Decidability and Coincidence of Equivalences for Concurrency

Sibylle Fröschle

Doctor of Philosophy
Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh
2002

(Graduation date:May 2004)

Abstract

There are two fundamental problems concerning equivalence relations in concurrency. One is: for which system classes is a given equivalence decidable? The second is: when do two equivalences coincide? Two well-known equivalences are history preserving bisimilarity (hpb) and hereditary history preserving bisimilarity (hhpb). These are both 'independence' equivalences: they reflect causal dependencies between events. Hhpb is obtained from hpb by adding a 'backtracking' requirement. This seemingly small change makes hhpb computationally far harder: hpb is well-known to be decidable for finite-state systems, whereas the decidability of hhpb has been a renowned open problem for several years; only recently it has been shown undecidable. The main aim of this thesis is to gain insights into the decidability problem for hhpb, and to analyse when it coincides with hpb; less technically, we might say, to analyse the power of the interplay between concurrency, causality, and conflict.

We first examine the backtracking condition, and see that it has two dimensions: the number of transitions over which one may backtrack, and the number of backtracking moves. These dimensions translate into two hierarchies of bisimilarities; we find that both of them are strict, and that each of their levels is decidable.

Our second approach is to analyse which behavioural properties of concurrent systems are crucial to the increased power of hhpb. After establishing a minimum of behavioural situations necessary to keep hpb and hhpb distinct, we study two aspects of the interplay of causality, concurrency, and conflict: three synchronization witness (SW) situations, and the notion of confusion. With the help of a composition and decomposition result we prove that in their entirety the SW situations are essential for non-coincidence (for bounded-degree systems). However, we show this is not so for confusion, which disproves the long-standing conjecture that hpb and hhpb coincide for confusion-free systems.

We continue by studying two structural system classes with promising behavioural properties. First we consider basic parallel processes (BPP), with a suitable partial order semantics. These systems are infinite-state, but they restrict synchronization. Using the tableau technique, we prove the decidability and coincidence of hpb and hhpb for simple BPP (SBPP). The two bisimilarities

do not coincide for the complete BPP class, but we separately achieve decidability of both (a known result for hpb, but not for hhpb).

The second structural class is (safe) free choice systems, an important class in Petri net theory. These systems have a controlled interplay of concurrency and conflict, and thereby exclude confusion. Having shown that hpb and hhpb do not coincide here, we identify another interesting candidate: live strictly state machine decomposable (SSMD) free choice systems. For this class, we prove that an auxiliary bisimilarity satisfies a restricted backtracking property. As a consequence we achieve the coincidence of hpb and hhpb for a subclass of live SSMD free choice systems: the only known positive result for a class with a reasonable amount of interplay between concurrency, causality, and conflict while still admitting considerable nondeterminism.

Acknowledgements

First of all, I would like to thank my supervisor Julian Bradfield. Julian has triggered my interest in concurrency theory during his lecture 'Communication and Concurrency'. I am very grateful for his encouragement and support of my undertaking PhD studies. During the project Julian has created a supportive and stimulating atmosphere in which to work while giving me the freedom to develop and pursue my own line of research. With the hhp bisimilarity problem he has introduced me to a topic that has held my fascination ever since.

I am grateful to my examiners Javier Esparza and Mogens Nielsen for their careful reading of the thesis. Their comments and suggestions have helped to improve the final version considerably. I also thank Colin Stirling and Alex Simpson. Colin has provided crucial guidance at various stages of the project; his advice has helped to bring the thesis back into line when things seemed in danger of going adrift. Alex was always ready to offer his expertise; he has given concrete technial help at various points.

I would like to express my deepest gratitude to Jane Hillston: she has given tremendous support to me during the final stage of the project, both academically and morally. I simply could not have done without her help. Jane's reading of the whole thesis was extremely valuable, and her many comments have made this a much better work.

A very important phase of my PhD studies was the time I spent at BRICS, Aarhus. I am very grateful to Mogens Nielsen and Glynn Winskel for their hospitality. I would like to thank the many people who made this visit so inspiring and instructive. I am grateful to Mogens for the many fruitful discussions on free choice nets; I thank P.S. Thiagarajan for drawing my attention to many interesting issues within true-concurrency. Furthermore, I thank Thomas Hildebrandt for his collaboration in the hierarchy work, Marcin Jurdziński for his sharing ideas and keeping me informed about developments on hhp bisimilarity, and Jesper Henriksen for pointing me to so many useful references during my first short visit. I thank the above, Oliver Möller, and many other members of BRICS for their good company, and for creating such a wonderful atmosphere.

I have contributed from many people's comments during conferences and short visits. In particular, I would like to acknowledge Ilaria Castellani, Astrid Kiehn, Richard Mayr, Anca Muscholl, Rob vanGlabbeek, and Walter Vogler; they have all, in some way, added to this thesis.

The LFCS has always proved a most inspiring place to work at. I have profited from the friendship of many of its present and former members. In particular, I thank Dilsun Kırlı, John Longley, Monika Maidl, Matias Menni, and Jitka Stříbrná for sharing many pleasant coffee and/or lunch breaks. I also thank three of them in their role as office mates.

I would like to take the opportunity here to thank Harald Greiner, who supervised my project at Hewlett Packard, Germany in 1993. It was during this time that I took root in Computer Science.

Last but far from least I thank Rob Godfrey for all the support he has given me during the last few years. His being there kept me going.

This work has been funded by several sources which I would thankfully like to acknowledge: the first three years were sponsored by a scholarship of the then Department of Computer Science, University of Edinburgh. For my long-term visit in Aarhus I received a stipend from BRICS. The remaining time I was supported by EPSRC Research Grant GR/M84763.

Declaration

I declare that this thesis was composed by myself, and the work contained in it is my own, unless stated otherwise.

The material on the first hierarchy in Chapter 3 is joint work with Thomas Hildebrandt; it is based on [FH99]. Some results of Chapter 5 were published in [Frö99].

Table of Contents

List of Figures **7**

Chapter 1 Introduction **10**

1.1 Global Context and Motivation . . . 11
1.2 Equivalences for Concurrency . . . 20
1.2.1 Equivalences for Concurrency . . . 21
1.2.2 Decidability of Equivalences for Concurrency . . . 33
1.3 Further Background . . . 35
1.3.1 Petri Net Theory . . . 35
1.3.2 Infinite-State Verification . . . 39
1.3.3 Composition and Decomposition . . . 46
1.4 Approach and Preview . . . 48

Chapter 2 Background **53**

2.1 Models for Concurrency . . . 53
2.1.1 Net Systems . . . 54
2.1.2 Labelled Asynchronous Transition Systems . . . 55
2.1.3 Unfoldings . . . 59
2.1.4 Further Restrictions . . . 60
2.2 hp and hhp Bisimilarity . . . 60
2.3 The Decidability of hp Bisimilarity . . . 64
2.4 Conventions . . . 67
2.5 Further Concepts . . . 68
2.5.1 Bisimulation Approximations . . . 68
2.5.2 Shuffle Product . . . 69

Chapter 3 Approximating hhp Bisimilarity **71**

3.1 Introduction . . . 71
3.2 A Technicality . . . 72
3.3 Bounding the Depth of Backtracking . . . 74

3.3.1 Definition: (n)hhp Bisimilarity 74
3.3.2 Strictness of the Hierarchy 75
3.3.3 Decidability of (n)hhp Bisimilarity 77
3.4 Bounding the Number of Backtrack Moves 80
3.4.1 Definition: (n)nhp Bimilarity 80
3.4.2 Strictness of the Hierarchy 81
3.4.3 Decidability of (n)nhp Bisimilarity 82
3.5 Application to the Decidability Problem of hhp Bisimilarity . . . 88
3.5.1 Retrospection 89
3.5.2 Bounded Asynchronous Systems 89
3.5.3 Systems with Transitive Independence Relation 91
3.6 Final Remarks 92

Chapter 4 The Interplay of Causality, Concurrency & Conflict 94
4.1 Introduction 94
4.1.1 A First Intuition 95
4.2 A Minimum of Behavioural Situations 97
4.2.1 Concurrency and Conflict but not Causality 97
4.2.2 (L&C)-nondeterminism 99
4.2.3 (L&C)-nondet. Conflict or (L&C)-nondet. Concurrency . . 100
4.3 Composition and Decomposition Results 104
4.3.1 Preliminaries 105
4.3.1.1 Concurrent Steps 105
4.3.1.2 Decomposed Systems 106
4.3.2 Composition Results 112
4.3.3 Decomposition Results 113
4.3.4 Consequences 120
4.4 Synchronization Witness Situations 122
4.4.1 Definition 122
4.4.2 The SW-X Situations and the Coincidence Problem 124
4.4.3 hp and hhp Bisimilarity Coincide for Bounded-degree SW-free Systems 125
4.5 Confusion 128
4.5.1 Introduction 128
4.5.2 Confusion and the Coincidence Problem 131
4.5.3 A New Kind of Confusion: Syn-confusion 132
4.6 Liveness 134

Chapter 5 Basic Parallel Processes **136**
5.1 Introduction 136
5.2 Definitions and Methodology 137
5.2.1 BPP and SBPP 137
5.2.2 Normal Forms 140
5.2.3 The Tableau Technique 142
5.2.4 Further Definitions 143
5.3 Simple Basic Parallel Processes 144
5.3.1 Partial Order Semantics 144
5.3.2 Decidability and Coincidence of hp and hhp Bisimilarity . 147
5.4 Basic Parallel Processes 153
5.4.1 Partial Order Semantics 153
5.4.2 Non-coincidence of hp and hhp Bisimilarity 158
5.4.3 Decidability of hp Bisimilarity 158
5.4.4 Coincidence of hp and Distributed Bisimilarity 165
5.4.5 Decidability of hhp Bisimilarity 166
5.5 Final Remarks 171

Chapter 6 Free Choice Systems **176**
6.1 Introduction 176
6.1.1 Approach 179
6.1.2 Realization 182
6.1.3 Synopsis 185
6.2 Background 186
6.2.1 Free Choice Systems 186
6.2.2 SMD and SSMD Systems 188
6.2.2.1 Definitions 188
6.2.2.2 Structural Observations 193
6.2.2.3 Behavioural Observations 194
6.2.3 Live Free Choice Systems 196
6.2.4 More on Free Choice Systems 197
6.2.4.1 The Computations of SM-components in Free Choice Systems 198
6.2.4.2 The Decision-Making of SM-components in SM-decomposed Free Choice Systems 200
6.2.5 Further Preliminaries 201
6.2.5.1 Place-liveness 201
6.2.5.2 Paths 202

6.2.5.3 Subprocesses 204
6.2.5.4 A Framework 206
6.3 hp and hhp Bisimilarity do not Coincide for Free Choice Systems 208
6.4 cp and (c)hcp Bisimilarity 209
6.4.1 Introducing cp and (c)hcp Bisimilarity 209
6.4.2 Relation to hp and (c)hhp Bisimilarity 218
6.4.3 Further Concepts and Observations 219
6.4.4 Proof Methodology 222
6.5 Interlude I 225
6.6 On the Behaviour of Live SSMD FC Systems 226
6.7 On Links and Wedges 232
6.7.1 Links 232
6.7.2 Wedges 241
6.7.3 The WNL Theorem 246
6.8 The cp Transition System on SSMD fc Systems 247
6.9 Is cp Bisimilarity K-decomposable? 256
6.10 swfsi Matching in cp Bisimilarity is Deterministic 265
6.10.1 Definitions and First Insights 266
6.10.2 The SWFSI Matching Lemma 271
6.10.3 The SWFSI Matching Theorem 273
6.10.4 The SWFSI Prediction Theorems 273
6.11 cp Bisimilarity is K-decomposable, sw-(1)coherent, and sw-(1)-hereditary 275
6.12 Interlude II 279
6.13 A Coincidence Result for cp, hcp, and chcp Bisimilarity 280
6.14 Results for hp and (c)hhp Bisimilarity 285
6.14.1 bp Bisimilarity 285
6.14.2 Overcoming the Gap between bp and cp Bisimilarity . . . 290
6.14.3 Overcoming the Gap between hp and bp Bisimilarity . . . 295
6.14.4 Final Results 298

Chapter 7 Final Remarks **300**
7.1 Summary and Conclusions 300
7.1.1 Summary 300
7.1.2 Main Conclusions and Future Work 306
7.1.3 Shortcomings and Future Work 310
7.2 The Undecidability of hhp Bisimilarity 312
7.3 General Outlook and Application 314

Bibliography **317**

Appendix A Relating to Chapter 3 **330**

Appendix B Relating to Chapter 4 **334**

B.1 Event Structures 334

B.2 Proof of Theorem 4.4.2. 334

Appendix C Relating to Chapter 6 **339**

C.1 Some Intuition 339

C.2 Diamond Interrelations 343

C.3 Appendix to Section 6.6 347

C.4 Appendix to Section 6.14 348

C.4.1 Relating to Section 6.14.2 348

C.4.2 Relating to Section 6.14.3 350

List of Figures

1.1 $P = a \,||\, b$ and $Q = a.b + b.a$ are represented by the same transition system . . . 12

1.2 P and Q are represented by distinct Petri nets . . . 12

1.3 G and H are trace equivalent but not bisimilar . . . 22

1.4 [vGG89b] K and $L = (a \,||\, b) + a.b$ are pomset bisimilar but not hp bisimilar . . . 23

1.5 [vGG89a, vGG01] E and F are pomset bisimilar and wh bisimilar but not hp bisimilar . . . 25

1.6 [DD90] The causal tree captures the causal behaviour of both system A and system B of Figure 1.7 . . . 27

1.7 [NC94] Counter-example 1: A and B are hp bisimilar but not hhp bisimilar . . . 27

1.8 [Bed91] Counter-example 2: A and B are hp bisimilar but not hhp bisimilar . . . 29

1.9 [Che96] A and B are hhp bisimilar but not chhp bisimilar . . . 31

1.10 Equivalences for concurrency . . . 32

1.11 Decidability and complexity of equivalences for concurrency on finite-state systems . . . 33

3.1 Two nets N and N' that are (n)hhp but not (n+1)hhp bisimilar . 76

4.1 Counter-example 3, System S . . . 102

4.2 Counter-example 3, System S' . . . 102

4.3 hp bisimilarity is *not* decomposable with respect to the set of prime components in general . . . 119

4.4 t_s is a synchronization at $\{p_1, p_2\}$ . . . 122

4.5 Illustration of the three SW situations . . . 123

4.6 A cis-decomposable system . . . 126

4.7 The two basic cases of confusion . . . 130

4.8 Counter-example 4: a more compact counter-example that demonstrates non-coincidence for confusion-free systems 132
4.9 The basic case of syn-confusion 134
4.10 A live version of system A of Figure 1.7 135

5.1 Transition rules relative to a BPP defining system Δ 139
5.2 The Petri net representation of $\mathcal{E}$ 145
5.3 The unfolding of $PN(\mathcal{E})$, and thus $\mathcal{E}$ 147
5.4 Tableau rules relative to two SBPP in SNF $\mathcal{E}$, $\mathcal{F}$ 151
5.5 The net fragments of Δ 155
5.6 The unfolding of $\mathcal{E}$ 157
5.7 Distributed transition rules relative to a BPP defining system Δ 159
5.8 Tableau rules for hp bisimilarity relative to two BPP in ENF $\mathcal{E}$, $\mathcal{F}$ 164
5.9 Let $\mathcal{N}_1$, $\mathcal{N}_2$ be two net systems. The figure gives conditions for a set $R \subseteq \mathcal{P}(T_1 \times T_2)$ 168
5.10 Let $\mathcal{E}, \mathcal{F}$ be two BPP in ENF with $E_0 \in ENF(\Delta_{\mathcal{E}})$, $F_0 \in ENF(\Delta_{\mathcal{F}})$. The figure gives conditions for a set $R \subseteq \mathcal{P}(T(E_0) \times T(F_0))$ . . . 170
5.11 Tableau rules for hhp bisimilarity relative to two BPP in ENF $\mathcal{E}$, $\mathcal{F}$ 170
5.12 Summary of the main results 172
5.13 A proper-comm free net system that has no BPP representation . 172
5.14 The BPP spectrum 173
5.15 Results and conjectures for net sytems and lats' 174

6.1 Overview of the modules and their logical interdependence 184
6.2 Closing the remaining gaps 185
6.3 Allowed and excluded substructures of free choice nets 186
6.4 A free choice system that is SMD but not SSMD 190
6.5 A SSMD fc system 191
6.6 A SSMD fc system, which is prone to deadlocks 199
6.7 A trivial counter-example 219
6.8 A simple link λ of the system in Figure 6.5 235
6.9 An indirect link λ of the system in Figure 6.5 236
6.10 A proper wedge W of the system in Figure 6.5 243
6.11 Examples of nondeterminism admitted (transition t_1) and disallowed (transition t_2) by the psd restriction 281
6.12 An example of the two substructures allowed at the postset of a transition in buffered fc nets 290

6.13 Examples of nondeterminism admitted (transition t_1) and disallowed (transition t_2) by the spsd restriction 296

7.1 Classes with restricted nondeterminism 301
7.2 Classes with tree-like behaviour 301
7.3 The free choice spectrum 302

Chapter 1

Introduction

Recent decades have seen a tremendous growth in the deployment of computer systems. With their advance our dependence on hardware and software has increased, and so has our vulnerability to their failure. Theoretical computer science aims to model and understand the complexity of computer systems, and thereby creates the basis for their formal verification: to mathematically prove that a system satisfies its specification.

