'$, $e \sharp e'$, or $\{e\} \mapsto e'$. There then exists a transition $P \xrightarrow{a(b)[m]} a(b)[m].Q[x := b_{[m]}]$ and the rest of the case is straightforward.
- Suppose $P = \overline{a}(x)[n].Q$. Let $\{\!|Q|\!\}_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$, $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$, and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$. Then $\mathsf{Init} \xrightarrow{\{e\}} X$ implies $\mathsf{Init}_Q \xrightarrow{\{e\}} X \cap E_Q$. We therefore have a transition $Q \xrightarrow{\mu[m]} Q'$ such that $\{\!|Q'|\!\}_{\mathcal{N}} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $\lambda_Q(e) = \mu$, $f_Q \circ k'_Q = k_Q[e \mapsto m]$, and $f_Q(X \cap E_Q) = \mathsf{Init}'_Q$. This gives us a transition $P \xrightarrow{\mu[m]} \overline{a}(x)[n].Q'$ and the rest of the case is straightforward.
- Suppose $P = a(x)[n].Q$. This case is a combination of the previous two.
- Suppose $P = Q + R$. Let $\{\!|Q|\!\}_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$, $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$, $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $\mathcal{E}_R = (E_R, F_R, \mapsto_R, \sharp_R, \rhd_R, \lambda_R, \mathsf{Act}_R)$, and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$. Either $e = (0, e_Q)$ or $e = (1, e_R)$. In the first case we get a transition transition $Q \xrightarrow{\mu[m]} Q'$ such that $\{\!|Q'|\!\}_{\mathcal{N}} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $\lambda_Q(e_Q) = \mu$, $f_Q \circ k'_Q = k_Q[e_Q \mapsto m]$, and $f_Q(\{e'_Q | (0, e'_Q) \in X\}) = \mathsf{Init}'_Q$. We therefore define $f(e) = \begin{cases} (0, f_Q(e_Q)) & \text{if } e = (0, e_Q) \\ e & \text{otherwise} \end{cases}$ and the rest of the case is straightforward. If $e = (1, e_R)$, the proof is similar.
- Suppose $P = Q | R$. Let $\{\!|Q|\!\}_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$, $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$, $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $\mathcal{E}_R = (E_R, F_R, \mapsto_R, \sharp_R, \rhd_R, \lambda_R, \mathsf{Act}_R)$, and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$. Either $e = (e_Q, *)$, $e = (*, e_R)$, or $e = (e_Q, e_R)$. If $e = (e_Q, *)$ then we have a transition $Q \xrightarrow{\mu[m]} Q'$ such that $\{\!|Q'|\!\}_{\mathcal{N}} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $\lambda_Q(e_Q) = \mu$, $f_Q \circ k'_Q = k_Q[e_Q \mapsto m]$, and $f_Q(\{e'_Q | (e'_Q, *) \in X \text{ or } (e'_Q, e'_R) \in X\}) = \mathsf{Init}'_Q$. We therefore get $P \xrightarrow{\mu[m]} Q' | R$ so long as $\mathsf{fsh}[m](R)$, and if not we can do the same with a different m. We can then define

$$f(e') = \begin{cases} (f_0(e'_0), *) & \text{if } e' = (e'_0, *) \\ (*, e'_1) & \text{if } e' = (*, e'_1) \\ (f_0(e'_0), e'_1) & \text{if } e' = (e'_0, e'_1) \end{cases}$$

and the rest of the case is straightforward. If $e = (*, e_R)$, the case is similar. If $e = (e_Q, e_R)$, then we have transition $Q \xrightarrow{\alpha[m]} Q'$ such that $\{\!|Q'|\!\}_{\mathcal{N}} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $\lambda_Q(e_Q) = \alpha$, $f_Q \circ k'_Q = k_Q[e_Q \mapsto m]$, and $f_Q(\{e'_Q | (e'_Q, *) \in X \text{ or } (e'_Q, e'_R) \in X\}) = \mathsf{Init}'_Q$, and transition $R \xrightarrow{\alpha'[m]} R'$ such that $\{\!|R'|\!\}_{\mathcal{N}} = \langle \mathcal{E}'_R, \mathsf{Init}'_R, k'_R \rangle$ and isomorphism $f_R : \mathcal{E}_R \to \mathcal{E}'_R$ such that $\lambda_R(e_R) = \alpha'$, $f_R \circ k'_R = k_R[e_R \mapsto m]$, and $f_R(\{e'_R | (*, e'_R) \in X \text{ or } (e'_Q, e'_R) \in X\}) = \mathsf{Init}'_R$ and there exist names a, b such that either $\alpha = a(b)$ and $\alpha' = \overline{a}(b)$ or $\alpha' = a(b)$ and $\alpha = \overline{a}(b)$. We therefore get a transition $P \xrightarrow{\tau[m]} (\nu b)(Q' | R')$ and define

$$f(e) = \begin{cases} (f_0(e'_0), *) & \text{if } e = (e'_0, *) \\ (*, f_1(e'_1)) & \text{if } e = (*, e'_1) \\ (f_0(e'_0), f_1(e'_1)) & \text{if } e = (e'_0, e'_1) \end{cases}$$

and the rest follows.

- Suppose $P = (\nu a)Q$. Let $[\![Q]\!]_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$, $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$, and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$. Then either there exist past actions $b(a)[m]$ and $\overline{b}(a)[m]$ in Q which are not guarded by a restriction (νa) in Q or not. If such $b(a)[m]$ and $\overline{b}(a)[m]$ do exist, then they must be in parallel, and therefore there exists an event $e' \in E \setminus \mathsf{Init}$ such that $\lambda(e) = \overline{b}(a)$, and for all other events $e'' \in E$, if $\lambda(e')$ outputs a then $e' = e$, and if $a \in no(\lambda(e'))$ then $\{e'\} \mapsto e''$. Additionally there exists $e''' \in \mathsf{Init}$ such that $e''' \sharp e'$ and $\lambda(e''') = \tau$. We therefore get that $a \notin n(e)$. Additionally $\mathsf{Init}_Q = \mathsf{Init} \xrightarrow{\{e\}} X$ and by induction we have a transition $Q \xrightarrow{\mu[m]} Q'$ such that $[\![Q']\!]_{\mathcal{N}} = \langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \rangle$ and an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $\lambda_Q(e) = \mu$, $f_Q \circ k'_Q = k_Q[e \mapsto m]$, and $f_Q(X) = \mathsf{Init}'_Q = \mathsf{Init}'$. We define $f = f_Q$ and the result follows. If no such $b(a)[m]$ and $\overline{b}(a)[m]$ exist in Q then clearly $a \notin n(\lambda(e))$, and restriction preserves morphisms, meaning the proof is straightforward.

□

D.13 Lemma 6.6.3

Proof. We first prove that if X is a possible cause, then it fulfils these conditions by induction on the size of X.

If $X = \emptyset$, then the result is trivial.