Designing software without the use of formal methods is a bit like building a bridge without having verified its statistics: both can collapse. The difference is that software can be debugged; that is tested and repaired (while this would be an expensive exercise in the construction of a bridge!). In many applications verification by testing is adequate. However, testing can only detect errors, not prove their absence. If an application is safety-critical, or cost-intensive, then there is a strong rationale for formal verification. The crash of Ariane 5 in 1996, caused by software error, is a case in point. Furthermore, with the increasing complexity and internetworking of computer systems, formal methods may be the only means to retain control over our artefacts. The most remarkable thing about the Millenium Bug was not the various disturbances that it actually caused, but the uncertainty over its possible consequences. This shows that to a certain degree we have already lost control.

Concurrency theory addresses the phenomenon that many computer applications involve a high degree of concurrency: examples are digital circuits, networking, and multi-processing. Computation is then about the ongoing behaviour of a number of interacting processes. Since the complexity of concurrent systems is high they are prone to failure, and formal verification becomes particularly important. Formal methods are most convenient to use when they can be run *automatically*, i.e. by a computer program. Then a pre-condition is the classical question of computability: can the verification problem in principle be computed.

(A second pre-condition is tractability: can the problem practically be computed considering limitations of space and time.) We speak of decidability instead of computability if a problem calls for a yes or no answer. One way to approach a difficult decidability problem 'from below' is to compare it to a related problem which is known to be decidable.

This thesis is concerned with a key notion in concurrency theory: *hereditary history preserving (hhp) bisimilarity.* The concrete aim is to gain insights into the decidability problem of hhp bisimilarity, and to analyse when it coincides with the weaker *history preserving (hp) bisimilarity.* hhp bisimilarity is an equivalence for *true-concurrency*: it reflects the interplay between *causality*, *concurrency*, and *conflict* in great detail. More abstractly, we might therefore say, the aim of this thesis is to investigate the complexity that arises from mixing the three fundamental situations of concurrent systems. It is the premise of the thesis that such an investigation contributes to a theory of true-concurrency, and it is hoped that there will be benefits for automatic verification.

The remainder of this introduction is organized as follows. In Section 1.1 we motivate the decidability and coincidence problem of hhp bisimilarity within the global context of true-concurrency and true-concurrency in verification. In Section 1.2 we give an introduction to equivalences for concurrency, in particular concentrating on hp and hhp bisimilarity. Our discussion will lead us to a basic theme: *true-concurrency versus causality.* This is one of two themes which will provide guidance throughout the thesis. Further, we cover decidability and complexity issues of equivalences for concurrency in the finite-state world. In Section 1.3 we introduce two areas which we will draw upon and the second guiding theme: the areas *Petri net theory* and *infinite-state verification*, and the theme *composition and decomposition.* Finally, we provide an overview of the thesis.

1.1 Global Context and Motivation

Interleaving versus True-Concurrency. In concurrency theory one can distinguish between two fundamental viewpoints. In the *interleaving approach* concurrency is equated with nondeterministic sequentialization: $P = a \,||\, b$ is unified with $Q = a.b + b.a.$[1] A concurrent system is then straightforwardly represented by a set of states and a set of labelled transitions between them, *i.e.* by a *labelled transition system.* This has the advantage that intrinsic connections with automata theory can be exploited. The disadvantage is that concurrency and

[1] As usual, '$||$' denotes parallel composition, '$+$' nondeterministic choice, and '$a.P$' means 'a then P'.

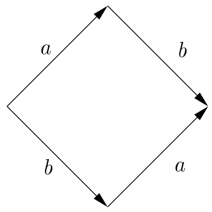


Figure 1.1: $P = a \,||\, b$ and $Q = a.b + b.a$ are represented by the same transition system

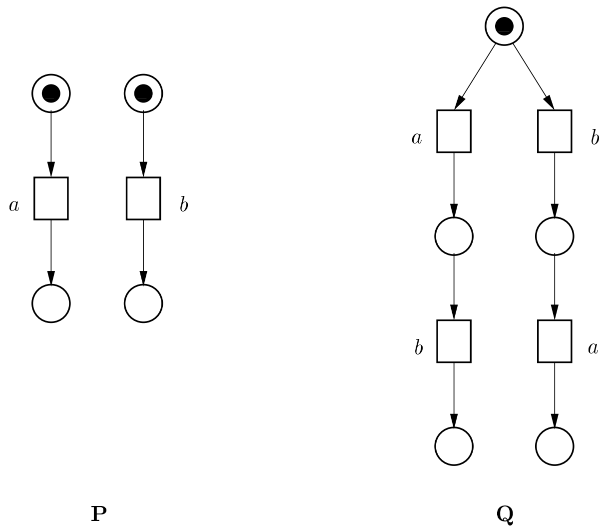


Figure 1.2: P and Q are represented by distinct Petri nets

nondeterminism become inseparable (cf. Figure 1.1).

The latter aspect is remedied by the *truly-concurrent approach*. Here, the models are able to distinguish between the three fundamental situations of concurrency: two actions can be in *conflict* with each other, they can be *causally dependent*, or occur *concurrently*. Common to true-concurrency models is that they have additional structure which shows when two transitions are independent of each other; the independence relation can either be a primary notion, as in the models *labelled asynchronous transition systems* (*lats'*) and *labelled transition systems with independence* (*tsi's*), or be derived from a locality of states and transitions, as in traditional *Petri nets* (cf. Figure 1.2). One aspect of true-concurrency models is that they capture *causality*: runs are not only viewed as sequences of transitions, but associated with each run r is a partial order that expresses how the transitions of r are causally related. A further, perhaps more fundamental, aspect is that true-concurrency models have an inherent notion of *event*: given two runs r_1, r_2 and two transition occurrences t_1 on r_1 and t_2 on r_2 it is possible to tell whether t_1 and t_2 are the same modulo independent behaviour. This ensures that causality, concurrency, and conflict are global concepts: we can recognize how any two transitions of a system are related with respect to these basic situations; causality and concurrency do not have to be interpreted with respect to a particular run of the system.

A drawback of true-concurrency models is that they do not have as clean a mathematical theory as the transition system. Some help has been provided by *trace theory*, which starts out from the partial commutation aspect of concurrency: if two transitions t_1 and t_2 are independent then the run t_1t_2 can be identified with t_2t_1; execution sequences that are the same modulo such identifications are grouped into equivalence classes called *traces*. The basis for this is given by the *first axiom of independence*: whenever two independent transitions occur consecutively they can also occur in the opposite order. For the systems we consider, the *second axiom of independence* also applies: whenever two independent transitions are enabled at the same state then they can also occur one after the other.

Verification Paradigms. There are two primary paradigms for the verification of concurrent systems: *behavioural equivalences*, to compare an implementation against a coarser system that acts as its specification, and *temporal logics*, to specify properties and verify whether these are satisfied by an implementation. Orthogonally to the dichotomy 'interleaving versus true-concurrency' behavioural equivalences and temporal logics are classified according to whether they take a *linear-time* or a *branching-time* view. In linear-time the behaviour of a system is determined by its set of executable computation paths. In branching-time the choice structure of a system is preserved: the behaviour of a system is represented by a tree of possible futures. The equivalences and logics of the interleaving world are classical: Milner's *bisimilarity* is *the* branching-time equivalence, *LTL* is the standard linear-time logic, and *CTL* the standard among branching-time logics. Many attempts have been made to design variants of these notions based on the true-concurrency paradigm: for example, *hp* and *hhp bisimilarity* are both truly-concurrent versions of Milner's bisimilarity, *ISTL*, and CTL_P generalize LTL, and CTL respectively. These concepts are more expressive than their classical counterparts: while interleaving equivalences and logics can only reason about concurrent behaviour in terms of action patterns, truly-concurrent paradigms can refer to concurrency and causality explicitly. One way to achieve this higher expressive power lies in the use of backtracking or past operators; for example such operations are primary in hhp bisimilarity and CTL_P. It seems that this method is particularly powerful: backtracking or past operators can be employed to expose subtle aspects of the interplay between causality, concurrency, and conflict.

True-Concurrency in Verification. The temporal logic paradigm has been successfully taken up by industry: *model checking* [CGP99], i.e. automatically checking whether a finite-state system is a model of a specified formula, is now widely applied in the design of digital circuits and communication protocols — using the classical interleaving logics. The major obstacle hindering the progress of model-checking is the state explosion problem: the state space grows exponentially with the number of concurrent components in the system, which puts a limit to the applicability of the model-checking algorithms to realistic systems. In hardware verification this problem has successfully been tackled by the use of binary decision diagrams (BDDs): BDDs provide a compact symbolic representation of the state transition graph, which reflects regularities typical to the structure of a circuit. Due to this reduction technique it is now possible to verify circuits with state space up to more than 10^{120} states [CGP99].

BDDs are less useful in software verification: software lacks the regular structure of hardware, and there is typically more asynchrony between the concurrent components of a system. The latter makes state explosion an even more pressing problem in software verification. But, fortunately, it also makes software amenable to a reduction method based on an idea of true-concurrency: the assumption of the *partial order reduction techniques* [CGP99] is that if two runs belong to the same trace then often they will satisfy the same properties, in which case it is sufficient to consider only one of them. As noted in [PPH97a], "the success of partial order techniques in the domain of software verification may be compared to the success of BDDs in the domain of hardware verification".

The success of the partial order techniques has sparked off a special branch of research into true-concurrency logics: to construct new logics that are partial order robust in that a formula is either satisfied by all the linearizations of a trace or by none of them. Clearly, this would allow the exploitation of the idea behind partial order techniques in a very direct fashion. Most work so far has concentrated on the linear-time spectrum, and more specifically on finding the natural counterpart to LTL. A key position is held by the logic *LTrL* [TW02]: LTrL is equal in expressive power to the first order theory of traces (while LTL has the same expressive power as the first order theory of sequences), which also implies that LTrL exactly captures the properties that are expressible by partial order robust formulae in LTL.

There seems to be a consensus in the model checking community that all interesting properties can be expressed by interleaving logics, that the higher expressiveness of true-concurrency logics is not needed. However, their higher expressiveness might be crucial in areas of verification other than the traditional fields of model-checking. [Pen93] gives examples of important properties that cannot be expressed by the classical interleaving logics: inevitability under concurrency fairness assumption, serializability of database transactions, causal successor, or the parallel execution of program segments. In particular, true-concurrency is of primary interest in the following areas of verification.

In *automatic synthesis* the task is to build a system automatically from a specification such that the correctness of the system is guaranteed by construction. This approach has much potential: unlike model checking automatic synthesis could eliminate the expensive development cycles of error detection followed by error correction altogether. Early work on automatic synthesis goes back to the 80s: seminal papers are [MW82] and [CE82]. Indeed, [CE82] is one of the two papers in which model-checking is pioneered. However, despite of its potential usefulness and in contrast to the success of model checking, automatic synthesis has never taken off. In order to offer an explanation it has been argued that early synthesis work suffers from two limitations [PR90]. In [CE82] and [MW82] a specification consists of a property formulated in one of the interleaving logics, and the synthesis algorithms extract a correct system from a tableau-based proof that the specification is satisfiable. The first criticism is that this method only applies to closed systems, where a cooperative environment is assumed. To expand synthesis to real-life applications, synthesis should generate open systems, which provide a strategy that will win against any hostile behaviour of the environment.

The second, perhaps more elemental, limitation originates from the fact that the synthesis problem has been formulated and studied in an interleaving framework: consequently the generated systems consist of a single process. As noted in [PR90] “this is particularly embarrassing in cases that the problem we set out to solve is meaningful only in a distributed context, such as the mutual exclusion problem, and a centralized single module solution does not seem very relevant.” An ad hoc way to overcome this problem is to decompose the sequential system after it has been synthesized. However, this method is incomplete and may lead to unnatural solutions. The more natural solution, which avoids the problem altogether, consists of considering the synthesis problem in a true-concurrency setting from the outset. The idea is to integrate into the specification information about the architecture of the system. This can either be done in a direct way:

then the input to the synthesis problem consists of two parts, the specification of the architecture besides the usual temporal property; or indirectly via the use of a true-concurrency logic: the input will still consist of a temporal formula, but now formulated in a true-concurrency logic: the higher expressiveness of these logics can be employed to specify properties about the internal structure of a system, e.g. to express that two activities must be independent of each other. Examples of the direct approach are the papers [PR90, KV01, MT02, ŞEM03], while the indirect approach is advocated in [PP90, Pen92]. However, the move to true-concurrency does not come without cost: these works are littered with undecidability and intractability results.

As advocated in [Bra] a further application of true-concurrency is the field of *fault analysis*. An interleaving model draws no distinction between temporal ordering and actual causal dependencies. Thus it is not possible to tell whether two events occur one after the other because the second is dependent on the first or whether this order is purely coincidental. Why drawing this distinction is essential is underlined by an example presented in [Bra]: "In the recent Ariane 5 crash, the active Inertial Reference System failed after the backup IRS failed, but this certainly does not mean that the failure of the backup caused the failure of the active system: in fact, both failed owing to a common cause, and the temporal ordering is just an artefact."

In recent years, true-concurrency has received much attention in the form of *Message Sequence Charts* (MSC's). They are close to the sequence charts of the Unified Modelling Language (UML), the now standard modelling language in software engineering, and have become popular in the automatic verification of communication protocols. Since MSC's have a partial order semantics true-concurrency is primary here. This has led to new model checking problems, and interesting decidability and complexity results [AY99, MP00].

A classical area where true-concurrency plays a crucial role with respect to behavioural equivalences is *action refinement* [BGV91, vGG01]. It is popular to design and verify concurrent systems in top-down fashion by stepwise refinement: one starts off with a high-level design where actions can represent complex processes, and refines this coarse model stepwise by replacing the high-level actions by concrete implementations at a lower design level. For example, in Petri net theory action refinement means replacing all the transitions of a specific label by some net fragment. Naturally, one would expect that the behaviour of a refined system is fully determined by the behaviour of the unrefined system and the respective refinement operation. In other words, one would require that whenever two behaviourally equivalent systems are refined in the same way then the resulting systems are still behaviourally equivalent: behavioural equivalence should be preserved under action refinement.

It was already observed in [Pra86] and [Lam86] that the interleaving approach is not sufficient when one is concerned with actions at different levels of abstraction. "A serious difficulty with the interleaving model is that exactly what is interleaved depends on which events of a process one takes to be atomic"[2] [Pra86]. This is highlighted by the following standard example (e.g. [vGG01]): under interleaving semantics the behaviour $P = a \,||\, b$ is identified with its interleaved behaviour $Q = a.b + b.a$. However, if a is refined to $a_1.a_2$ then P intuitively admits the interleaving $a_1.b.a_2$, while this is not the case for Q. It follows that interleaving equivalences are *not* preserved under action refinement.

Consequently, [Pra86] advocates the use of a more faithful view of concurrency, suggesting that when moving to partial order semantics the assumption of action atomicity is no longer necessary: "In the partial-order model what it means for two events to be concurrent does not depend on the granularity of atomicity."[2] Indeed, this could be confirmed: in linear-time, action refinement is preserved by *pomset trace equivalence* [CDMP87, vGG01], while in the branching-time spectrum it turned out that hp bisimilarity is the coarsest partial order equivalence that preserves action refinement. The latter has been proved in [vGG89a, vGG01] for event structures, and generalized to Petri nets in [BDKP91]. This result has given hp bisimilarity its prominent place among behavioural equivalences. Being a strengthening of hp bisimilarity, naturally, hhp bisimilarity is also preserved under action refinement.

Rationale for True-Concurrency. Besides any practical needs, researchers and practitioners alike have always been attracted by ideas of true-concurrency. The interleaving approach might be supported by nice and simple mathematics, but it does not adequately reflect the nature of a concurrent world. As noted in [PPH97a], "only when one imagines each and every event in the universe lining up to take its turn can one confidently apply any of the sequential models". This, of course, is not our intuition. The more natural a model is the more flexible and easier it is to work with, — and the more acceptance it will gain among practitioners. The traditional use of Petri nets as the modelling language in a great variety of practical applications is a case in point, and so is the more recent success of MSC's in the design of communication protocols. Considering

[2] Cited similarly to [vGG01].

temporal logics, Laroussinie and Schnoebelen address the question of practical expressiveness [LS95]: they find that many important properties can more directly and naturally be expressed in logics with past operators than in the standard interleaving logics. Improved practical expressiveness can also be expected from a move to true-concurrency logics. Finally, as suggested in [PPH97b], unnatural models are more likely to break down than natural ones when one slightly varies or generalizes the characteristics of the scenario under study; — a point which is perfectly exemplified by action refinement. Altogether, a remark of Pratt sums up [PPH97b]: "having to think about systems in terms of their interleaving is like trying to do arithmetic with Roman numerals. Yes, Roman numerals indeed code integers, and furthermore the algorithms for adding and multiplying Roman numerals do work, but that's not a great reason to stick with Roman numerals". Perhaps the time is ripe to leave the Roman numbers behind us, and to turn to more faithful models of concurrency — at least whenever this is feasible.

The Hardness of True-Concurrency. The equivalences and logics of the interleaving world are feasible. In particular, when the systems are finite-state these notions can in principle be decided by exhaustive search. Furthermore, tractable algorithms have been developed: classical bisimilarity can be decided in polynomial-time [KS90], model checking LTL is linear in the state space although PSPACE-complete in the formula (but usually the formulae are small), and model checking CTL is polynomial in both the model and the formula (cf. [CGP99]). Recent efforts have concentrated on tackling the state explosion problem, or on tackling classes of infinite-state systems. In the true-concurency world decidability and complexity issues are not as well understood. Equivalences and logics such as hp and hhp bisimilarity, CTL_P, and ISTL, are based on 'history information', which means that even in the finite-state case one has to deal — at least a priori — with infinite spaces. Further, if the notions are sufficiently fine no obvious quotienting will be available; then the concepts are usually both difficult to tackle and computationally hard. In particular, one problem remained open for a long time: it has only recently been resolved that hhp bisimilarity is undecidable [JN00]. Other negative results are reported in [PK95] and [Pen92]: satisfiability of CTL_P, and respectively ISTL, are shown undecidable. Even if the concepts are decidable, they seem to be computationally hard: hp bisimilarity is decidable [Vog91] but DEXPTIME-complete [JM96]; model checking CTL_P is NP-hard [PK95]; and even LTrL, which was designed with partial order reduction methods in mind, was recently shown to be non-elementary [Wal98].

The negative trend continues in automatic synthesis. [PR90] proves that the synthesis problem of distributed open reactive systems is undecidable. [MT98] addresses the related problem of synthesizing controllers for discrete event systems: in an interleaving setting this problem is decidable and can be computed in polynomial-time, but in a truly-concurrent setting the problem is undecidable. Even in the setting with local specifications, the distributed controller synthesis problem is undecidable for almost all architectures [MT01]. Some positive results for restricted architectures have been obtained [PR90, KV01, MT02], but the algorithms are of high complexity: [PR90, KV01] and [MT02] report nonelementary, and respectively doubly exponential, complexity.

On the positive side, there are two matters to record. In the context of synthesis Kupferman and Vardi put forward that the high complexity results do not necessarily present a limitation [KV01]. They argue that when looking behind the complexity measure synthesis is not any harder than interleaving verification. In general, the higher complexity of truly-concurrent concepts may be attributed to the fact that a truly-concurrent measure may, in general, amount to an exponential compression of the corresponding interleaving state space. Indeed, such considerations apply to the complexity of hp bisimilarity (cf. Section 1.2.2). The second matter concerns the area of infinite-state verification. Here a positive trend emerges from the works [EK95] and [SN96]: although in the finite-state world truly-concurrent problems are typically harder than their interleaving counterparts, [EK95] and [SN96] give examples of the converse holding for standard classes of infinite-state systems. The same phenomenon has recently been established by the complexity results of [Las03] and [Jan03].

However, some of the undecidability results indicate that true-concurrency is fundamentally hard. At the bottom of the undecidability proofs of [MT98] and hhp bisimilarity [JN00] is the insight that finite-state concurrent systems have the power to encode tiling systems. This entails that in their unfolding structure, where the interplay between the basic situations, causality, concurrency, and conflict, is visible, concurrent systems are almost Turing powerful; at this level they can simulate counter machines in a weak sense. This further motivates hhp bisimilarity as a key concept in developing and refining our understanding of true-concurrency. By characterizing what exactly makes hhp bisimilarity intractable we can hope to discern the power of concurrent systems. One phenomenon that was experienced in the development of this thesis, and which is also experienced in Petri net analysis (cf. Section 1.3.1), is that it is the interplay between causality, concurrency, and conflict that makes things difficult. In view of the undecidability of hhp bisimilarity it seems that it is the complexity of this interplay that stands behind the computational power of concurrent systems.

Rationale for an Exploration of True-Concurrency. The hardness results should not give us any reason to despair of using true-concurrency. It is their structural richness that can make truly-concurrent concepts both difficult to analyse and computationally hard. But in this richness also lies a chance: the increased structure provides a source to establish patterns, to uncover periodicity. Such insights may help us to tackle truly-concurrent problems but they may also be relevant for the interleaving view. Only if we understand concurrent systems at their most fundamental level will we be able to realize and exploit all the tools that are at hand for developing more efficient techniques, be it in terms of good algorithms for subclasses or in terms of heuristics. The partial order reduction methods are a case in point.

Given this on top of the usefulness and naturalness of true-concurrency there is a strong rationale for a deeper investigation into the hardness of true-concurrency: to bring to light when decidability or tractability can be achieved; to characterize which phenomena of concurrent systems cause problems. So far, the borderline between decidability and undecidability, tractability and intractability has hardly been investigated for true-concurrency concepts. One reason for this is, perhaps, that in true-concurrency there is no well-established hierarchy of subproblems to consider. This is in contrast to language theory or infinite-state verification where well-defined hierarchies exist. Thus, part of any analysis in true-concurrency must be to identify suitable subclasses, to develop an approach that allows for as systematic a borderline investigation as possible. Only then a unified view can be obtained.

In this thesis we will investigate the decidability problem of hhp bisimilarity, and analyse when it coincides with hp bisimilarity. Our investigation is based on, and motivated by, the following premise: (1) a systematic understanding of true-concurrency requires a systematic understanding of the interplay between causality, concurrency, and conflict, and (2) hhp bisimilarity is a key concept in studying this interplay. Thus, it is hoped that this thesis contributes a small part towards building a theory of true-concurrency.

1.2 Equivalences for Concurrency

Behavioural equivalences play a fundamental role in the theory and verification of concurrent systems: they provide the means to identify when two concurrent systems exhibit the same behaviour with respect to a certain viewpoint. On the one hand, this allows for abstraction of unwanted detail: concrete system models can be collected together into equivalence classes to obtain a more abstract semantics. For example, in process algebra the meaning of terms is usually considered up to some behavioural equivalence to ensure that the terms represent abstract behaviours. On the other hand, behavioural equivalences constitute a key verification paradigm: as we saw earlier, they are used to establish whether an implementation satisfies a specification.

In the following, we introduce hp and hhp bisimilarity informally in the context of other equivalences for concurrency. We also explain a theme, which will arise from our discussion: *true-concurrency versus causality*. This is one of the two themes which will provide guidance throughout the thesis. Furthermore, in Section 1.2.2 we address the decidability and complexity issues of equivalences for concurrency on finite-state systems. Results on infinite-state systems are considered separately in Section 1.3.2. In our examples we employ transition systems, lats', and 1-safe Petri nets (net systems) in an informal way. A formal introduction of these models can be found in Section 2.1. The formal definition of hp and hhp bisimilarity is given in Section 2.2.

Convention 1.2.1. Throughout the thesis we employ the following convention with respect to the labelling of transitions. Transitions are labelled by actions of $Act := \{a, b, c, \ldots\}$. In the drawings we only exhibit the transition identifiers, which are supposed to determine the labelling of the transitions: e.g. transitions a, a_1, and a_1' are supposed to be labelled by a.

1.2.1 Equivalences for Concurrency

Linear-Time versus Branching-Time. In concurrency theory, computation is about the ongoing behaviour of interactive processes. Thus, the behaviour of a concurrent system cannot be captured in terms of an input-output function, rather a behavioural equivalence will take into account the patterns of activity that can be detected by observing or experimenting with a system. Then, the crudest notion of behavioural equivalence is *trace equivalence*: two systems are considered to be equivalent iff they can perform the same sequences of actions. However, as put forward by Milner [Mil80], there is a sense in which trace equivalence is not sufficient when the systems contain nondeterminism. Consider system G and H of Figure 1.3. If we accept that nondeterministic choices are resolved irreversibly then G and H are intuitively not behaviourally equivalent even though they are trace equivalent: G will be deadlocked with respect to the b action after

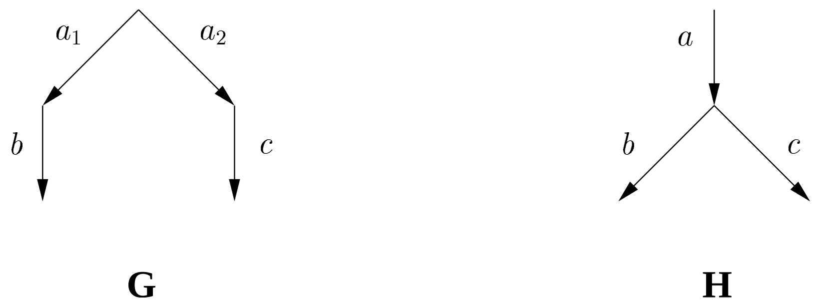


Figure 1.3: G and H are trace equivalent but not bisimilar

processing the a on the right, but such a deadlock can never arise in H. In other words, we would like to reject the distributive law $a(x+y) = ax + ay$, and take into account the branching structure of a system.

These thoughts led to the definition of *observational equivalence* [Mil80], which was later refined to the technically more elegant *bisimulation equivalence* (short: *bisimilarity*), a concept due to Park [Par81]. Two processes or states P and Q are *bisimilar* iff every action that can be performed by P can be matched by an action that can be executed by Q, and the resulting processes P' and Q' are bisimilar again; symmetrically, every action that can be performed by Q must be matched by P in the analogous way. It is easy to see that G and H are not bisimilar.

Trace equivalence and bisimilarity constitute the two extremes of the linear-time – branching-time spectrum: many intermediary notions have been defined which reflect branching to some degree. The full spectrum is reviewed in [vG01]. For us, however, it is more important to pursue an orthogonal direction: the dichotomy *interleaving versus true-concurrency*.

The Partial Order Approach. It is clear that trace equivalence and bisimilarity adopt the interleaving approach: they are defined for transition system semantics and do not reflect the higher structure of true-concurrency models in any way. Neither equivalence can distinguish between $P = a \,||\, b$ and $Q = a.b + b.a$ (cf. Figure 1.2). Consequently, many generalizations of trace equivalence and bisimilarity have been suggested to capture aspects of true-concurrency.

A first idea that comes to mind is to capture true-concurrency by allowing the observation of *concurrent steps*. This implements the intuition that whenever two transitions can happen concurrently then they can happen at the same time. *Step semantics* (cf. [vGG89a]) can distinguish between P and Q: the step $\{a, b\}$ is possible in P but not in Q. However, when it comes to mixing concurrency and causal dependencies step semantics have not much distinguishing power. Com-

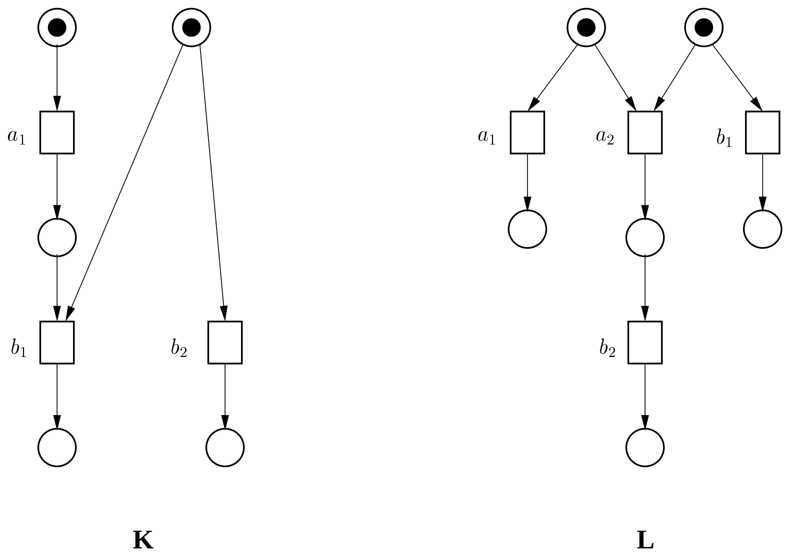


Figure 1.4: [vGG89b] K and $L = (a \,||\, b) + a.b$ are pomset bisimilar but not hp bisimilar

pare P with $L = (a \,||\, b) + a.b$ of Figure 1.4. It is not possible to distinguish P from L via step semantics [vGG89b]: in P and L exactly the same steps are possible; $\{a\}$ followed by $\{b\}$, $\{b\}$ followed by $\{a\}$, or $\{a, b\}$.

This is why it has been advocated to move to a *partial order approach* (e.g. [Pra86, CDMP87, vGG89b]): more powerfully, we allow ourselves to observe the causal dependencies between the transitions to be matched. Then it is easy to distinguish between P and L: L can do both, a independent b, and a followed by dependent b, whereas the latter is not possible in P. Since transitions with the same label may occur concurrently, technically we are dealing with *partially ordered multisets* of actions, or *pomsets* as coined by Pratt. The partial order counterpart of trace equivalence is then readily presented by *pomset trace equivalence*: two systems are *pomset trace equivalent* iff they can perform the same pomsets of actions.