Otherwise, if $X \neq \emptyset$ is a possible cause of e then for each $e' \in X$, these exists $X' \subseteq (X \setminus \{e'\}$ such that X' is a possible cause of e'. By definition, X is finite and conflict free. Additionally, there can only exist one such X', since different possible causes of the same event must include different events from the same bundle, and are therefore in conflict. We can therefore say that $e' <_X e''$ if there exists a possible cause of e'', X'', such that $X' \subseteq X'' \subseteq X$, and since events cannot appear in their own possible causes, $\leq_X$ is a partial order. By induction, X must then be reachable, by performing the events in a sequence ordered by $<_X$.

Since X contains one event from each bundle associated with e, and $X \cup \{e\}$ is conflict free and therefore a configuration, $X \xrightarrow{e}$.

For any reachable configuration, $X'' \subset X$, since X'' is reachable, if $e' \in X \setminus X''$, then there cannot exist $e'' \in X''$ such that $e' <_X e''$. Therefore there must exist at least one $e''' \in X \setminus X''$ such that e''' is a maximal element of X with regards to $<_X$. This means that $e''' \in Y \mapsto e$ for some Y, and therefore $X'' \cap Y = \emptyset$, and $X'' \not\xrightarrow{e}$.

If X is reachable, $X \xrightarrow{e}$, and for all reachable configurations $X'' \subset X$, $X'' \not\xrightarrow{e}$, and for any e_Y and $Y \subset X$ such that Y is reachable, $Y \xrightarrow{e_Y}$, and for all reachable configurations $Y'' \subset Y$, $Y'' \not\xrightarrow{e_Y}$, Y is a possible cause of e_Y, then we show that X fulfils the conditions of Definition 6.6.2:

- Since $X \xrightarrow{e}$, $e \notin X$.
- X is finite by definition.
- Since $X \xrightarrow{e}$, by definition, for any $X' \mapsto e$, $X' \cap X \neq \emptyset$.
- Since $X \xrightarrow{e}$, $X \cup \{e\}$ is a configuration and therefore conflict free.
- Since X is reachable, there exists a trace, $\emptyset \xrightarrow{e_0} X_1 \xrightarrow{e_1} \ldots X_n \xrightarrow{e_n} X$, and for any e_i, there exists $X'_i \subseteq X_i$, which fulfils the conditions and is therefore a possible cause of e_i.
- If $X' \subseteq X$ then either X' is not a reachable configuration, or $X' \not\xrightarrow{e}$. In the first case, if X' is not a configuration, then X' is not conflict-free and therefore cannot be a possible cause. If X' is a non-reachable configuration, then there exists $e' \in X'$ such that for all reachable $X'' \subset X'$, $X'' \nrightarrow e'$. As shown in the first part of this proof, this means that no $X'' \subseteq X'$ is a possible cause of e', and therefore X' is not a possible cause. If X' is a reachable configuration and $X' \not\xrightarrow{e}$, then either $X' \cup \{e\}$ is not a configuration and therefore not conflict free, or there exists $Y \mapsto e$ such that $Y \cap X' = \emptyset$. Either way, X' is not a possible cause of e. □

D.14 Proposition 6.6.6

Proof. We first show that given a CLRBES, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $p_B(\mathcal{E})$ is a CLRPES. This follows from Definition 4.4.13. We then show that given a morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$ between two CLRBESs $P_B(f) : P_B(\mathcal{E}_0) \to P_B(\mathcal{E}_1)$ is an LRPES morphism. Let $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$, $P_B(\mathcal{E}_0) = (E'_0, F'_0, \prec'_0, \sharp'_0, \preceq'_0, \rhd'_0, \lambda'_0, \mathsf{Act}'_0)$, $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$, and $P_B(\mathcal{E}_1) = (E'_1, F'_1, \prec'_1, \sharp'_1, \preceq'_0, \rhd'_0, \lambda'_0, \mathsf{Act}'_0)$.

We then show that $P_B(f)$ fulfils the conditions of a LRPES morphism.

- Given $(X_0, e_0) \in E'_0$ with $P_B(f)((X_0, e_0)) = (X_1, e_1) \neq \bot$ then $\{(X'_1, e'_1) \mid (X'_1, e'_1) \prec'_1 (X_1, e_1)\} = \{(X'_1, e'_1) \mid X'_1 \cup \{e'_1\} \subseteq X_1\} \subseteq \{f((X'_0, e'_0)) \mid X'_0 \cup \{e'_0\} \subseteq X_0\}$.

- Given events $(X, e), (X', e') \in E'_0$ such that $P_B(f)((X, e)) \neq \bot \neq P_B(f)((X', e'))$, if we have a conflict $P_B(f)((X, e)) \sharp'_1 P_B(f)((X', e'))$ then either $P_B(f)((X, e)) \sharp''_1 P_B(f)((X', e'))$, $P_B(f)((X', e')) \sharp'_1 P_B(f)((X, e))$, or there exists (X'', e'') such that $(X'', e'') \prec'_1 P_B(f)((X', e'))$ and $P_B(f)((X, e)) \sharp'_1 (X'', e'')$.

  If $P_B(f)((X, e)) \sharp''_1 P_B(f)((X', e'))$ then either $f(e) \sharp_1 f(e')$ or $f(e) = f(e')$, and in either case $e \sharp_0 e'$, meaning $(X, e) \sharp'_0 (X', e')$.

  If $P_B(f)((X', e')) \sharp'_1 P_B(f)((X, e))$, then $(X', e') \sharp'_0 (X, e)$ and since $\sharp'_0$ is closed under symmetry, $(X, e) \sharp'_0 (X', e')$.

  If there exists (X'', e'') such that $(X'', e'') \prec'_1 P_B(f)((X', e'))$ and $P_B(f)((X, e)) \sharp'_1 (X'', e'')$, then $X'' \cup \{e''\} \subseteq f(X')$, meaning we have e''_0 such that $f(e''_0) = e''$ and $e''_0 \in X'''_0 \mapsto_0 e'$ and $X''_0 \subseteq X'$ such that $(X'', e_0) \in E_0$ and $f(X''_0) = X''$. We therefore get $(X'', e'') \prec'_0 (X', e')$ and $(X, e) \sharp'_0 (X'', e'')$, and therefore $(X, e) \sharp'_0 (X', e')$.

- $\prec'_0 = \prec_0 = \emptyset \times \emptyset$.

- Given $(X, e), (X', e') \in E'_0$ such that $P_B(f)((X, e)) \neq \bot \neq P_B(f)((X', e'))$, if $P_B(f)((X, e)) \rhd'_1 \underline{P_B(f)((X', e'))}$ then $P_B(f)((X', e')) \prec'_1 P_B(f)((X, e))$, meaning $f(X') \cup \{f(e')\} \subseteq f(X)$. Since f is a morphism, we get $e \rhd_0 \underline{e'}$, and since $\mathcal{E}_0$ is causal, $e' \in X'' \mapsto_0 e$, meaning $(X', e') \prec'_0 (X, e)$, and therefore $(X, e) \rhd'_0 (X', e')$.

- If $P_B(f)((X, e)) = P_B(f)((X, e')) \neq \bot$ and $(X, e) \neq (X', e')$ then either $e = e'$ and $X \neq X'$, or $e \neq e'$ and $f(e) = f(e')$. Either way $(X, e) \sharp'_0 (X', e')$.

- P_B does not affect F.

- P_B does not affect λ.