There is less clarity when one moves to the branching-time world: branching and causality can be integrated in different ways, and consequently several notions of bisimilarity based on the partial order idea have been defined (cf. [vGG89a, vGG01]). The most straightforward approach is taken by the *pomset bisimilarity* of Boudol and Castellani [BC87]: this notion generalizes bisimilarity by considering transitions of pomsets rather than of single actions. However, an example of [vGG89b] demonstrates that pomset bisimilarity is not capable of capturing all subtleties that arise from branching and causality. Consider Figure 1.4. In both K and L the following pomset transitions are possible: we execute a single action a and then b is possible, or we perform a single action b and then the remaining behaviour is a, or we perform 'a independent b', or we execute 'a dependent b'. Thus, K and L are pomset bisimilar. But, intuitively, K and L do not have the same causal branching structure: in K, after each a we can choose between a causally dependent b and an independent b, whereas in L, the choice between the two options is made at the beginning: once we have executed an a it is decided whether the remaining b will follow dependently or independently of the a. It is not possible to detect this difference via pomset bisimilarity because we cannot see how new actions to match causally relate to the actions we have already matched. As in interleaving bisimilarity we move from state to state; the only difference is that we consider transitions with a more sophisticated structure, we interleave pomsets of actions rather than single actions.

History Preserving Bisimilarity. To be able to fully capture the interplay between branching and causality we must proceed as follows: we keep the history of the transitions that have already been matched, and require that this matching history grows *pomset isomorphic*. This is the idea behind *history preserving* (short: *hp*) *bisimilarity*. Technically, rather than dealing with pairs of states we keep triples (r_1, r_2, f), where r_1 is a run of the first system, r_2 is a run of the second system, and f is a pomset isomorphism relating the transition occurrences of r_1 to those of r_2. A triple (r_1, r_2, f) is contained in the largest hp bisimulation iff every transition t_1 that can be performed at r_1 can be matched by a transition t_2 that can be executed at r_2 such that f can be extended to a pomset isomorphism $f' = f \cup \{(t_1, t_2)\}$, and the triple (r_1t_1, r_2t_2, f') is contained in the largest hp bisimulation again; further, the symmetric condition must also be satisfied.

Via hp bisimilarity it is straightforward to distinguish between K and L: if a_1 is performed in L, clearly, this transition must be matched by a_1 in K. Then, in K we can execute b_1, which is dependent on a_1. But this move cannot be matched by L: only b_1, which is independent of a_1, is possible.

A more subtle example is presented in [vGG89a]; we show it in Figure 1.5. In both E and F any complete run will consist of two a actions and one b action such that the two a's are independent of each other, and the b is dependent on one and only one of the a's. The crucial difference between the two systems is: in E we can do two a actions (a_3 and a_4) and then we can choose between b that is dependent on the first a, and b that is dependent on the second a (b_3 and b_4 respectively). In F this is not possible: it will always be resolved with the second a at the latest whether the remaining b behaviour will occur dependently on the first a or on the second a. This difference is easily spotted by hp bisimilarity, but

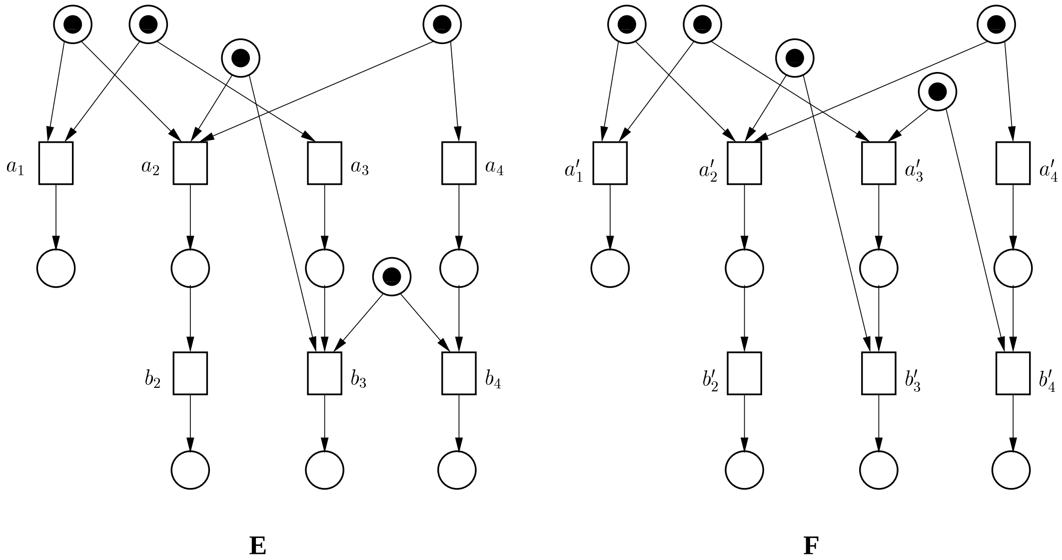


Figure 1.5: [vGG89a, vGG01] E and F are pomset bisimilar and wh bisimilar but not hp bisimilar

it is straightforward (though tedious) to check that E and F are pomset bisimilar.

Furthermore, the example demonstrates that hp bisimilarity is strictly finer than the *NMS partial ordering bisimilarity* of [DDM87, DDNM89], which is also studied as *weak history preserving* (short: *wh*) *bisimilarity*[3] in [vGG89a, vGG01]. Like hp bisimilarity, wh bisimilarity is based on pairs of runs and requires that any two related runs must be pomset isomorphic. However, wh bisimilarity does not demand that the matching history *grows* pomset isomorphic: hp bisimilarity is stronger in that with each pair of runs a particular pomset isomorphism is associated, and whenever a tuple is extended by two matching transitions then the isomorphism associated with the new tuple must be an extension of the one associated with the previous tuple. Say we match a_3 against a'_3 and then a_4 against a'_4. In hp bisimilarity this fixes the pomset isomorphism $p = \{(a_3, a'_3), (a_4, a'_4)\}$, whereas in wh bisimilarity it is sufficient to know that the two histories are pomset isomorphic. Now, let's execute b_4. In hp bisimilarity, b_4 cannot be matched by F: the only possible b action in F is b'_3; but since b'_3 is dependent on a'_3 whereas b_4 is independent of a_3, p cannot be extended as required. In contrast, with respect to wh bisimilarity b'_3 does provide a suitable match: the induced pomsets of $a_3a_4b_4$ and $a'_3a'_4b'_3$ are related by the isomorphism $p' = \{(a_3, a'_4), (a_4, a'_3), (b_3, b'_4)\}$. Had we started out by matching a_3 against a'_4 a symmetric argument would apply.

The example of Figure 1.5 also demonstrates that the combination of wh and pomset bisimilarity, *whpb bisimilarity*, is still not as strong as hp bisimilarity

[3]Note the clash of terminology: this is not to be confused with the weak hp bisimilarity that abstracts away from *silent* actions.

(cf. [vGG89a, vGG01]). On the other hand, hp bisimilarity subsumes all the equivalences we have met so far.

hp bisimilarity was first introduced in [RT88] and [DDNM89] under the name of *behaviour structure* bisimilarity, and *mixed ordering* bisimilarity respectively. The term *history preserving* results from [vGG89a], where Goltz and vanGlabbeek define the notion for event structures and prove its key property: hp bisimilarity is preserved under action refinement. This result has given hp bisimilarity a prominent place among true-concurrency bisimilarities. In [BDKP91] the notion is introduced as *fully concurrent* bisimilarity. There it is independently shown that hp bisimilarity preserves action refinement for the more general model of Petri nets. In [DD89, DD90] hp bisimilarity has also been studied as *causal bisimilarity* on the model of *causal trees*.

True-Concurrency versus Causality. So far, we have concentrated on capturing one particular aspect of true-concurrency models: their ability to model the causal relationship between transitions. In hp bisimilarity we have found an equivalence that fully reflects the interplay between branching and causality. hp bisimilarity can thus be considered to be *the* bisimulation equivalence for causality. However, as described in the beginning of Section 1.1, a further aspect of true-concurrency models such as lats' or net systems is that they induce a notion of *event*: given two interleaved runs r_1, r_2 and two transition occurrences t_1 on r_1 and t_2 on r_2 it is possible to tell whether t_1 and t_2 are the same modulo independent behaviour, that is whether t_1 and t_2 present the same event. This means true-concurrency models depart from the interleaving approach in a fundamental way: the unfolded behaviour of a concurrent system is no longer represented by a tree-like structure but the various computation branches are interlinked: when two computations are the same modulo shuffling of independent transitions they are considered to join together in a common 'state'. (Technically, unfolded behaviour is now represented by an *occurrence tsi or lats*, or — making the notion of event primary — by an *event structure*.) Note that this view of unfolded behaviour is accompanied by the idea that history can be traced back in different ways, reflecting that independent transitions can be shuffled in their order.

There are models which fully capture causality and branching while still taking a tree-like approach. Such a model is, for example, provided by the *causal trees* of [DD89, DD90]: causal trees are tree-shaped labelled transition systems where a label consists of an action and a set of backwards pointers; the backwards pointers are understood to identify those arcs which caused the respective arc.

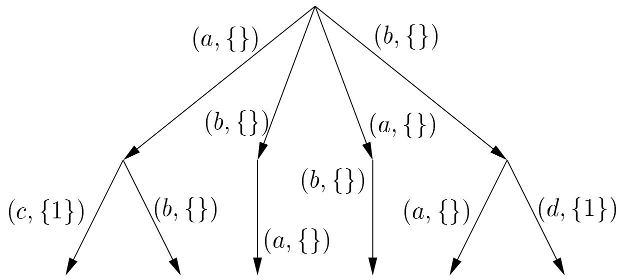


Figure 1.6: [DD90] The causal tree captures the causal behaviour of both system A and system B of Figure 1.7

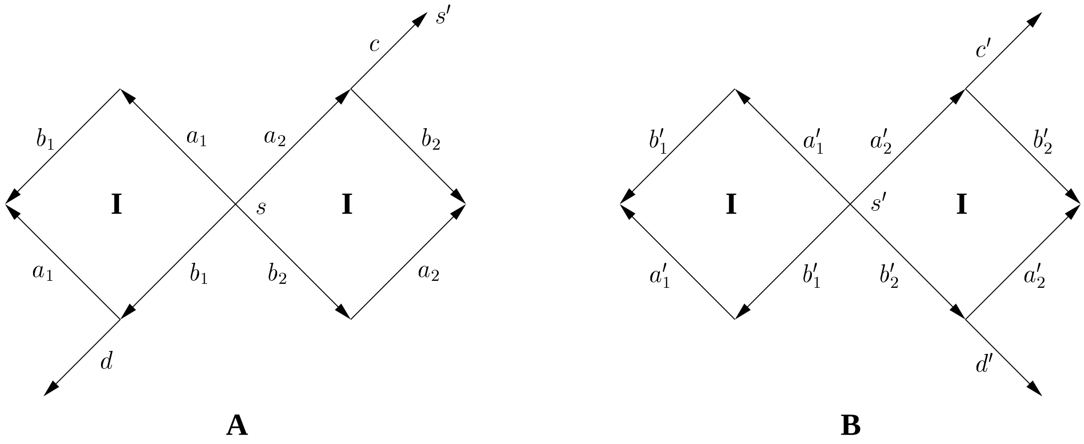


Figure 1.7: [NC94] Counter-example 1: A and B are hp bisimilar but not hhp bisimilar

Figure 1.6 gives an example of a causal tree. In models for causality such as causal trees independence and concurrency are only meaningful when interpreted with respect to a branch; to be able to recognize how any two transitions of a system are related with respect to causality, concurrency, and conflict we require the notion of event. Consequently, causality models are too abstract to capture all aspects of the interplay between causality, concurrency, and conflict. Indeed, the causal tree of Figure 1.6 represents the causal behaviour of both system A and system B of Figure 1.7.

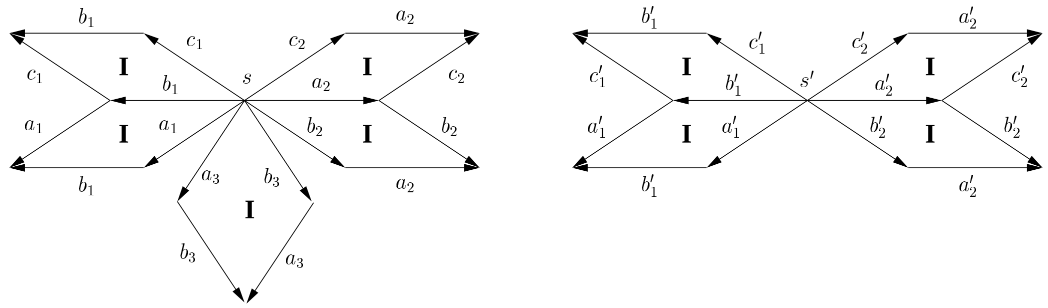
Every notion of bisimilarity we have discussed so far could have been defined on a model for causality just as well: none of them makes use of the extra structure associated with the aspect of event. Consequently, hp bisimilarity is not able to capture subtleties of the interplay between causality, concurrency, and conflict that rely on recognizing transitions as the same event. We demonstrate this with the help of the standard counter-example of [NC94], which shows that hhp bisimilarity is strictly finer than hp bisimilarity.

Counter-example 1, Part 1 (Figure 1.7). Both A and B have an a transition (b

transition) that can be followed by dependent c (d) or alternatively by independent b (a). And both systems can perform an a transition (b transition) which can be followed by independent b (a) as the only option. The crucial difference between the two systems is: in A the a and b on which c and respectively d are dependent are in conflict with each other, whereas in B the respective a and b are independent of each other. In hp bisimilarity this difference can be hidden by adopting the following strategy: if we execute a_2 as the first transition we will choose a_2' as its match: this will take care of the c option. We will then have to match b_2 against b_2'. But this is no problem: the d' transition is not visible since it is disabled by a_2'. On the other hand, if we perform b_2 as the first transition we will match it against b_1' avoiding the d' transition. The c transition is already hidden, and we can safely match a_2 against a_1'. The remaining cases can be dealt with in the same spirit.

Hereditary HP Bisimilarity. *Hereditary history preserving* (short: *hhp*) *bisimilarity* does exploit the aspect of event. Technically, hhp bisimilarity is obtained from hp bisimilarity by the addition of a *backtracking* requirement: for any two related runs, the runs obtained by backtracking a pair of related transitions, must be related, too. We allow backtracking not only in the order which is laid down by the related runs; as long as a pair of transitions is maximal in the associated pomset isomorphism, it can be backtracked. This takes into account the first axiom of independence: whenever independent transitions occur consecutively they can also occur in the opposite order. Ultimately, the backtracking requirement ensures that the matching is not dependent on the order in which independent behaviour is linearized: if we first match a transition t_1 against a transition t_2 and then a sequence of transitions w_1 against a sequence w_2 such that $t_1\ I\ w_1$, or $t_2\ I\ w_2$ equivalently, — meaning t_1 (t_2) is independent of all the transitions in w_1 (w_2) — then backtracking requires that w_1 and w_2 provide a suitable match irrespective of whether the (t_1, t_2) match is interleaved or not. Observe that in the strategy which proves that A and B of Figure 1.7 are hp bisimilar we make the matching of the parallel a's and b's dependent on the order in which they appear in the runs to match. With the help of backtracking it is straightforward to distinguish between the two systems.

Counter-example 1, Part 2 (Figure 1.7). A and B are *not* hhp bisimilar. As we saw, the c transition dictates that we have to match a_2 to a_2', and further a_2b_2 to $a_2'b_2'$. But now the b_2 - b_2' match is no longer safe: we can backtrack the pair of a transitions and then require that the runs b_2 and b_2' are related. But from this



A **B**

Figure 1.8: [Bed91] Counter-example 2: A and B are hp bisimilar but not hhp bisimilar

point, B can perform a d transition, which A cannot match. So b_2 and b'_2 can clearly not be related runs.

A second counter-example, which demonstrates the difference between hp and hhp bisimilarity, is exhibited in [Bed91]; we show it in Figure 1.8.

Counter-example 2 (Figure 1.8). System A can be described by the expression $(a.0 + c.0 \mid\mid b.0) + (a.0 \mid\mid b.0) + (a.0 \mid\mid b.0 + c.0)$, and B by $(a.0 + c.0 \mid\mid b.0) + (a.0 \mid\mid b.0 + c.0)$. Behaviourally, the key difference between the two systems is that in A there is a pair of independent a and b transitions such that neither the a nor the b can ever occur in parallel with a c. In B no such pair of a and b exists. With respect to hp bisimilarity this difference can easily be hidden by adopting the following strategy: if a_3 occurs as the first transition we will match it to a'_1; then in both systems 'parallel b' is the only remaining behaviour, and b_3 can safely be matched to b'_1. If we start out with b_3 we will match b_3 to b'_2. Then it is safe to match a_3 to a'_2 since the c option is hidden. A and B are *not* hhp bisimilar. Their difference is readily detected by backtracking: c'_2 dictates to match a_3 to a'_1 and further a_3b_3 to $a'_1b'_1$; but at $(a_3b_3, a'_1b'_1)$ we can backtrack the a transitions, and thereby expose c'_1.

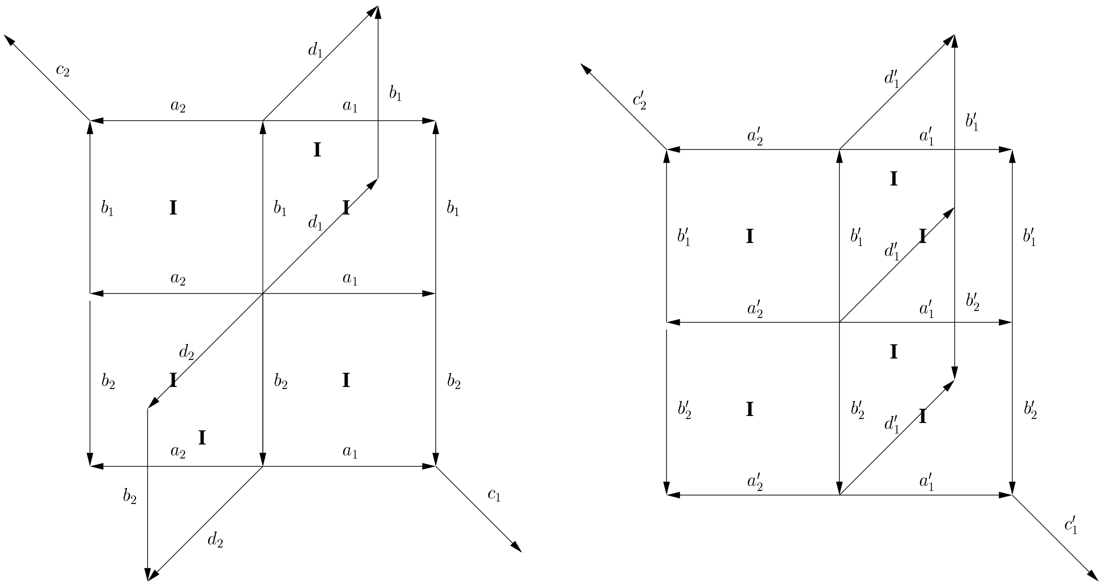
The two counter-examples demonstrate how hhp bisimilarity integrates the aspect of event. In summary, we put forward the following view, which we consider to pinpoint the source of the differing distinguishing and computational power of hp and hhp bisimilarity. Our view will provide guidance throughout the thesis.

> *hhp bisimilarity is a notion for true-concurrency whereas hp bisimilarity only captures causality.*

This characterization was first advocated in [FH99], where it is shown that hhp bisimilarity can be understood as hp bisimilarity with the added requirement of *trace-consistency*. Our view is further backed by the theory of *open maps*. In [JNW96] Joyal, Nielsen, and Winskel describe a uniform way of defining a bisimulation equivalence across a wide range of different models by applying category theory: two objects of a model category are bisimilar iff there exists a span of open maps between them, where open maps are relative to a choice of path category within the model category. For many concrete models, the abstract bisimilarity specializes to already known equivalences [JNW96, CN95]. As one would expect, for standard transition systems one obtains classical bisimilarity. For true-concurrency models such as tsi's or event structures, the abstract bisimilarity specializes to hhp bisimilarity. As shown in [Che96] the open map characterization of hhp bisimilarity is very robust with respect to the choice of path category. In particular, it is not clear how hp bisimilarity could be captured when tsi's or event structures are taken to be the model category. On the other hand, it has been found that hp bisimilarity can very naturally be characterized via open maps when a causality model such as *history dependent automata* or causal trees is employed [Pis99, Frö03]. This further suggests that hhp bisimilarity is the natural bisimulation equivalence for true-concurrency, whereas hp bisimilarity is the one for causality. In the context of the open map characterizations it is shown that a hhp bisimulation can itself be viewed as a tsi or an event structure, while a hp bisimulation can be understood as a causal tree or a history dependent automaton. This also underlines our view.

The notion of hhp bisimilarity first appears in [Bed91], where Bednarczyk studies several history preserving bisimulations with a downwards closure condition. He calls sets that satisfy this condition *hereditary*. In [JNW96], in the context of open maps, hhp bisimilarity has independently been introduced under the name of *strong* hp bisimilarity. In [NC94] logical and game-theoretical characterizations are found which come as conservative extensions of the corresponding characterizations of classical bisimilarity.

Coherent HHP Bisimilarity. There is an even stronger notion of bisimilarity, which will be of importance later on. hhp bisimilarity reflects the first axiom of independence but it does not relate to the second axiom: whenever two independent transitions are enabled at the same state then they can occur one after the other. This is remedied by the *coherent hhp* (short: *chhp*) *bisimilarity* of [Che96]. chhp bisimilarity complements the backtracking condition with a *padding* requirement: whenever in the matching there is an ‘independent branching’ in that as



A **B**

Figure 1.9: [Che96] A and B are hhp bisimilar but not chhp bisimilar

the continuation of two related runs we match on the one hand a transition t_1 against a transition t_2, and on the other hand a transition sequence w_1 against a sequence w_2 such that $w_1\ I\ t_1$ and $w_2\ I\ t_2$ then we require that (t_1, t_2) and (w_1, w_2) must also appear in the matching in consecutive order.

A counter-example in [Che96] shows that chhp and hhp bisimilarity are indeed distinct notions. The example is presented in Figure 1.9. In both systems there are two a transitions and two b transitions such that the a's are in conflict with each other but independent of the b's; and this is symmetrical for the b's. In addition, having executed a pair of diagonal a and b, e.g. a_1 parallel b_2, or a_2 parallel b_1, we can do a c. For now let us ignore the d transitions. With respect to hhp bisimilarity if an a or b occurs as the first transition it can be matched both straight across, e.g. a_1 to a_1', and also diagonally, e.g. a_1 to a_2'. The c options enforce that we have to take more care when an a or b transition occurs in second place; then our strategy must be consistent with that of the previous match: e.g. if we have matched a_1 against a_2' and now want to match b_2 we have to stick to the diagonal strategy and match b_2 to b_1'. The union of all these matches certainly provides a hhp bisimulation. In contrast, to obtain a chhp bisimulation we have to stick to one of the two strategies:[4] although (a_1, a_1') and (b_1, b_2') are both suitable matches, in consecutive order they are not.

Now, we take into account the d transitions. In both A and B each b can occur

[4]Note that this highlights an 'abnomaly' of chhp bisimilarity: chhp bisimulations are not closed under union.

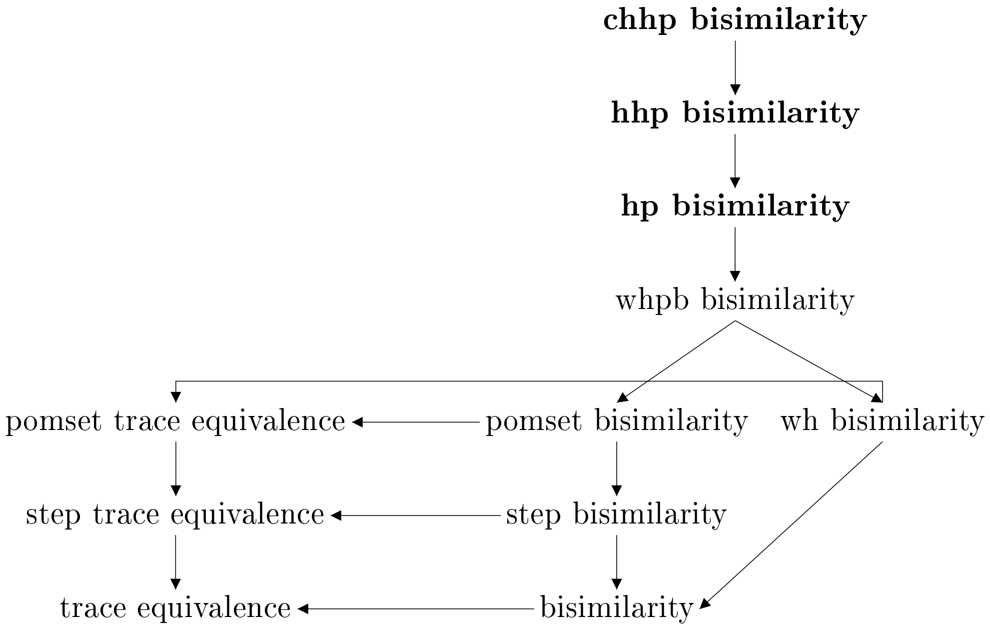


Figure 1.10: Equivalences for concurrency

in parallel with a d action, which is in conflict with the a's. The only difference between the two systems is that in A there are two conflicting d transitions, d_1 in parallel with b_1, and d_2 in parallel with b_2, whereas in B there is exactly one d, d_1', which is in parallel with both b_1' and b_2'. With respect to hhp bisimilarity it is possible to hide this difference: d_1 and d_2 can both be matched by the one d_1'. In contrast, chhp bisimilarity exposes the difference. Say we start with d_1, and necessarily match it to d_1'. At this point both b_1' and b_2' are enabled, and have to be matched against b_1. By backtracking this enforces that both (b_1, b_1') and (b_1, b_2') must be in the bisimulation. From above we know that this is no problem with respect to hhp bisimilarity, but it also means that no chhp bisimulation can be found.

Final Remarks. Figure 1.10 gives an overview of the equivalences we have discussed in this section. The partial order approach has mainly been studied in the context of action refinement. More details and examples can be found in the surveys [vGG89b, vGG89a, vGG01].

Many other approaches have been studied that model concurrent systems more faithfully than the interleaving approach, and accordingly many more behavioural equivalences have been defined. A few of them touch upon this thesis. *Timed semantics* [Hen88] and *ST semantics* [vGV87] capture the aspect of duration: as the name suggests timed semantics add a notion of time, whereas in ST semantics each action occurrence is separated out into a start section and an end section. ST

	Equivalence	Decidability/Complexity
LT	trace equivalence on transition systems	PSPACE-complete
	trace equivalence on 1-safe Petri nets	EXPSPACE-complete
	pomset trace equivalence	EXPSPACE-complete
BT	bisimilarity on transition systems	PTIME-complete
	bisimilarity on 1-safe Petri nets	DEXPTIME-complete
	hp bisimilarity	DEXPTIME-complete
	hhp bisimilarity	undecidable

Figure 1.11: Decidability and complexity of equivalences for concurrency on finite-state systems. LT ... Linear-time, BT ... Branching-time

semantics are important with respect to action refinement: like hp bisimilarity the associated ST equivalences are preserved under this operation. In process algebra, noninterleaving semantics have been obtained by taking into account the aspects of distribution and locality [Cas01]. *Distributed bisimilarity* [Cas88, CH89] is the natural bisimulation equivalence for Castellani's *distributed transition semantics*, which reflect that a concurrent process can be considered to evolve into two parts, a local and a concurrent residual. A more sophisticated approach to locality is provided by the *location semantics* of [BCHK93, BCHK94]: here process terms are syntactically enriched by locations, reflecting that concurrent processes are situated at different locations. A unified framework that enables the study of both locality and causality has been provided by the *local/global cause semantics* of [Kie94].

In the context of process algebra it is important to consider *weak* versions of equivalences, which abstract away from the *silent action* τ. This is an aspect we shall not be concerned with in this thesis.

1.2.2 Decidability of Equivalences for Concurrency

With respect to automatic verification it is important to know whether a notion of behavioural equivalence is decidable: is it decidable in general whether two systems S_1 and S_2 are equivalent under the given notion of equivalence? In this section we concentrate on finite-state systems; infinite-state verification will be considered separately in Section 1.3.2.

The equivalences of the interleaving world are well-understood. Checking trace equivalence on finite-state transition systems reduces to checking language equivalence on finite automata, which is known to be decidable by Moore's classical algorithm (see e.g. [HU79]). [KS90] studies the complexity of several equivalence problems central to interleaving process theory. Here it is shown that trace equivalence is PSPACE-complete (a result due to Chandra and Stockmeyer), which sharpens PSPACE-completeness of language equivalence [SM73]. Classical bisimilarity is easily seen to be decidable by exhaustive search: for finite-state systems bisimulations are subsets of a finite domain; consequently, it is possible to check whether one of the finitely many candidate sets is indeed a bisimulation. Moreover, efficient algorithms have been found: [KS90] achieves a polynomial-time algorithm by reduction to *generalized partitioning*. Building on an efficient algorithm by Paige and Tarjan for that problem they achieve a complexity of $O(n+m\log n)$, where n is the number of states and m the number of transitions. Futhermore, checking bisimilarity is proved PTIME-complete in [ÀBGS91]. The complexity of the classical equivalences have also been investigated for models other than transition systems. [Rab97] investigates networks of communicating finite agents; [JM96] studies finite 1-safe Petri nets. The results of these works demonstrate that in true-concurrency we may obtain higher complexities simply because truly-concurrent presentations can be more compact by avoiding state explosion: [JM96] establishes that with respect to 1-safe Petri nets trace equivalence is EXPSPACE-complete, and bisimilarity is DEXPTIME-complete. The upper bounds follow from the complexities on transition systems when considering that the transition system induced by a 1-safe Petri net is in general exponentially larger than the size of the net.

Many of the true-concurrency equivalences are also well-investigated. Among them is hp bisimilarity. hp bisimilarity was first shown to be decidable for 1-safe Petri nets by Vogler [Vog91]. An alternative proof is provided by Jategaonkar and Meyer in [JM96], where they study various behavioural equivalences on 1-safe Petri nets. The insight behind the decidability result is this: it is not necessary to keep the entire history to capture hp bisimilarity, but to see whether pomsets grow isomorphic it is sufficient to record only those events that can act as maximal causes. Vogler and Jategaonkar both develop notions that capture this essential fragment of the history in a finite way, *ordered markings* (*OMs*), and *growth-sites* respectively. Altogether, hp bisimilarity can then be decided by exhaustive search. The approach of Jategaonkar and Meyer provides a tight complexity result: hp bisimilarity on 1-safe Petri nets is DEXPTIME-complete. Via the growth-sites quotienting they also achieve decidability and complexity results for pomset trace equivalence and pomset bisimilarity: the first is EXPSPACE-complete, the latter DEXPTIME-hard and decidable in EXPSPACE. Thus, with respect to 1-safe Petri nets the complexity of hp bisimilarity and pomset trace equivalence are not higher than those of their interleaving counterparts. Vogler's OM approach has been generalized to prove decidability of weak hp bisimilarity [Vog95]; [JM96] already takes care of nets with silent actions. In a further development, hp bisimilarity has been shown decidable for n-safe Petri nets by Montanari and Pistore [MP97].

In Section 1.2.1 we have seen that although hhp bisimilarity is obtained from hp bisimilarity by the seemingly small addition of a backtracking requirement its distinguishing power is far greater. The same applies with respect to computational power. There is no straightforward adaptation of the decidability proofs of hp bisimilarity to hhp bisimilarity. In contrast, the decidability problem of hhp bisimilarity has been open for several years, renowned for its inscrutability. Some conjectures and notes can be found in [Che96]. Only recently the problem has been resolved: hhp bisimilarity is shown undecidable by Jurdziński and Nielsen in [JN00]. We will discuss this result, and put it in the context of this thesis in Section 7.2. It appears hhp bisimilarity is the only equivalence known to be undecidable for finite-state systems.

Figure 1.11 gives an overview of the decidability and complexity results.