The rest of the proof of P_B being a functor is straightforward. □

D.15 Lemma 6.6.7

Proof. We first prove that if $Y \xrightarrow{e}$ then there exists a reachable configuration $Y' = \{(X', e') \mid X' \cup \{e'\} \subseteq Y\}$ and $X \subseteq Y$ such that $Y' \xrightarrow{(X,e)}$. We do this by induction in the size of Y.

If $Y = \emptyset$, then clearly $\emptyset$ is a possible cause of e, and $Y' = \emptyset \xrightarrow{(\emptyset,e)}$.

If $Y \neq \emptyset$ then since $\mathcal{E}$ is causal, Y must be forwards-reachable. Therefore, by induction, Y is also forwards-reachable. And clearly Y' contains all causes of and no conflicts with (X, e), giving us $Y' \xrightarrow{(X,e)}$.

We then prove that if $Y' \xrightarrow{(X,e)}$ then $Y = X \cup \bigcup_{(X',e') \in Y'} X' \cup \{e'\}$ is a reachable configuration and $Y \xrightarrow{e}$. We prove this by induction in the size of Y'.

If $Y' = \emptyset$ then $X = \emptyset$ and therefore $Y = \emptyset$. And since $\emptyset$ is a possible cause of e, $Y \xrightarrow{e}$.

If $Y' \neq \emptyset$ then since $P_B(\mathcal{E})$ is causal, Y' must be forwards-reachable. And since $Y' \xrightarrow{(X,e)}$, $X \subseteq Y'$, and $Y = \bigcup_{(X',e') \in Y'} X' \cup \{e'\}$. Therefore, by induction, Y is also forwards-reachable. And since Y contains a possible cause of e, $Y \xrightarrow{e}$. □

D.16 Theorem 6.6.9

Proof. Suppose $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $\mathcal{E}' = (E', F', \mapsto', \sharp', \prec', \rhd', \lambda', \mathsf{Act}')$, and $P_B(\mathcal{E}) = (E_P, F_P, <_P, \sharp_P, \prec_P, \rhd_P, \lambda_P, \mathsf{Act}_P)$.

We first note that, if an event in E' has a minimal representative, t, containing an action, which either receives names not in $\mathcal{N}$, or can only be reached by using α-conversion in a way other than what is used for replication when generating $\mathcal{E}$, then e cannot cause any events, for which the above does not hold.

We prove the theorem by induction on the structure and level of nesting of replication in P.

If P does not contain replication, then we look at the structure of P:

1. Suppose $P = 0$. Then $\mathcal{E}$ and $\mathcal{E}'$ both have no events.

2. Suppose $P = a(x).Q$. Then for each name n, $\mathcal{E}'$ has an event containing the trace $(a(n), [P][Q[x := n]]), \emptyset)$ and for each name $n' \in \mathcal{N}$, $\mathcal{E}$ has an event with no bundles, $a(n')$. We have for each $Q[x := n]$, $\{\!|Q[x := n]|\!\}_{\mathcal{N}} = \langle \mathcal{E}_n, \mathsf{Init}_n, k_n \rangle$ and $\mathcal{E}'_n$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q[x := n]$ and by induction we have morphisms $f_n : P_B(\mathcal{E}_n) \to \mathcal{E}'_n$ and $f'_n : \mathcal{E}'_n \to P_B(\mathcal{E}_n)$ such that $f_n \circ f'_n = 1_{P_B(\mathcal{E}_n)}$. This means we can define $f : P_B(\mathcal{E}) \to \mathcal{E}'$ as

$$f(e) = \begin{cases} \{(a(n), [P][Q[x := n]]), \emptyset)\} & \text{if } \lambda(e) = a(n) \text{ and} \\ & \text{no } e' \text{ exists such that } e' < e \text{ and} \\ \{(a(n), [P][Q[x := n]]), \emptyset)t \mid t \in f_n(e')\} & \text{if } e = (n, e') \text{ and} \\ & \text{there exists } e'' \text{ such that } e'' < e \end{cases}$$

This is clearly a morphism.

We also define $f' : \mathcal{E}' \to P_B(\mathcal{E})$ ans

$$f'(e) = \begin{cases} e' & \text{if } ((a(n), [P][Q[x := n]]), \emptyset) \in e, \\ & \lambda(e') = a(n), \text{ and no } e'' \text{ exists such that } e'' < e' \\ f'_n(\{t \mid ((a(n), [P][Q[x := n]]), \emptyset)t \in e\} & \text{if } ((a(n), [P][Q[x := n]]), \emptyset)t \in e \text{ for } t \neq \epsilon \\ \bot & \text{otherwise} \end{cases}$$

This is clearly also a morphism and $f \circ f' = 1_{P_B(\mathcal{E})}$. Additionally, for any $e' \in E'$, if $f'(e')$ is defined, then $f(f'(e')) = e'$.

3. Suppose $P = \overline{a}(x).Q$. This case is similar to the previous.

4. Suppose $P = Q|R$. Then we have $\{\!|Q|\!\}_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$, $P_B(\mathcal{E}_Q) = \mathcal{E}_{P_Q}$, and $\mathcal{E}'_Q$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q$ as well as $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $P_B(\mathcal{E}_R) = \mathcal{E}_{P_R}$, and $\mathcal{E}'_R$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash R$, and by induction there exist morphisms $f_Q : P_B(\mathcal{E}_Q) \to \mathcal{E}'_Q$, $f'_Q : \mathcal{E}'_Q \to P_B(\mathcal{E}_Q)$, $f_R : P_B(\mathcal{E}_R) \to \mathcal{E}'_R$, and $f'_R : \mathcal{E}'_R \to P_B(\mathcal{E}_R)$, such that $f_Q \circ f'_Q = 1_{P_B(\mathcal{E}_Q)}$ and $f_R \circ f'_R = 1_{P_B(\mathcal{E}_R)}$. We then define $f : P_B(\mathcal{E}) \to \mathcal{E}'$ as

$$f((X,e)) = \begin{cases} [(\mu, 0u, \emptyset)]_\approx & \text{if } X = \emptyset,\ e = (e_Q, *), \text{ and } f_Q(e_Q) = (\mu, u, \emptyset) \\ [(\mu, 1u, \emptyset)]_\approx & \text{if } X = \emptyset,\ e = (*, e_R), \text{ and } f_R(e_R) = (\mu, u, \emptyset) \\ [(\tau, \langle 0u_0, 1u_1\rangle, \emptyset)]_\approx & \text{if } X = \emptyset,\ e = (e_Q, e_R),\ f_Q(e_Q) = (\alpha_0, u_0, \emptyset), \\ & \text{and } f_R(e_R) = (\alpha_1, u_1, \emptyset) \\ e' & \text{if } e = (e_Q, *),\ f_Q(e_Q) = (\mu, u, D),\ \mathsf{last}(e') = (\mu, 0u, D'), \\ & \text{and } e'' < e' \text{ iff } \exists (X''', e''').f((X''', e''') = e'' \text{ and } X''' \cup \{e'''\} \subseteq X \\ e' & \text{if } e = (*, e_R),\ f_R(e_R) = (\mu, u, D),\ \mathsf{last}(e') = (\mu, 1u, D'), \\ & \text{and } e'' < e' \text{ iff } \exists (X''', e''').f((X''', e''') = e'' \text{ and } X''' \cup \{e'''\} \subseteq X \\ e' & \text{if } e = (e_Q, e_R),\ f_Q(e_Q) = (\alpha_0, u_0, D_0),\ f_R(e_R) = (\alpha_1, u_1, D_1), \\ & \mathsf{last}(e') = (\tau, \langle 0u_0, 1u_1\rangle, D'), \\ & \text{and } e'' < e' \text{ iff } \exists (X''', e''').f((X''', e''') = e'' \text{ and } X''' \cup \{e'''\} \subseteq X \end{cases}$$

To define the inverse of f, $f' : \mathcal{E}' \to P_B(\mathcal{E})$, we first create two helper functions g_0 and g_1, which extract the part of a trace which took place in each half of the process:

$$g_Q(t) = \begin{cases} \epsilon & \text{if } t = \epsilon \\ g_Q(t')(\mu, u, D') & \text{if } t = t'(\mu, 0u, D), D' = \{(n,u) \mid (n, 0u) \in D\} \\ g_Q(t') & \text{if } t = t'(\mu, 1u, D) \\ g_Q(t')(\alpha_0, u_0, D') & \text{if } t = t'(\tau, \langle 0l_0[P_0][P_0'], 1l_1[P_1][P_1']\rangle, D), \\ & \exists i \in \{0,1\}.P_i \xrightarrow{a(b)} P_i',\ P_{1-i} \xrightarrow{\overline{a}(b)} P_{1-i}',\ \alpha_i = a(b), \text{ and } \alpha_{1-i} = \overline{a}(b), \\ & \text{and } D' = \{(n,u) \mid n \in no(\alpha_0) \text{ and } t = t_0(\overline{a}(n), 0u, D_0)t_1\} \end{cases}$$

$$g_R(t) = \begin{cases} \epsilon & \text{if } t = \epsilon \\ g_R(t')(\mu, u, D') & \text{if } t = t'(\mu, 1u, D), D' = \{(n,u) \mid (n, 1u) \in D\} \\ g_R(t') & \text{if } t = t'(\mu, 0u, D) \\ g_R(t')(\alpha_1, u_1, D') & \text{if } t = t'(\tau, \langle 0l_0[P_0][P_0'], 1l_1[P_1][P_1']\rangle, D), \\ & \exists i \in \{0,1\}.P_i \xrightarrow{a(b)} P_i',\ P_{1-i} \xrightarrow{\overline{a}(b)} P_{1-i}',\ \alpha_i = a(b), \text{ and } \alpha_{1-i} = \overline{a}(b), \\ & \text{and } D' = \{(n,u) \mid n \in no(\alpha_1) \text{ and } t = t_0(\overline{a}(n), 1u, D_1)t_1\} \end{cases}$$

It is clear that if t is a trace in $Q|R$, then $g_Q(t)$ is a trace in Q and $G_R(t)$ is a trace in R. We then define f' on e' with minimal representative $t(\mu, u, \emptyset)$:

$$f'(e') = \begin{cases} (X, (e_Q, e_R)) & \text{where if } u = 0u_0 \text{ or } u = \langle 0u_0, 1u_1\rangle \text{ then } (X_Q, e_Q) = f_Q'([g_Q(t(\mu, u, \emptyset))]_\approx) \neq \perp, \\ & \text{and otherwise } e_Q = * \\ & \text{and if } u = 1u_1 \text{ or } u = \langle 0u_0, 1u_1\rangle \text{ then } (X_R, e_R) = f_R'([g_R(t(\mu, u, \emptyset))]_\approx) \neq \perp, \\ & \text{and otherwise } e_R = * \\ & \text{if } t \text{ is empty, then } X = \emptyset, \text{ otherwise } X = \bigcup_{\exists e'' < e'.f''(e'') = (X', e)} X' \cup \{e\} \\ \perp & \text{if } f_Q''([g_Q(t(\mu, u, \emptyset))]_\approx) = \perp \text{ or } f_R''([g_R(t(\mu, u, \emptyset))]_\approx) = \perp \text{ in the relevant cases} \end{cases}$$

We must first prove that if $(X, e) \in E_P$, then $f((X, e)) \in E'$. By Theorems 6.3.8, 6.4.11, and 6.4.12 and Lemma 6.6.7, this only requires proving that there exists a trace $e_0, e_1, \ldots, e_n, e$ from $\emptyset$ where

$\{e_0, e_1, \ldots, e_n\} = X$. If $X = \emptyset$, then there exist no bundles $X' \mapsto e$, and this trivial. If $X \neq \emptyset$, then for each $e' \in X$, there exists $X' \subseteq X$ such that X' is a possible cause of e', so we order X based on $e'' < e'$ if $e'' \in X'$, and any sequence obeying this order is a trace. We then show that $X \xrightarrow{e}$ in the CS of E, This follows from Definition 4.4.4.

We then prove that for any $e_P = (X, e) \in E_P$, $f'(f(e_P)) = e_P$. We do this by induction in the size of X.

If $X = \emptyset$, then this trivial.

If $X \neq \emptyset$, and for all $(X', e') \in E_P$ with X' smaller than X, $f'(f((X', e'))) = (X', e')$ then:

If $e = (e_Q, *)$, then by induction, there exists $(X_Q, e_Q) \in E_{Q_P}$ such that $f((X_Q, e_Q)) = [g_Q(f(e_P))]_\approx$ and $f'([g_Q(f(e_P))]_\approx) = (X_Q, e_Q)$. Therefore, $f'(f(e_P)) = (X'', e)$ for some X''. For any $e'_P < e_P$, we know $f'(f(e'_P)) = e'_P$, and therefore we get $X'' = X$.

We then prove that f' is a morphism, if $e' \in E'$ and $f'(e') \neq \bot$, then there exists $f'(e') \in E_P$, and $f(f'(e')) = e'$. We do this by induction on the length of the minimal representatives of e', showing that that $f'(e') = (X_P, e_P) \in E_P$ and $f'_n = f' \upharpoonright \{e \mid$ all minimal representatives of e are of length $n\}$ is a morphism, and $f(f'_n(e') = e'$.

If the length of the minimal representatives of e' is 1, then we have three cases:

If $t = (\mu, 0u, \emptyset)$ is a minimal representative of e' and $f'(e') \neq \bot$, then by Theorems 6.3.8, 6.4.11, and 6.4.12 and Lemma 6.6.7 there must exist an event $e_Q \in E_Q$, such that $f'_Q((\mu, u, \emptyset)) = (X, e_Q)$, which can be performed from the configuration $\emptyset$ and has no link causes from R, meaning $X = \emptyset$ and $(\emptyset, (e_Q, *)) \in E_P$. Additionally, by induction, $f_Q(f'_Q([(\mu, u, \emptyset)]_\approx)) = [(\mu, u, \emptyset)]_\approx$, and therefore $f(f'_n(e') = e'$.

If $t = (\mu, 0u, \emptyset)$ is a minimal representative of e' then the argument is similar.