1.3 Further Background

We now present two areas which we will draw upon, *Petri net theory* and *infinite-state verification*, and discuss their relevance for this thesis. Furthermore, we explain a second theme that will guide and continuously appear throughout the thesis: *composition and decomposition.*

1.3.1 Petri Net Theory

Petri Nets. Petri net theory was initiated by Petri in his seminal doctoral thesis of 1962. The aim was to build a theory "for the description, in a uniform and exact manner, of as great as possible a number of phenomena related to *information transmission and information transformation*" (cited following [Rei82]). Petri suggested a net-based approach: the static structure of the system under study is represented by a *net*; that is by a set of circles or *places*, a set of boxes or *transitions* and a neighbourhood relationship which shows how these two types of elements are interrelated. Places are understood to represent local states, and transitions represent local activities. The dynamics or behaviour of a Petri net are determined by an initial global state, or *marking*, and the *transition firing rule*. Markings and transition firing are subject to the principles of *distribution* and *locality of state change*: (1) Markings are composed of local states; they are distributions of *tokens* over places. (2) If a marking evolves into a new marking via a transition then the marking will be affected only locally: only the places belonging to the neighbourhood of the transition will change; the other places stay unaffected. Via these principles Petri nets induce natural notions of independence, causality, and concurrency, and a concept of event as is typical for true-concurrency models. They carefully distinguish between nondeterminism and concurrency. It is in the context of net theory that Petri identified concurrency, conflict, and causality as the fundamental situations of concurrent systems [BT87]. On the other hand, Petri nets provide a more concrete true-concurrency model than for example tsi's or lats': they maintain structural information such as concepts of locality and distribution, and the related notions of structural interaction like synchronization.

There are many different net models (cf. [Rei82]). The perhaps most commonly used type is *Place/Transition Petri nets*; often they are referred to simply as *Petri nets*, a convention we shall follow here. It is essentially this net model we have described above. Since there is no bound on the number of tokens that can be held at a place, a Petri net is potentially infinite-state. A Petri net is *bounded* if on each place the token load is limited by a natural number. In particular *safe Petri nets*, where the bound is one, are a popular system class for finite-state analysis. In the sequel we will often call them *net systems*, or simply *systems*, if the context is clear.

Petri Net Problems. The traditional verification problem for Petri nets is the *analysis problem*: given a Petri net, does it satisfy certain behavioural properties of interest? Properties of interest are for example deadlock-freedom, reachability, boundedness or safeness, and liveness. A Petri net is *live* iff every transition can always be made to occur again. The decidability and complexity of most analysis problems were settled in the late 70s and early 80s. As portrayed in [Esp98] one can intepret this as a first phase of research on computational issues of Petri nets. In other phases model-checking and equivalence problems were addressed. Furthermore, the various Petri net problems were also pursued for net systems. The surveys [Esp98] and [EN94] give thorough accounts of the results that were achieved in Petri net exploration.

Results on net systems that are relevant for us have already been reviewed in Section 1.2.2. Relevant results on equivalence checking for Petri nets will be covered in the next section, where we address infinite-state verification. However, it is work of the early analysis phase which will be of direct use in this thesis. Confronted with the difficulty of the analysis problem in the general case, the question was raised whether good analysability could be obtained for natural subclasses of Petri nets. (Indeed, all interesting analysis problems for Petri nets were later shown to be EXPSPACE-hard; cf. [Esp98].) The investigation of the analysability border led to the identification of important behavioural and structural Petri net classes together with the development of a rich *structure theory*.

Structure versus Behaviour. Structure theory studies the interplay between the behaviour of a Petri net and the structure of its underlying net. Two typical questions are: do structural restrictions correspond to behavioural situations? Can interesting behavioural properties be captured by structural properties? Accordingly, the motivation and successes of structure theory are twofold. On the one hand, it has been observed that natural constraints on the structure of the net yield natural behavioural subclasses — in a sufficient sense. For example, *T-systems*, which are also known as *marked graphs* and *synchronization graphs*, structurally capture conflict-free systems, whereas *S-systems* give rise to sequential systems.[5] The motivation is that structure is easier to work with than behaviour; structural classes allow for a very systematic investigation of the borderline of problems such as Petri net analysis.

On the other hand, structure is more efficiently analysable than behaviour: no state space exploration is required. This is the rationale behind the second aspect of structure theory: for restricted classes interesting behavioural properties are tightly coupled — in a necessary and sufficient way — to structural properties; then the analysis of the behavioural properties can be done efficiently by analysing the structure. For example, a classic connection concerning T-systems is: a T-system is live iff all of its simple cycles carry at least one token at the initial marking; and further: a live T-system is safe iff it is covered by simple cycles which carry at most one token at the initial marking [CHEP71, GL73]. Good surveys on classical structure theory can be found in [Bes87] and [BT87].

Confusion and Free Choice Petri nets. S-systems and T-systems are very basic system classes, where concurrency, or respectively conflict, is excluded entirely. It was experienced that the mixture of these two fundamental situations can make the analysis of Petri nets very difficult. In particular, one behavioural situation was identified as a source of trouble: the situation of *confusion* arises when conflict and concurrency are mixed in a specific way such that the firing of one transition can have an impact on how a conflict concerning a concurrent transition is resolved. Confusion was probably first pinpointed by Holt (as reported in [RT86]).

[5] The Petri net must also be safe and connected.

A structural restriction that yields confusion-free systems (when imposed on safe Petri nets) was found: *free choice Petri nets* were introduced in [Hac72]. Devised as a combination of S-systems and T-systems they admit both conflict and concurrency, but in a controlled fashion that disentangles the interplay of the two situations. A rich structure theory was obtained for free choice Petri nets, and they came to hold a central place in net theory: they are considered the largest Petri net class that allows for a good theory and consequently good analysability. To express that beyond free choice Petri nets 'things become troublesome' the term *free choice hiatus* has been coined [Bes87].

Classic free choice theory is due to Commoner and Hack. They came up with two central theorems. *Commoner's Theorem* [Com72, Hac72] gives a structural characterization of liveness in free choice Petri nets. This first result is complemented by Hack's *S-coverability Theorem*, which structurally captures the safeness of live free choice Petri nets. Importantly for this thesis, the S-coverability Theorem sets up a decomposition theory; it says: every live and safe free choice Petri net is covered by S-components which carry exactly one token each, where S-components are special kinds of S-systems associated with a net. Consequently, each live free choice system can be understood as a synchronization of a set of live S-systems. The two theorems are reviewed and elaborated in the surveys [Bes87] and [BT87], together with other pieces of classic free choice theory (for example, S-coverability has a dual: T-coverability).

Free choice theory has subsequently been refined and extended. A comprehensive presentation is provided by the book [DE95]. It demonstrates how the free choice hiatus could be confirmed by further developments. The free choice condition studied in [DE95] slightly generalizes the classic constraint, however it does so while maintaining the behavioural effect.

It is important to keep in mind that the term 'free choice hiatus' was coined against the backdrop of the nice structure theory of free choice nets and the EXPSPACE-hardness of all interesting analysis problems for general Petri nets. Most interesting problems for free choice Petri nets are at least NP-hard [Esp98]. For example, deciding non-liveness is a NP-complete problem [JLL77, DE95]. On the other hand, analysing liveness and boundedness in one go can be decided in polynomial-time [DE95]. Furthermore, many interesting questions about live and safe free choice Petri nets also have polynomial-time algorithms [Esp98]. On second thought, this is not surprising: it is the class of *live and safe* free choice Petri nets that enjoys deep structural properties such as the S-coverability Theorem. Thus, we note: when considering decidability or complexity problems for net systems it might be a better strategy to look for a hiatus at *live* free choice systems rather than at free choice systems. In particular, one might exploit the decomposition theory.

Relevancy. To conclude we now extract in which way the presented themes and insights will be relevant for our investigation of the two hhp bisimilarity problems.

1. Structure versus Behaviour. We will employ safe Petri nets as our model at a structural level, where interaction of local components such as synchronization is visible. hp and hhp bisimilarity abstract away from such detail but with respect to our borderline analysis it may be advantageous to maintain structural information: to be able to refer to the structure that ultimately stands behind the interplay of causality, concurrency, and conflict. As in Petri net analysis, structure is easier to work with than behaviour.

2. A Source of Interesting Subclasses. The behavioural and structural subclasses which have been identified during research on Petri net analysability seem relevant for us. We may also profit from the rich structure theory that has been established. Sequential and conflict-free systems, or S-systems and T-systems respectively, will not occupy us for long; however:

3. Confusion and Free Choice Systems. Since confusion-free and hence free choice systems keep the interplay between concurrency and conflict under control, they constitute ideal classes to consider with respect to the hhp bisimilarity problems. This is further motivated by [Che96]: here it is conjectured that hp and hhp bisimilarity coincide for free choice systems.

4. Live Free Choice Systems. As motivated by the previous paragraph *live* free choice systems provide an alternative candidate for analysing the borderline of the hhp bisimilarity problems. In particular, we would be able to exploit the rich decomposition theory induced by the S-coverability Theorem.

1.3.2 Infinite-State Verification

Language Theory. The origin of research on decidability issues for infinite-state systems can be traced back to classical language theory. Much effort has been (and still is) dedicated to approximating the decidability border for the language equivalence problem: given two language generators, do the two languages they define coincide? Due to the theoretical limits set by the halting problem this problem cannot be decidable for Turing machines in general. On the other hand, it was readily proved by Moore in 1956 that language equivalence is decidable for finite automata [Moo56]. Since then researchers have tried to determine where exactly the cut-off point between decidability and undecidability lies — considering the remaining generators of the Chomsky hierarchy of languages and grammars. Soon enough it was established that context-free languages are too expressive to allow for a decidable theory [BHPS61]. In particular, this left open the decidability of language equivalence for deterministic pushdown automata, which was to become one of the most celebrated problems in theoretical computer science. Only in 1997, after decades of effort, the problem was proved decidable by Sénizergues [Sén97, Sén01].

Infinite-State Verification under Interleaving Semantics. With the advent of process algebras and their associated behavioural equivalences it was natural to ask how these new formalisms relate to classical language theory. The starting point for this line of research can be seen in work by Baeten, Bergstra, and Klop [BBK87, BBK93]: they translated the concept of context-free grammars into the process calculus *Basic Process Algebra* (*BPA*); contrasting the negative result for language equivalence, they found that classical bisimilarity is decidable for *normed* BPA, which corresponds to context-free grammars without redundant symbols and productions. Later the result was extended to the entire class of BPA [CHS92, CHS95].

The results on BPA demonstrated how process algebra can be understood as an extension of formal language theory, and how this generalized view can lead to a refinement and improvement of known results in the area of formal languages: research on bisimilarity may expose periodicity or algebraic structure that may in itself be relevant for the language theoretic view. In particular, for certain generators, e.g. normed deterministic processes, bisimilarity and language equivalence coincide. The fruitfulness of the interplay between language and process theory is ultimately illustrated by work concerning the equivalence problem of deterministic pushdown automata: Stirling provided a simpler proof of decidability by viewing the problem as a bisimilarity problem for a process calculus, and employing the tableau technique, a method that is common in infinite-state process theory [Sti01]. This approach also establishes a primitive recursive upper bound on the complexity of the problem [Sti02].

Research on decidability issues for infinite-state systems is also strongly motivated from within concurrency theory: the success of automatic verification in the finite-state world is contrasted by the reality that in practice most systems have either an infinite or an extremely large state space. Thus, it is important to clarify: how far can the automatic methods of the finite-state world be extended to subsume infinite-state classes? It is folklore that full process calculi such as

CCS are too expressive to allow for a decidable theory. Milner pointed out in [Mil89] that CCS is Turing powerful: it can express objects such as counters and stacks, which in turn give rise to Turing powerful computational models; for example, the two counter machines of Minsky are universal. In [Tau89] it is made formal that every Turing machine can be translated into a CCS process such that the behaviour of that process exactly corresponds to the execution of the Turing machine. Consequently, the halting problem for Turing machines can be encoded as a bisimulation problem of CCS processes. In analysing the borderline of decidability a kind of Chomsky hierarchy of infinite-state transition systems emerged. Apart from BPA and pushdown automata, which provide models of sequential computation, in particular two formalisms with explicit concurrency turned out to be significant: the calculus *basic parallel processes* (*BPP*) of Christensen [Chr93] and traditional *Petri nets* (both considered under interleaving semantics).

The calculus BPP was conceived as a parallel counterpart to BPA: BPA can be seen as an extension of finite automata by a sequential operator; the class BPP includes a parallel combinator instead. More precisely, BPP are defined as a process algebra which comprises action prefix, choice, recursion, and parallel composition. They can also be viewed as a special class of Petri nets: in *communication-free Petri nets* tokens are allowed to flow freely through the net; if a token activates a transition it does so independently, without the help of other tokens [Hir94]. To be precise it is BPP in *standard normal form* (*SNF*) [Chr93] that exactly correspond to communication-free Petri nets [Esp97b]. But since every BPP can effectively be translated into a bisimilar BPP in SNF [Chr93] in the interleaving world it is indeed safe to identify BPP and communication-free nets. In the (branching-time) true-concurrency world BPP in SNF form a subclass of their own: following [EK95] we call this class *simple BPP* (*SBPP*). Under true-concurrency semantics BPP and SBPP induce a very special dynamic structure: the partial order computations of BPP are tree-like, more strongly the partial order unfoldings of SBPP are bipartite forests.

Positive results were soon obtained for BPP: bisimilarity was shown to be decidable first for normed BPP in [CHM93b], then for the entire class in [CHM93a]. Indeed, these results can also be shown for BPP_τ (BPP with τ), an extension of BPP that integrates synchronization on complementary actions [Chr93]. The two results are based on very different techniques. [CHM93a] relies on syntactic insights concerning commutativity that help to establish finiteness of a tableau system. In contrast, the proof for normed BPP is based on a decomposition result: with respect to bisimilarity normed BPP are *uniquely decomposable into* *prime components* (cf. Section 1.3.3). Based on this insight [HJM96] show that bisimilarity on normed BPP can be decided in polynomial-time.

While BPP can be seen as a minimal infinite-state model for concurrency, Petri nets reside at the upper end of expressiveness: "no natural model of concurrent computation lying strictly between Petri nets and Turing machines seems to have been proposed so far" [Esp97a]. It is folklore that Petri nets are very close to being Turing powerful (cf. [Pet81]): given a counter machine it is straightforward to construct a Petri net that simulates the machine in a weak sense. Counters are translated into unbounded places with the token load of a place corresponding to the value of the respective counter. The program of instructions is represented by a network of transitions; the transitions will output to or input from counter places according to whether the corresponding instruction increases or decreases the respective counter. The weakness of this translation lies in the fact that Petri nets cannot test for zero: a run of the net can nondeterministically choose a zero branch even if the value of the respective counter is not zero; there will always be one 'faithful' run which exactly corresponds to the counter machine execution, but additionally there may be many 'cheating' runs. The limits of the computational power of Petri nets are also demonstrated by the classical result that reachability, and thus halting, is decidable for Petri nets [May84].

Despite of this, Jančar came up with a reduction from the halting problem of counter machines which proves that bisimilarity as well as language (or trace) equivalence[6] is undecidable for Petri nets [Jan95]: given a counter machine C one constructs two variations of the Petri net that weakly simulates C such that the difference beween these two nets can only be exposed by faithfully simulating C and reaching the halting state (in one of the nets); the two nets are non-equivalent iff C halts. Building on Jančar's technique Hirshfeld managed to resolve that trace equivalence is undecidable for communication-free Petri nets, and hence BPP [Hir94].

For both BPA and BPP it was carried over that all the intermediate equivalences of the linear-time – branching-time spectrum are undecidable [GH94, HT95, Hüt94]. Thus, on BPA and BPP bisimilarity is the only decidable equivalence of the classical spectrum. Many more results, less relevant for this thesis, have been achieved for classes of infinite-state transition systems. For example, model-checking problems have also been studied intensively. The handbook chapter [BCMS01] provides a comprehensive survey on the key decidability and

[6]The undecidability of language equivalence was first proved by Hack, but [Jan95] provides a stronger proof; cf. [Jan95, EN94].

complexity results that have been achieved for the standard infinite structures. It subsumes the surveys [Mol96] and [HM96], which concentrate on the area of equivalence checking; these have partly inspired our presentation. In addition, all these surveys present unified views of the infinite-state formalisms they investigate: it is common to present infinite-state processes as a hierarchy of special classes of term rewrite systems.