If $t = (\tau, \langle 0u_0, 1u_1\rangle, \emptyset)$ is a minimal representative of e', then by induction we have events $(X_Q, e_Q) = f'_Q([g_Q((\tau, \langle u_0, u_1\rangle, \emptyset))]_\approx) \in E_{P_Q}$, and $(X_R, e_R) = f'_R([g_R((\tau, \langle u_0, u_1\rangle, D))]_\approx) \in E_{P_R}$, and by Theorems 6.3.8, 6.4.11, and 6.4.12 and Lemma 6.6.7 neither e_Q or e_R have any causes, meaning $X_Q = X_R = \emptyset$ and $(\emptyset, (e_Q, e_R))$. Additionally, by induction, we get $f_Q(f'_Q([g_Q((\tau, \langle u_0, u_1\rangle, \emptyset))]_\approx)) = [g_Q((\tau, \langle u_0, u_1\rangle, \emptyset))]_\approx$ and $f_R(f'_R([g_R((\tau, \langle u_0, u_1\rangle, \emptyset))]_\approx)) = [g_Q((\tau, \langle u_0, u_1\rangle, \emptyset))]_\approx$, and therefore $f(f'_n(e') = e'$.

Additionally, we must prove that $f' \upharpoonright \{e \mid$ all minimal representatives of e are of length $0\} = f'_0$ is a morphism.

(a) for all $e \in E'$, if $f'_0(e) \neq \bot$ then $\{e' \mid e <_P f'_0(e')\} = \emptyset = \{f(e'') \mid e'' <' e'\}$.

(b) for all $e, e' \in E'$, if $f'_0(e) \neq \bot \neq f'_0(e')$ and $f'_0(e) \sharp_P f'_0(e')$ then e has a minimum representative $(\mu, u, \emptyset)$ and e' has a minimum representative $(\mu', u', \emptyset)$ and either $f'_0(e) = (\emptyset, (e_Q, *))$ and $f'_0(e') = (\emptyset, (e'_Q, *))$, $f'_0(e) = (\emptyset, (e_Q, *))$ and $f'_0(e') = (\emptyset, (e'_Q, e'_R))$, $f'_0(e) = (\emptyset, (e_Q, e_R))$ and $f'_0(e') = (\emptyset, (e'_Q, e'_R))$, $f'_0(e) = (\emptyset, (e_Q, e_R))$ and $f'_0(e') = (\emptyset, (e'_Q, *))$, $f'_0(e) = (\emptyset, (*, e_R))$ and $f'_0(e') = (\emptyset, (*, e'_R))$, $f'_0(e) = (\emptyset, (*, e_R))$ and $f'_0(e') = (\emptyset, (e'_Q, e'_R))$, or $f'_0(e) = (\emptyset, (e_Q, e_R))$ and $f'_0(e') = (\emptyset, (*, e'_R))$.
If $f'_0(e) = (\emptyset, (e_Q, *))$ and $f'_0(e') = (\emptyset, (e'_Q, *))$, then by induction $(\emptyset, e_Q) \sharp_{Q_P} (\emptyset, e_Q)$ and $u = 0u_Q$ and $u' = 0u'_Q$, meaning $[(\mu, u_Q, \emptyset)]_\approx \sharp'_Q [(\mu' u'_Q, \emptyset)]_\approx$, and therefore $e' \sharp e''$.
If $f'_0(e) = (\emptyset, (e_Q, *))$ and $f'_0(e') = (\emptyset, (e'_Q, e'_R))$, then by induction $(\emptyset, e_Q) \sharp_{Q_P} (\emptyset, e_Q)$, $u = 0u_Q$, $u' = \langle 0u'_Q, 1u'_R\rangle$, and $[(\mu, u_Q, \emptyset)]_\approx \sharp'_Q [g_Q((\tau, u', \emptyset)]_\approx$. Therefore, $e' \sharp' e''$.
The other cases are similar.

(c) for all $e' \in F'$, if $f'_0(e) \neq \bot$ then $\{e \mid e <_P \underline{f'_0(e)}\} = \emptyset = \{f'_0(e'') \mid e'' <' \underline{e'}\}$.

(d) for all $e \in E'$ and $e' \in F'$, if $f'_0(e) \neq \bot \neq f'_0(e')$ then $f'_0(e) \not> _P \underline{f'_0(e')}$.

(e) for all $e, e' \in E'$, if $f'_0(e) = f'_0(e') \neq \bot$ and $e \neq e'$ then $(\mu, u, \emptyset)$ is a minimal representative of e, $(\mu', u', \emptyset)$ is a minimal representative of e', and for some $i \in \{R, Q\}$, $f'_i([g_Q((\mu, u, \emptyset))]_\approx) = f'_i([g_i((\mu', u', \emptyset))]_\approx)$ and $[g_i((\mu, u, \emptyset))]_\approx \neq [g_i((\mu', u', \emptyset))]_\approx$.
If $i = Q$ then $[g_Q((\mu, u, \emptyset))]_\approx \sharp'_Q [g_Q((\mu', u', \emptyset))]_\approx$, and therefore $e \sharp_0 e'$. If $i = R$, the argument is similar.

(f) $f'_0(F') \subseteq F_P = E_P$.

If the length of the minimal representatives of e' is $n > 1$, and for any $e'' \in E'$ with minimal representatives of length $n' < n$, $f'(e'') = (X''_P, e''_P) \in E_P$, and we have a morphism $f'_{n-1} = f' \upharpoonright \{e \mid$ all minimal representatives of e are of length $n-1\}$, then we first show that $f'(e') \in E_P$, for which we again have 3 cases for e':

If $\mathsf{last}(e') = (\mu, 0u, D)$ and t of length n is a minimal representative of e', then $g_Q(t)$ is a trace in Q, and therefore by induction, $f'_Q(t) = (X_Q, e_Q) \in E_{P_Q}$, meaning there exists $(X, (e_Q, *)) \in E_P$ for some X. We therefore need to prove that $X = \bigcup_{\exists e'' < e'. f'(e'') = (X', e)} X' \cup \{e\}$ is a possible cause of $(e_Q, *)$ in $\mathcal{E}$. If $e'' < e'$, then e'' must have a minimum representative t' of length $n' < n$ such that $t'' \overline{\sim} t'$ is a prefix of t. By induction, this means $f'(e'') \in E_P$. By Lemma 6.6.3, we only need to prove that (1) X is a configuration, which it is because $\{e'' \mid e'' < e'\}$ is conflict free, and by induction $\{f'_{n-1}(e'') \mid e'' < e'\}$ is conflict free, making X conflict free. (2) X is reachable, which it is because for each $e \in X$, there exists a possible cause of e, $X' \subset X$, and if we order the events of X by inclusion of these associated possible causes, we get a partial order in which the events can be performed, similarly to Lemma 6.6.3. (3) X is finite, which it is because $\{e'' \mid e'' <' e'\}$ is finite, and each potential cause of e'' is finite. (4) $X \xrightarrow{e}$ by Theorems 6.3.8, 6.4.11, and 6.4.12 and Lemma 6.6.7. (5) If $X'' \subset X$ is a reachable configuration, then there exists $e'' <' e'$ such that $f'_{n-1}(e'') = (X'', e)$ and $e \notin X''$, and since X'' is reachable, and therefore must include all causes of its events, we have minimum representative of e'', t'', such that $t = t_0 t_1$ and $t_0 \overline{\sim} t''$. Since t is a minimal representative, there must exist an action, (μ, u, D) in t_1 such that $\neg(\mathsf{last}(t'') I (\mu, u, D))$. If $(\mu, u, D) \neq \mathsf{last}(t)$, then by induction, since X'' is reachable, there exists e''' such that $\mathsf{last}(e''') = (\mu, u, D)$ and $e'' < e''' < e'$ and $f'_{n-1}(e''') = (X''', e'''_P)$ for $e_P \in X \setminus X''$. If $(\mu, u, D) = \mathsf{last}(t)$, then by Theorems 6.3.8, 6.4.11, and 6.4.12 and Lemma 6.6.7, $X'' \xrightarrow{e}$.

In addition, we must show that $f(f'(e')) = e'$. To do this, we merely need to show that there does not exist $e \in E'$ such that $\mathsf{last}(e) = \mathsf{last}(e')$ and for all $e'' \in E'$, $e'' < e$ iff $e'' < e'$, but $e \neq e'$. This is clearly not possible.

If $\mathsf{last}(e') = (\mu, 1u, D)$ or $\mathsf{last}(e') = (\tau, \langle 0u_0, 1u_1 \rangle, D)$, then the arguments are similar.

We then prove that $f'_n = f' \upharpoonright \{e \mid$ all minimal representatives of e are of length $n\}$ is a morphism:

(a) for all $e' \in E'$, if $f'_n(e') = (X, e) \neq \bot$ then $\{e'' \mid e'' <_P f_n(e')\} = \{(X'', e'') \mid \{e''\} \cup X'' \subseteq X\}$. We must now show that there cannot exist $(X'', e'') \in E_P$ such that $\{e''\} \cup X'' \subseteq X$ and there does not exist $e''' \in E'$ such that $f'_n(e''') = (X'', e'')$. This follows from Theorems 6.3.8, 6.4.11, and 6.4.12 and Lemmas 6.6.3 and 6.6.7.

(b) for all $e, e' \in E'$, if $f_n(e) = (X, e_P) \neq \bot \neq f_n(e') = (X', e'_P)$ and $f_n(e) \sharp_P f_n(e')$ then we have minimal representatives of e and e', t and t' and either $e_P \sharp e'_P$, or $e_P = e'_P$ and $X \neq X'$. In the first case, either $f'_Q([g_Q(t)]_\approx) \sharp_{Q_P} f'_Q([g_Q(t')]_\approx)$ or $f'_R([g_R(t)]_\approx) \sharp_{R_P} f'_R([g_R(t')]_\approx)$, and by induction, either $[g_Q(t)]_\approx \sharp'_Q [g_Q(t')]_\approx$ or $[g_R(t)]_\approx \sharp'_Q [g_R(t')]_\approx$, meaning $e \sharp' e'$. In the second case, there must exist an action (μ, u, D) in t, which is not in t', and (μ', u', D') in t', which is not in t, and $\neg((\mu, u, D) I \mathsf{last}(t'))$ and $\neg((\mu', u', D') I \mathsf{last}(t))$. Therefore, $e \sharp' e'$.

(c) for all $e' \in F'$, if $f'_n(e') \neq \bot$ then $\{e \mid e \prec_P \underline{f'_n(e')}\} = \emptyset = \{f'_n(e) \mid e \prec' \underline{e'}\}$;

(d) for all $e \in E$ and $e' \in F_0$, if $f'_n(e) \neq \bot \neq f'_n(e')$ and $f'_n(e) \rhd_P \underline{f'_n(e')}$ then $f'_n(e') <_P f'_n(e)$, meaning as previously shown, $e' <' e$, and therefore $e \rhd' \underline{e'}$.

(e) for all $e, e' \in E'$, if $f'_n(e) = f'_n(e') \neq \bot$ then $e = e'$.

(f) $f(F_0) \subseteq F_1$.

We then prove that f is a morphism.

(a) for all $e \in E_P$, if $f(e) \neq \bot$ then $\{e' \mid e' <' f(e)\} = \{e' \mid \exists t' \in e'. \forall t \in f(e). \exists t'' \overline{\sim} t. t''$ is a prefix of $t'\} = \{f((X', e_P) \mid X' \cup \{e_P\} \subseteq X\} = \{f(e_P) \mid e_P <_P e\}$.

(b) for all $e_0, e_1 \in E_P$, if $f(e_0) \neq \bot \neq f(e_1)$ and $f(e_0) \sharp' f(e_1)$ then there exist events $e'_0 \leq' f(e_0)$ and $e'_1 \leq' f(e_1)$ and traces $t(\mu_0, u_0, D_0) \in e'_0$ and $t(\mu_1, u_1, D_1) \in e'_1$, such that $(\mu_0, u_0, D_0) \neq (\mu_1, u_1, D_1)$ and not $(\mu_0, u_0, D_0) I (\mu_1, u_1, D_1)$. We therefore have $e''_0, e''_1 \in E_P$, such that $e''_0 \leq e_0$ and $f(e''_0) = e'_0$ and $e''_1 \leq e_1$ and $f(e''_1) = e'_1$. Since $\sharp_P$ is closed under conflict heredity, we only need to prove that $e''_0 \sharp e''_1$. If there exists $i \in 0, 1$ such that $u_0 = iu'_0$ and $u_1 = iu'_1$, then this follows from induction. Otherwise, there exists $i \in \{0, 1\}$ and $n \in \mathbb{N}$ such that $(n, u_i) \in D_{1-i}$, but this would require some (μ'_i, u_i, D'_i) to appear in t, which contradicts $t(\mu_i, u_i, D_i)$ being a trace.

(c) for all $e \in F_P$, if $f(e) \neq \bot$ then $\{e' \mid e' \prec' \underline{f(e)}\} = \emptyset = \{f(e') \mid e' \prec_P \underline{e}\}$.

(d) for all $e \in E_P$ and $e' \in F_P$, if $f(e) \neq \bot \neq f(e')$ and $f(e) \rhd' \underline{f(e')}$ then $f(e') \prec' f(e)$, and from Item 1 we get $e' \prec_P e$, and therefore $e \rhd_P \underline{e'}$.

(e) for all $e, e' \in E_0$, if $f(e) = f(e') \neq \bot$ then $e = e'$.

(f) all our events are reversible.

5. Suppose $P = Q + R$. Then we have $\{\!|Q|\!\}_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$ and $\mathcal{E}'_Q$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q$ as well as $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$ and $\mathcal{E}'_R$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash R$, and there exist morphisms $f_Q : P_B(\mathcal{E}_Q) \to \mathcal{E}'_Q$, $f'_Q : \mathcal{E}'_Q \to P_B(\mathcal{E}_Q)$, $f_R : P_B(\mathcal{E}_R) \to \mathcal{E}'_R$, and $f'_R : \mathcal{E}'_R \to P_B(\mathcal{E}_R)$ such that $f_Q \circ f'_Q = 1_{P_B(\mathcal{E}_Q)}$ and $f_R \circ f'_R = 1_{P_B(\mathcal{E}_R)}$. We define $f : P_B(\mathcal{E}) \to \mathcal{E}'$ as

$$f(e) = \begin{cases} \{(\alpha, [Q' + R][Q''], D)t \mid (\alpha, [Q'][Q''], D)t \in f_Q(e')\} & \text{if } e = (Q, e') \\ \{(\alpha, [Q + R'][R''], D)t \mid (\alpha, [R'][R''], D)t \in f_R(e')\} & \text{if } e = (R, e') \end{cases}$$

. This is clearly a morphism.