Infinite-State Verification under True-Concurrency Semantics. In the above line of research, classes with explicit concurrency, such as Petri nets and BPP, are employed under interleaving semantics, in their character as generators of infinite-state transition systems. It is obvious that such formalisms can be investigated under true-concurrency semantics just as well. Then, they provide generators of infinite-state tsi's or net systems, allowing us to explore how truly-concurrent equivalences (or logics) behave in the infinite-state world.

As noted in [Jan95] and [Esp98], for Petri nets the undecidability of trace equivalence and classical bisimilarity (and indeed of all the intermediate equivalences) can directly be carried over: Jančar's technique only employs sequential systems, for which partial order and interleaving concepts naturally coincide. Then, undecidability for Petri nets also applies to notions such as step bisimilarity, pomset trace equivalence, pomset bisimilarity, hp bisimilarity, and hhp bisimilarity.

However, for BPP many positive results have been obtained. One of the earliest such results is Christensen's proof of the decidability of distributed bisimilarity for BPP and BPP_τ [Chr92, Chr93]. This result actually precedes the works on classical bisimilarity [CHM93b] and [CHM93a]. Its tableau-based proof relies on a decomposition property that is directly induced by the nature of distributed bisimilarity. Alternatively, decidability can be proved in the style of [CHM93b]: w.r.t. distributed bisimilarity unique decomposition into prime components is given for the full BPP (or BPP_τ) class [Chr93]. Furthermore, in [KH94] Kiehn and Hennessy present decision procedures for strong and weak versions of causal bisimulation, location equivalence, and ST-bisimulation, also for the system class of BPP_τ. This work builds on the proof for classical bisimilarity [CHM93a], and hence relies on syntactic insights about commutativity.

So far, the results are as one could expect, and in accordance with the results of the interleaving world. However, a special trend for true-concurrency in the infinite-state world emerges from the works [SN96] and [EK95]. In [SN96] Sunesen and Nielsen study causality- and locality-based linear-time equivalences for infinite-state classes. Contrasting Hirshfeld's undecidability result for trace equivalence they prove that pomset trace equivalence and location trace equivalence are decidable for BPP and BPP_M, where BPP_M corresponds to BPP_τ. With [EK95] a similar trend had already been revealed in model-checking: Esparza and Kiehn show that a logic equivalent to CTL^* is decidable for BPP under partial order interpretation, whereas under interleaving semantics a small fragment of this logic is already undecidable for *very basic* BPP. Both of these works exploit the special tree-like structure of (S)BPP: the decidability results of [SN96] follow by a reduction to the equivalence problem of recognizable tree languages; [EK95] employs a reduction to the validity problem for the monadic second order logic of a tree with fan-out degree n (SnS). The above trend is also confirmed by recent complexity results. In [Las03] Lasota proves that distributed bisimilarity (and thus several other notions of bisimilarity, see below) is polynomial-time decidable for BPP. In contrast, [Jan03] establishes that classical bisimilarity on BPP is PSPACE-complete.

Altogether this shows that moving to standard classes of infinite-state systems does not necessarily increase the difficulty in true-concurrency. On the contrary, things may become easier. In the finite-state world truly-concurrent paradigms are typically harder than their interleaving counterparts; in the infinite-state world this effect may be reversed: the standard infinite-state classes may have natural decomposition properties and good structural features; such characteristics are particularly exploitable in the true-concurrency world, where they may very directly translate into decision procedures. In the end, it is the interplay between causality, concurrency, and conflict that matters in true-concurrency.

The special structure of BPP processes also seems to be responsible for many coincidence results: several important non-interleaving equivalences coincide for BPP-like process languages. In [Ace92b] Aceto shows that distributed, timed, and causal bisimilarity coincide for a language that is essentially BPP without recursion. Furthemore, Kiehn has recently extended these results by proving that location equivalence, causal bisimulations, and distributed bisimulations coincide over CPP, a language that corresponds to BPP_τ without explicit τ actions [Kie99]. In the linear-time world, Sunesen and Nielsen find that location and pomset trace equivalence coincide for BPP and BPP_M [SN96].

Relevancy. Analogously to Section 1.3.1 we now summarize in which way the presented results and themes of the area infinite-state verification are relevant to an investigation of hp and hhp bisimilarity. The last point will not directly manifest itself in this thesis but it is important in view of the undecidability result, which we will discuss in Section 7.2.

1. General Connection. In general, infinite-state verification is very relevant for us: even if we are concerned with finite-state systems, these act as generators of infinite spaces on which the hhp bisimilarity problems are ultimately defined. As in infinite-state verification the question is whether infinite spaces can be tamed via insights into regularity or periodicity, or with the help of structural insights. Thus, the methodology and ideas of infinite-state verification may be very relevant for an investigation of problems in true-concurrency. Conversely, we can hope: just as the finer view of interleaving process theory has proved to be fruitful for formal language theory, true-concurrency may be relevant for refining our understanding of the coarser interleaving view. This applies to the finite-state as well as the infinite-state world.

2. Infinite-State Investigations. We have seen that it is fruitful to investigate truly-concurrent problems on standard infinite-state classes equipped with partial order semantics. We are interested in examining whether the positive trend that emerged from the works [EK95] and [SN96] can be confirmed with respect to hp and hhp bisimilarity. In particular:

3. SBPP and BPP provide the natural classes to start with when exploring truly-concurrent concepts in the infinite-state world. Under partial order semantics they exhibit tree-like behaviour, and thereby restrict the interplay of causality, concurrency, and conflict in an interesting way. Together, this pinpoints SBPP and BPP as ideal classes to consider in our analysis of hp and hhp bisimilarity.

4. The Tableau Technique provides a formal proof method in which characteristic insights such as decomposition theorems can be employed to establish decidability. In particular, this technique allows us to tackle state spaces that are inherently infinite in that they do not admit a quotienting. By incorporating a way of looping back to earlier points in the proof tree, a finite tableau can represent an infinite bisimulation.

5. Composition and Decomposition. As we saw, many of the results in the infinite-state world are based on an exploitation of decomposition theorems. The idea of composition and decomposition will be crucial for us in many ways; it will provide our main technique to prove decidability and coincidence results in the finite-state as well as the infinite-state world. We will present this theme in more detail in Section 1.3.3.

6. An Undecidability Proof. Jančar's technique [Jan95] provides a scheme for proving a behavioural equivalence undecidable. It gives a starting point when investigating whether hhp bisimilarity is undecidable: can we show that (finite-state) concurrent systems can simulate Turing machines in a sense that is relevant for the hhp bisimilarity problem? If yes, can we use this find to reduce the halting problem to checking hhp bisimilarity? With the undecidability result [JN00] these questions have been answered affirmatively (cf. Section 7.2).

1.3.3 Composition and Decomposition

As we saw in Section 1.3.2 the idea of composition and decomposition is one of the crucial techniques to establish decidability and complexity results in the area of infinite-state verification. For example, the polynomial-time decision procedure for bisimilarity on normed BPP [HJM96] is based on the following insight: any normed BPP can be expressed uniquely, up to bisimilarity, as a parallel composition of prime factors [CHM93b]. A process is *prime* if it is not the nil process and it is irreducible with respect to parallel composition, up to bisimilarity. Such a decomposition theory translates into cancellation properties of the form "$P \,||\, Q \sim R \,||\, Q$ implies $P \sim R$", which provide the means to reduce pairs of processes to compare into smaller pairs of processes to check. Questions about prime decomposability were first addressed by Milner and Moller in [MM93]. In particular, they show that unique decomposition with respect to bisimilarity is given for finite processes, but they disprove decomposition (into a *finite* set of prime factors) for arbitrary finite-state processes.

In the interleaving world, concurrency is not present at the semantic level. Consequently, the concepts of prime component and prime decomposability are considered with respect to syntax, i.e. with respect to the parallel operator '$||$', and they are directly coupled to a notion of equivalence: can a process *syntactically* be expressed as an equivalent parallel composition of prime processes, and, if yes, is such a presentation given uniquely up to the equivalence? In contrast, in the true-concurrency world, composition and decomposition can be considered at the semantic level: we can directly recognize whether a truly-concurrent system can be dissected into independent factors. It can then separately be investigated whether a specific decomposition view translates into a given equivalence.

In Section 4.3 we will introduce a very concrete notion of *decomposition into independent factors*, and it will follow easily that every lats uniquely decomposes into a set of prime factors, where prime factors are semantic entities, defined as particular subsystems of the respective lats. We then show that for *bsc-decomposable systems* hp and hhp bisimilarity are decomposable with respect to the set of prime components: whenever two bsc-decomposable systems are hp (hhp) bisimilar then there is a one-to-one correspondence between their prime components such that related components are hp (hhp) bisimilar. Thus, although decomposition has a more concrete flavour in true-concurrency the end is the same: it gives us the means to check whether two processes are equivalent by checking for equivalence among their smaller prime processes.

Naturally, to make such an approach sound we need complementary composition results. It will be straightforward to show that hp and hhp bisimilarity are composable in the following sense: assume two systems each decomposed into a set of independent factors; whenever we can exhibit a one-to-one correspondence between the components of the two systems such that related components are hp (hhp) bisimilar then the two systems are hp (hhp) bisimilar. This again is related to syntactic composition in the process algebra world, where the following congruence result is often exploited: if $P \sim Q$ and $P' \sim Q'$ then $P \,||\, Q \sim P' \,||\, Q'$. An observation that seems to be valid for both the interleaving and the true-concurrency world is that composition results are generally easy to obtain, but decomposition questions can be hard and highly intriguing [MM93].

Composition and decomposition are inherently connected to the shuffling of independent transitions: by the axioms of independence the global behaviour of a system is exactly the shuffle product of the behaviour of its independent factors. Thus, composition and decomposition theorems provide an important tool to establish coincidence results: after all proving coincidence between hp and hhp bisimilarity amounts to proving that whenever two systems are hp bisimilar there exists a hp bisimulation that satifies specific shuffle properties, the hereditary condition. To make this more precise, assume two bsc-decomposable systems S_1 and S_2 that are hp bisimilar. By decomposition for hp bisimilarity we obtain a bijection between the prime components of S_1 and those of S_2 such that related components are hp bisimilar. Then, provided that hp and hhp bisimilarity coincide for the class of the prime factors, by composition for hhp bisimilarity we can conclude that S_1 and S_2 are hhp bisimilar.

In this spirit we formulate and exploit various composition and decomposition views throughout the thesis. Our results on composition and decomposition into independent (prime) factors provide the key to several coincidence results. We will employ them in an inductive argument (Section 4.4), and furthermore in a tableau-based proof (Section 5.3), where they establish the soundness and completeness of a tableau system. In our exploration of BPP (Section 5.4) we formulate specific decomposition views; we show that these translate into hp and hhp bisimilarity in a natural way, which again leads to tableau-based proofs. In our study of live free choice systems (Chapter 6) we develop a more sophisticated decomposition theory. The idea is to generalize 'decomposition into independent prime factors' by integrating a concept of synchronization. Naturally, this will make things much more difficult, and further tools will be developed, which help to approach such a decomposition theory. One idea, which is related to the concept of decomposition, is to design and employ auxiliary equivalences which reflect notions of compositionality and locality. Such notions of bisimilarity can be understood to implement the 'interleaving aspect' of a decomposition property. Here, our theme 'composition and decomposition' meets our first theme 'true-concurrency versus causal (or local) interleaving'.

1.4 Approach and Preview

We now explain how we will proceed and what we shall achieve in the remainder of this thesis.

In *Chapter 2* we gather together the necessary background material. First of all, we define the models of concurrency which we shall use and set up the associated technical machinery. Then we give the formal definitions of hp, hhp, and chhp bisimilarity. We also present the ideas behind the decidability proofs of hp bisimilarity more formally since we will built on them later on.

We are then ready to present the contributions of this thesis. In general, our approach is to approximate the decidability of hhp bisimilarity and its coincidence with hp bisimilarity from below: we identify restricted problems for which we hope to obtain positive solutions. There are two ways in which one can restrict the decidability or coincidence problem of a behavioural equivalence: by coarsening or strengthening the equivalence in a way that will make the problem more accessible, or by imposing constraints on the behaviour of the systems under study. (Unlike the first, the second option will give rise to proper subproblems.)

hhp bisimilarity is so powerful due to its backtracking capability: as we saw in Section 1.2.1 backtracking can be employed to expose subtle aspects of the interplay between causality, concurrency, and conflict. In *Chapter 3* we aim to disentangle the power of hhp bisimilarity by constraining its backtracking capability. We will see that backtracking has two dimensions: the number of transitions over which one may backtrack, and the number of backtracking moves. These dimensions translate into two hierarchies of restricted backtracking bisimilarities. We find that both of them are strict, and that each of their levels is decidable for finite-state systems. We discuss how the hierarchy insights apply to the decidability problem of hhp bisimilarity. In particular we obtain decidability for two subclasses: finite-state *bounded asynchronous systems* and finite-state *systems* *with transitive independence relation.*

In the second and major part of our analysis we study the coincidence and decidability problem of hhp bisimilarity on system classes which have a restricted interplay of the three basic situations. Our interest is to analyse which behavioural properties of concurrent systems are crucial to the increased power of hhp bisimilarity. *Chapter 4* is our starting point. Here we identify important behavioural situations and deliver first insights on the coincidence problem. In Section 4.2 we start out by investigating the three basic situations separately. We find that concurrency and conflict are both cruical to keep hp and hhp bisimilarity distinct, but that this is not the case for causality. Furthermore we identify *(L&C)-nondeterminism* as a crucial situation. Investigating it will lead us to an interesting counter-example. In Section 4.3 we prove our first composition and decomposition results: hp and hhp bisimilarity are composable with respect to *decompositions of systems into independent components*, and for *bsc-decomposable systems* the two bisimilarities are decomposable with respect to the *set of prime components*. It will follow that hp and hhp bisimilarity coincide for *parallel compositions of sequential systems*, which confirms that the increased power of hhp bisimilarity relies on the intertwining of concurrency with conflict (and causality). Motivated by this in Section 4.4 we study three *synchronization witness* (short: *SW*) *situations*. With the help of our composition and decomposition result we show that in their entirety they are essential to distinguish between hp and hhp bisimilarity for *bounded-degree systems*. As a corollary we obtain coincidence for bounded-degree *communication-free net systems*. In Section 4.5 we study a further aspect of the interplay of causality, concurrency, and conflict: the situation of *confusion*. We show that hp and hhp bisimilarity do *not* coincide for confusion-free systems, or, more strictly, free choice systems, thereby disproving the conjecture of Cheng. Our counter-example will lead us to identifying a new kind of confusion: so-called *syn-confusion*. Finally, in view of Chapter 6 we introduce the concept of *liveness*.