We then define $f' : \mathcal{E}' \to P_B(\mathcal{E})$ as

$$f'(e) = \begin{cases} f'_Q(\{(\alpha, [Q'][Q''], D)t \mid (\alpha, [Q' + R][Q''], D)t \in e\}) & \text{if } \exists \alpha, Q', Q'', D, t. \\ & (\alpha, [Q' + R][Q''], D)t \in e \\ & \text{and } (\alpha, [Q'][Q''], D)t \in e_Q \in E'_Q \\ f'_R(\{(\alpha, [R'][R''], D)t \mid (\alpha, [Q + R'][R''], D)t \in e\}) & \text{if } \exists \alpha, R', R'', D, t. \\ & (\alpha, [Q + R'][R''], D)t \in e \\ & \text{and } (\alpha, [R'][R''], D)t \in e_R \in E'_R \\ \bot & \text{otherwise} \end{cases}$$

This is clearly also a morphism and since all names n P are distinct and therefore, $Q \neq R$, $f \circ f' = 1_{P_B(\mathcal{E})}$. Additionally, for any $e' \in E'$, if $f'(e')$ is defined, then $f(f'(e')) = e'$

6. Suppose $P = (\nu x)Q$. Isomorphism is preserved by restriction.

If P contains a nested replication of level n, and the result holds for all processes with a maximum level nested replication of $m < n$, then we again look at the structure of P. If $P \neq !P'$, then we can use the same arguments as above.

If $P = !P'$, then $\{\!|!_{\mathbf{x}} P'|\!\}_{\mathcal{N}} = \sup_{l \in \mathbb{N}} \{\!|!_{\mathbf{x}} P|\!\}_{(\mathcal{N}, l)} =$

$$\left\{\!\!\left| \ldots ((P'\{^{x_1,\ldots,x_k}/_{a_1,\ldots,a_k}\} | P'\{^{x_{k+1},\ldots,x_{2k}}/_{a_1,\ldots,a_k}\}) | P'\{^{x_{2k+1},\ldots,x_{3k}}/_{a_1,\ldots,a_k}\}) \ldots \right|\!\!\right\}_{\mathcal{N}}$$

where $\{a_1, \ldots, a_k\} = \mathsf{bn}(P')$, $x_1, \cdots \in \mathbf{x}$, and $x_i \neq x_j$ for $i \neq j$. The set of traces of $!P$ is the set of traces of $\ldots ((P' | P') | P') | \ldots$. By induction in the level of nesting replication, we get that the result holds for $\ldots ((P'\{^{x_1,\ldots,x_k}/_{a_1,\ldots,a_k}\} | P'\{^{x_{k+1},\ldots,x_{2k}}/_{a_1,\ldots,a_k}\}) | P'\{^{x_{2k+1},\ldots,x_{3k}}/_{a_1,\ldots,a_k}\}) \ldots$, and we refer to the morphisms in question as f_x and f'_x. Since we have

$$\ldots ((P' | P') | P') | \ldots \equiv_\alpha \ldots ((P'\{^{x_1,\ldots,x_k}/_{a_1,\ldots,a_k}\} | P'\{^{x_{k+1},\ldots,x_{2k}}/_{a_1,\ldots,a_k}\}) | P'\{^{x_{2k+1},\ldots,x_{3k}}/_{a_1,\ldots,a_k}\}) \ldots$$

we know the traces of $\ldots((P'\{^{x_1,\ldots,x_k}/_{a_1,\ldots,a_k}\}|P'\{^{x_{k+1},\ldots,x_{2k}}/_{a_1,\ldots,a_k}\})|P'\{^{x_{2k+1},\ldots,x_{3k}}/_{a_1,\ldots,a_k}\})\ldots$ are the same as the traces of $\ldots((P'|P')|P')|\ldots$. The result therefore follows from induction. □

D.17 Theorem 6.6.13

Proof. We Say $P_B(\mathcal{E}) = (E, F, <, \sharp, \prec, \rhd, \lambda, \mathsf{Act})$ and $\mathcal{E}' = (E', F', <', \sharp', \prec', \rhd', \lambda', \mathsf{Act}')$ and again prove this by induction on the structure and level of nesting of replication in P.

1. Suppose $P = 0$. Then both $\mathcal{E}$ and $\mathcal{E}'$ have no events.

2. Suppose $P = a(x).Q$. Then for each name n, $\mathcal{E}'$ has an event containing the trace $(a(n), [P][Q[x := n]]), \emptyset)$ and for each name $n' \in \mathcal{N}$, $\mathcal{E}$ has an event with no bundles, $a(n')$. We have for each $Q[x := n]$, $\{\!|Q[x := n]|\!\}_{\mathcal{N}} = \langle \mathcal{E}_n, \mathsf{Init}_n, k_n \rangle$, and $\mathcal{E}'_n$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q[x := n]$ and by induction we have morphisms $f_n : P_B(\mathcal{E}_n) \to \mathcal{E}'_n$ and $f'_n : \mathcal{E}'_n \to P_B(\mathcal{E}_n)$, defined as previously and $f''_n : \mathsf{Ir}(\mathcal{E}'_n) \to \mathsf{Ir}(\mathcal{E}_n)$ such that $f''_n \upharpoonright \mathsf{dom}(f'_n) = f'_n$.

   We then define f'' as

$$f''(e) = \begin{cases} f'_n(\{t \mid (a(n), u, D)t \in e\}) & \text{if } (a(n), u, D)t \in e \text{ for } t \neq \epsilon \text{ and } n \in \mathcal{N} \\ e' & \text{if } (a(n), u, \emptyset) \in e,\ n \in \mathcal{N}, \\ & \lambda(e') = a(n), \text{ and no } e'' \text{ exists such that } e'' < e' \\ f'_x(\{t[m := x]_{(u-\emptyset)} \mid (\mu, u, D)t \in e\}) & \text{if } (a(m), u, D)t \in e \text{ for } t \neq \epsilon \text{ and } m \notin \mathcal{N} \\ e' & \text{if } (a(m), u, \emptyset) \in e,\ m \notin \mathcal{N} \\ & \lambda(e') = a(x), \text{ and no } e'' \text{ exists such that } e'' < e' \end{cases}$$

   Clearly $f'' \upharpoonright \mathsf{dom}(f') = f'$, and f'' is a morphism.

3. Suppose $P = \overline{a}(b).Q$. This case is similar to the previous.

4. Suppose $P = Q|R$. Then we have $\{\!|Q|\!\}_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$, and $\mathcal{E}'_Q$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q$ as well as $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, and $\mathcal{E}'_R$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash R$, and by induction we have $f_Q : P_B(\mathcal{E}_Q) \to \mathcal{E}'_Q$, $f'_Q : \mathcal{E}'_Q \to P_B(\mathcal{E}_Q)$, $f''_Q : \mathsf{Ir}(\mathcal{E}'_Q) \to \mathsf{Ir}(\mathcal{E}_Q)$, $f_R : P_B(\mathcal{E}_R) \to \mathcal{E}'_R$, $f'_R : \mathcal{E}'_R \to P_B(\mathcal{E}_R)$, $f''_R : \mathsf{Ir}(\mathcal{E}'_R) \to \mathsf{Ir}(\mathcal{E}_R)$, with f_Q, f'_Q, f_R, and f'_R as before, and $f''_Q \upharpoonright \mathsf{dom}(f'_Q) = f'_Q$ and $f''_R \upharpoonright \mathsf{dom}(f'_R) = f'_R$.

   To defined f'', we again use the two helper functions g_0 and g_1 defined in the parallel case of the previous proof, which extract the part of a trace which took place in each half of the process:

$$g_Q(t) = \begin{cases} \epsilon & \text{if } t = \epsilon \\ g_Q(t')(\mu, u, D') & \text{if } t = t'(\mu, 0u, D), \text{ and } D' = \{(n, u) \mid (n, 0u) \in D\} \\ g_Q(t') & \text{if } t = t'(\mu, 1u, D) \\ g_Q(t')(\alpha_0, u_0, D') & \text{if } t = t'(\tau, \langle 0l_0[P_0][P'_0], 1l_1[P_1][P'_1]\rangle, D), \\ & \exists i \in \{0,1\}.P_i \xrightarrow{a(b)} P'_i,\ P_{1-i} \xrightarrow{\overline{a}(b)} P'_{1-i},\ \alpha_i = a(b), \text{ and } \alpha_{1-i} = \overline{a}(b), \\ & \text{and } D' = \{(n, u) \mid n \in no(\alpha_0) \text{ and } t = t_0(\overline{a}(n), 0u, D_0)t_1\} \end{cases}$$

$$g_R(t) = \begin{cases} \epsilon & \text{if } t = \epsilon \\ g_R(t')(\mu, u, D') & \text{if } t = t'(\mu, 1u, D), \text{ and } D' = \{(n,u) \mid (n,1u) \in D\} \\ g_R(t') & \text{if } t = t'(\mu, 0u, D) \\ g_R(t')(\alpha_1, u_1, D') & \text{if } t = t'(\tau, \langle 0l_0[P_0][P'_0], 1l_1[P_1][P'_1]\rangle, D), \\ & \exists i \in \{0,1\}. P_i \xrightarrow{a(b)} P'_i,\ P_{1-i} \xrightarrow{\overline{a}(b)} P'_{1-i},\ \alpha_i = a(b), \text{ and } \alpha_{1-i} = \overline{a}(b), \\ & \text{and } D' = \{(n,u) \mid n \in no(\alpha_1) \text{ and } t = t_0(\overline{a}(n), 1u, D_1)t_1\} \end{cases}$$

It is clear that if t is a trace in $Q|R$, then $g_Q(t)$ is a trace in Q and $G_R(t)$ is a trace in R. We define f'' of an event e' with a minimal trace $t(\mu, u, \emptyset)$:

$$f''(e') = \begin{cases} (X, (e_Q, e_R)) & \text{where if } u = 0u_0 \text{ or } u = \langle 0u_0, 1u_1\rangle \text{ then } (X_Q, e_Q) = f''_Q([g_Q(t(\mu,u,\emptyset))]_\approx) \neq \bot, \\ & \text{and otherwise } e_Q = * \\ & \text{and if } u = 1u_1 \text{ or } u = \langle 0u_0, 1u_1\rangle \text{ then } (X_R, e_R) = f''_R([g_R(t(\mu,u,\emptyset))]_\approx) \neq \bot, \\ & \text{and otherwise } e_R = * \\ & \text{if } t \text{ is empty, then } X = \emptyset, \text{ otherwise } X = \bigcup_{\exists e'' < e'. f''(e'') = (X', e)} X' \cup \{e\} \\ \bot & \text{if } f''_Q([g_Q(t(\mu,u,\emptyset))]_\approx) = \bot \text{ or } f''_R([g_R(t(\mu,u,\emptyset))]_\approx) = \bot \text{ in the relevant cases} \end{cases}$$

Clearly, $f'' \upharpoonright \mathsf{dom}(f') = f'$, and the proof of f'' being a morphism is similar to the proof of f' being a morphism in the previous theorem.

5. Suppose $P = Q + R$. Then we have $\{\!|Q|\!\}_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q\rangle$, and $\mathcal{E}'_Q$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q$ as well as $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R\rangle$, and $\mathcal{E}'_R$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash R$, and by inductions we have morphisms $f_Q : P_B(\mathcal{E}_Q) \to \mathcal{E}'_Q$, $f'_Q : \mathcal{E}'_Q \to P_B(\mathcal{E}_Q)$, $f''_Q : \mathsf{Ir}(\mathcal{E}'_Q) \to \mathsf{Ir}(\mathcal{E}_Q)$, $f_R : P_B(\mathcal{E}_R) \to \mathcal{E}'_R$, $f'_R : \mathcal{E}'_R \to P_B(\mathcal{E}_R)$, $f''_R : \mathsf{Ir}(\mathcal{E}'_R) \to \mathsf{Ir}(\mathcal{E}_R)$, with f_Q, f'_Q, f_R, and f'_R as before, and $f''_Q \upharpoonright \mathsf{dom}(f'_Q) = f'_Q$ and $f''_R \upharpoonright \mathsf{dom}(f'_R) = f'_R$. We then define

$$f''(e) = \begin{cases} f''_Q(\{(\alpha, [Q'][Q''], D)t \mid (\alpha, [Q' + R][Q''], D)t \in e\}) & \text{if } \exists \alpha, Q', Q'', D, t. \\ & (\alpha, [Q' + R][Q''], D)t \in e \\ & \text{and } (\alpha, [Q'][Q''], D)t \in e_Q \in E'_Q \\ f''_R(\{(\alpha, [R'][R''], D)t \mid (\alpha, [Q + R'][R''], D)t \in e\}) & \text{if } \exists \alpha, R', R'', D, t. \\ & (\alpha, [Q + R'][R''], D)t \in e \\ & \text{and } (\alpha, [R'][R''], D)t \in e_R \in E'_R \end{cases}$$

It is straightforward to check that this is a morphism and $f'' \upharpoonright \mathsf{dom}(f') = f'$.

6. Suppose $P = (\nu x)Q$. Then Then we have $\{\!|Q|\!\}_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q\rangle$, and $\mathcal{E}'_Q$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q$ and by inductions we have morphisms $f_Q : P_B(\mathcal{E}_Q) \to \mathcal{E}'_Q$, $f'_Q : \mathcal{E}'_Q \to P_B(\mathcal{E}_Q)$, and $f''_Q : \mathsf{Ir}(\mathcal{E}'_Q) \to \mathsf{Ir}(\mathcal{E}_Q)$, with f_Q and f'_Q as before, and $f''_Q \upharpoonright \mathsf{dom}(f'_Q) = f'_Q$. The traces of P will be a subset of the traces of Q, and we define f'' as:

$$f''(e) = f''_Q(e') \text{ if } e' \subseteq e \text{ and } e' \in E'_Q$$

7. Suppose $P = !Q$. Then the argument is much the same as in Theorem 6.6.9. □