Our analysis will be continued in Chapters 5 and 6 with the help of two structural classes. In *Chapter 5* we study the process algebras *SBPP* and *BPP*, with a suitable partial order semantics. Although we are primarily concerned with finite-state systems, our study of SBPP and BPP contributes to the area of infinite-state verification. Our motivation is twofold. Due to their behavioural properties SBPP and BPP exactly fit into our analysis of the interplay between causality, concurrency, and conflict: they have restricted synchronization and thus their study links up to our investigation of the SW situations. At the same time, we are interested in examining whether we can confirm the positive trend of true-concurrency in the infinite-state world, which emerged from the works [EK95] and [SN96]. We will achieve several decidability and coincidence results which indeed confirm this trend. For SBPP we prove decidability and coincidence of hp and hhp bisimilarity. Since SBPP are interpreted as a class of communication-free net systems this result is related (but orthogonal) to the coincidence for bounded-degree communication-free systems. For BPP the two bisimilarities do not coincide; this follows from the second standard counter-example. But we separately achieve decidability for both. The proofs also lead us to two coincidence results: for BPP, hp bisimilarity coincides with distributed bisimilarity and hhp bisimilarity with chhp bisimilarity. The results for hp bisimilarity are also known via insights on causal bisimilarity, which coincides with hp bisimilarity. The results on hhp bisimilarity are all new. The decidability of hhp bisimilarity on BPP is particularly interesting: it shows that in the true-concurrency world an equivalence that is undecidable for finite-state systems can be decidable for a standard class of infinite-state systems. The proofs behind our results follow a common scheme: BPP and SBPP have a tree-like structure, and consequently they enjoy good composition and decomposition properties. These translate into hp, hhp, and chhp bisimilarity in a natural way, which allows us to construct clear tableau proof systems. Our work also pinpoints that BPP can be interpreted as a type of *proper-communication free net systems.*

In *Chapter 6* we study our second group of structural system classes: sub-classes of finite-state *free choice* (*fc*) *net systems.* Having shown that hp and hhp bisimilarity do not coincide for the full class, we concentrate on *live* fc systems. Apart from being motivated by the free choice hiatus (cf. Section 1.3.1) they make a particularly good candidate due to their behavioural properties: apart from being confusion-free they appear to exclude syn-confusion. Yet, live fc systems provide a demanding class to tackle: since they admit both conflict and synchronization their unfolding structure may amount to a complicated intertwining of causality, concurrency, and conflict.

By Hack's S-coverability Theorem live fc systems can be understood as a synchronization of a set of state machine components. We use this decomposition characteristic as the basis of our work. Indeed we adopt a simplification: for the present we restrict our attention to live *strictly* state machine decomposable (*SSMD*) fc systems so as to obtain slightly better decomposition properties. Our initial idea is to tackle the coincidence problem of live (SSMD) fc systems by developing a decomposition theory for hp and hhp bisimilarity: to generalize ‘decomposition into independent prime factors’ by integrating a concept of synchronization. Such a decomposition theory is of course considerably more difficult to analyse and prove (or even formulate); however, guided by this idea we have come up with an approach that disentangles the difficulty of the problem by breaking it down into several accessible subgoals.

As one component of our approach we design and employ auxiliary equivalences which reflect notions of compositionality and locality in net systems. *Compositionality preserving* (*cp*) *bisimilarity* reflects the distribution of states into places and the locality of transition firing. *Block preserving bisimilarity* more generally reflects the distribution of states into ‘active blocks’, and is particularly interesting in the context of fc systems. These equivalences are closer to the decomposition properties we would like to obtain. Also, cp bisimilarity allows for a very direct exploitation of the topological information provided by a strict state machine decomposition.

Another guiding thought is the idea that in true-concurrency structural constraints in the systems may very directly lead to characteristics in the matching of bisimulations. One of our key theorems exposes a constraint in the topology of live SSMD fc systems: roughly speaking it says that a certain type of path cannot exist. Building on this insight we then show that a particular aspect of the matching in cp bisimulations is deterministic, and thus predictable.

In general, if the matching is deterministic in a certain way it could well be the case that we have so few options of how to build a bisimulation that we are forced to match in a hereditary way, or that we can easily transform any bisimulation into a hereditary one by shuffling the matching. In this spirit, we obtain that the shuffle product of certain extracts of cp bisimulations are contained in the largest cp bisimulation. Thereby, we will establish a specific decomposition property, and at the same time it will follow that cp bisimilarity satisfies a restricted coherent and hereditary property.

Altogether, we achieve a considerable part of the subgoals that together imply that hp and hhp bisimilarity coincide for live SSMD fc systems. We show that the remaining gaps can easily be overcome by imposing slight restrictions on our system class; all of these restrictions act locally on the post-set of transitions. First, we deduce that cp, hereditary cp (hcp), and coherent hcp (chcp) bisimilarity coincide for live *sy-psd* SSMD fc systems. Then, we gain decidability of chhp bisimilarity for live sy-psd *buffered* SSMD fc systems, and coincidence between hp, hhp, and chhp bisimilarity for live *spsd* buffered SSMD fc systems. To my knowledge, these are the only positive results for classes that allow a reasonable amount of interplay between causality, concurrency, and conflict while still admitting considerable nondeterminism.

In *Chapter 7* we summarize and discuss our results and draw conclusions. We also review the undecidability of hhp bisimilarity. Finally, we consider general directions for further research. In particular we speculate what might be gained with respect to automatic verification, connecting back to our motivation of Section 1.1.

Chapter 2

Background

In this chapter we gather together the necessary background material. In Section 2.1 we define the models of concurrency we shall use, and set up the associated technical machinery. In Section 2.2 we provide the formal definitions of hp, hhp, and chhp bisimilarity. Afterwards, in Section 2.3, we present the ideas behind the decidability proofs of hp bisimilarity. In Section 2.4 we explain some conventions we shall adopt, and in Section 2.5 we introduce further concepts we want to employ later on.

2.1 Models for Concurrency

For us a model for concurrency must be able to express the three fundamental situations in which two events can be related: causality, concurrency, and conflict [RE98, RT86]. Furthermore, it will be useful for us to work with models of three different levels of abstraction:

1. a *structural level*, which reveals the structure of the systems in greatest detail: concepts of locality and distribution are maintained, and hence structural interaction such as synchronization is visible;

2. a *first behavioural level*, which abstracts away from localities, but keeps the cyclic structure of the systems; at this level, concurrency will be captured as independence between transitions;

3. a *second behavioural level*, which in addition to the abstractions of the second level unfolds the systems.

We employ *1-safe Petri nets* as our structural model, *lats'* as the model at the first behavioural level, and *occurrence lats'* as the model of the third level. For the remainder of the thesis we fix a set $Act := \{a, b, c, \ldots\}$ of actions.

2.1.1 Net Systems

As explained in Section 1.3.1 *Petri nets* capture true-concurrency by making states and transitions distributed entities. One first defines the underlying structure of a Petri net:

Definition 2.1.1. A *(labelled) net* N is a tuple (P_N, T_N, F_N, l_N), where

- P_N is the set of *places*,
- T_N is the set of *transitions*,
- $F_N : (P_N \times T_N) \cup (T_N \times P_N) \to \{0, 1\}$ is the *flow relation*, and
- $l_N : T_N \to Act$ is the labelling function.

The *pre-set* of an element $x \in P_N \cup T_N$, ${}^\bullet x$, is defined by $\{y \mid F_N(y, x) > 0\}$, the *post-set* of x, $x^\bullet$, similarly is $\{y \mid F_N(x, y) > 0\}$.

For technical convenience we employ a commonly used restriction:

Restriction 2.1.1. We only consider nets N that satisfy the following property: $\forall t \in T_N.\ {}^\bullet t \neq \emptyset$.

A net becomes dynamic when it is equipped with a marking:

Definition 2.1.2. Let N be a net.

A *marking* M of N is a map $P_N \to \mathbb{N}_0$.

M *enables* a transition $t \in T_N$ if $M(s) \geq F_N(s, t)$ for every $s \in P_N$. If t is enabled at M then it can occur or *fire*. The resulting marking M' is defined by $M'(s) = M(s) - F(s, t) + F(t, s)$ for all $s \in P_N$. Altogether we write $M[t\rangle M'$ or $M \xrightarrow{t} M'$. Extending this notation to sequences of transitions, we write $M[w\rangle M'$ or $M \xrightarrow{w} M'$, where $w = t_1 t_2 \dots t_n \in T_N^*$, to denote $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} M_n$ for some markings $M_1, \dots, M_n$ with $M_n = M'$. Furthermore, we write $M[w\rangle$ or $M \xrightarrow{w}$ to say that $M \xrightarrow{w} M'$ for some M'.

A marking M' is *reachable from* M when $M \xrightarrow{w} M'$ for some sequence w. We denote the set of markings of N reachable from M by $Reach(N, M)$, or simply by $Reach(M)$ if the net is clear from the context.

Petri nets are then defined as follows:

Definition 2.1.3. A *Petri net* $\mathcal{N}$ is a pair (N, M_0), where N is a net, and M_0 is the *initial marking* of $\mathcal{N}$.

We say a marking M is *reachable in* $\mathcal{N}$ iff it is reachable from M_0. We denote the set of reachable markings of $\mathcal{N}$ by $\mathit{Reach}(\mathcal{N})$.

$\mathcal{N}$ is *1-safe* (short: *safe*) iff for every $M \in \mathit{Reach}(\mathcal{N})$ we have: $\forall p \in P_N.\ M(p) \leq 1$. Thus, in safe Petri nets a marking can be viewed as a set of places. We say $p \in P_N$ *holds at marking* M iff $p \in M$. A marking M of a net N is safe iff (N, M) is a safe Petri net.

Convention 2.1.1. Unless stated otherwise, we work with *1-safe* Petri nets; we will refer to them as *net systems*.

2.1.2 Labelled Asynchronous Transition Systems

A more abstract model for concurrency is provided by *labelled asynchronous transition systems* (*lats'*); they capture concurrency explicitly as independence between transitions. Formally, lats' are defined as transition systems with extra structure: an additional level of labelling identifies which transitions are to be thought of as occurrences of the same event, or Petri net transition; a relation of independence describes which events are independent of each other. Several axioms ensure that intuitive properties of events and independence are respected. We follow the definitions of [WN97].

Definition 2.1.4. A *(labelled) transition system* (*lts*) is a tuple $\mathcal{T} = (S, L, \mathit{Tran})$, where S is a set of *states*, L is a set of *labels*, and $\mathit{Tran} \subseteq S \times L \times S$ is the *transition relation*.

A *lts with initial state* (*ltsis*) is a tuple $\mathcal{T} = (S, s^i, L, \mathit{Tran})$, where (S, L, Tran) is a lts, and $s^i \in S$ is the *initial state*.

We write $s \xrightarrow{t} s'$ to indicate that $(s, t, s') \in \mathit{Tran}$. Similarly to before, we lift the arc-notation to strings of labels, and we write $s \xrightarrow{w}$ or $s[w\rangle$ to denote that $s \xrightarrow{w} s'$ for some s'. We define the states *reachable* from state s in $\mathcal{T}$ by $\mathit{Reach}(\mathcal{T}, s) := \{s' \mid \exists w.\ s \xrightarrow{w} s'\}$; we also write $\mathit{Reach}(s)$ if the context is clear. If $\mathcal{T}$ is a ltsis we write $\mathit{Reach}(\mathcal{T})$ for $\mathit{Reach}(\mathcal{T}, s^i)$ to denote the states reachable in $\mathcal{T}$.

Definition 2.1.5. A *(labelled) asynchronous transition system* (*lats*) is a structure $S = (S_S, s^i_S, T_S, I_S, \mathit{Tran}_S, l_S)$, where

- $(S_S, s^i_S, T_S, \mathit{Tran}_S)$ is a ltsis,
- T_S is the set of *events* or *transitions*[1],

[1] Usually, T (or rather E) is called a set of *events*; we prefer the term *transitions* here since

- $l_S : T_S \to Act$ is the *labelling function*, and
- $I_S \subseteq T_S \times T_S$, the *independence relation*, is an irreflexive, symmetric relation on the set T_S

such that

1. $t \in T_S \implies \exists s, s' \in S_S.\ (s, t, s') \in Tran_S$,
2. $(s, t, s') \in Tran_S\ \&\ (s, t, s'') \in Tran_S \implies s' = s''$,
3. $t_1\ I_S\ t_2\ \&\ (s, t_1, s_1) \in Tran_S\ \&\ (s_1, t_2, u) \in Tran_S$
$$\implies \exists s_2.\ (s, t_2, s_2) \in Tran_S\ \&\ (s_2, t_1, u) \in Tran_S.$$

We say an lats is *coherent* if it additionally satisfies

4. $t_1\ I_S\ t_2\ \&\ (s, t_1, s_1) \in Tran_S\ \&\ (s, t_2, s_2) \in Tran_S$
$$\implies \exists u.\ (s_1, t_2, u) \in Tran_S\ \&\ (s_2, t_1, u) \in Tran_S.$$

We carry over our notations for ltsis' of Def. 2.1.4 to lats' in the obvious way.

Axiom (1) says that every event appears as a transition, and axiom (2) that the occurrence of an event at a state leads to a unique state. Axioms (3) and (4) express two natural properties of independence: (3) asserts that if two independent transitions can occur consecutively then they can also occur in opposite order. (4) expresses that if two independent transitions can occur alternatively at a common state then they can also occur consecutively from that state. The latter two axioms induce the typical "independence squares". We refer to axiom (3) as the *first axiom of independence*, and to axiom (4) as the *second axiom of independence*.

We also employ the following definitions and conventions:

Definition 2.1.6. Let S be a lats.

We call the complement of I_S the *dependence relation* of S, and denote it by D_S. We lift independence and dependence to sequences and sets of transitions, e.g. we write $t_1 \dots t_n\ I_S\ t'_1 \dots t'_m$ iff $t_i\ I_S\ t'_j$ for all $i \in [1, n]$, $j \in [1, m]$.

We say S is *without redundant states* iff $s \in S_S \implies \exists w \in T_S^*.\ s_S^i \xrightarrow{w} s$ (in other words, iff $S_S = Reach(S)$).

---

in our context events can be interpreted as Petri net transitions (boxes). Note, however, that this leads to an unfortunate clash of terminology with transitions in the sense of '$s_1 \xrightarrow{t} s_2$'. Apart from a few exceptions (such as in the paragraph following this definition) the ambiguity will be resolved by the context, and we reserve the term 'event' for events in the sense of event structures, that is for 'unfolded' models.

S is *empty* iff $T_S = \emptyset$, and *non-empty* otherwise. (Note that if S is without redundant states then S is non-empty iff $s_S^i \xrightarrow{t}$ for some t, and empty iff $S = (\{s^i\}, s^i, \emptyset, \emptyset, \emptyset, \emptyset)$ for some s^i.)

Convention 2.1.2. In the following, we assume lats' to be *coherent* and *without redundant states.* Since lats' are our primary semantic model we usually refer to them simply as *systems.* This is also to indicate that a more structural model, such as a net system, may stand behind the respective lats.

The three fundamental situations, *concurrency*, *causality*, and *conflict*, are naturally captured as follows:

Definition 2.1.7. Let S be a system, and $s \in S_S$.

1. t_1 *can occur causally dependent on* t_2 *at* s, denoted by $t_2 <_s t_1$, iff $s[t_2\rangle$, $t_1\ D_S\ t_2\ \&\ s'[t_1\rangle$, where s' is such that $s \xrightarrow{t_2} s'$.

   We say t_c *can occur causally dependent on* t_1 *and* t_2 *at* s, denoted by $\{t_1, t_2\} <_s t_c$ iff $s[t_1t_2\rangle$ (or $s[t_2t_1\rangle$), $t_c\ D_S\ t_1$, $t_c\ D_S\ t_2\ \&\ s'[t_c\rangle$, where s' is such that $s \xrightarrow{t_1t_2} s'$ (or $s \xrightarrow{t_2t_1} s'$).

2. t_1 and t_2 *can occur concurrently at* s, denoted by $t_1\ co_s\ t_2$, iff $s[t_1\rangle$, $s[t_2\rangle$ & $t_1\ I_S\ t_2$.

3. t_1 and t_2 *are in conflict at* s, denoted by $t_1\ \#_s\ t_2$, iff $s[t_1\rangle$, $s[t_2\rangle$, and $t_1\ D_S\ t_2$.

We could define analogous characterizations for net systems: we can detect how two transitions are related with respect to causality, concurrency, and conflict at a given state, by considering whether and how their environments of places intersect. On the other hand, we can associate an independence relation with net systems, and translate them into lats'. This will make concrete that lats' can be seen as an abstraction of net systems, and concepts defined for lats' will carry over to net systems in the obvious way.

Definition 2.1.8. Let N be a net. We say two transitions t, $t' \in T_N$ are *independent* in N, denoted by $t\ I_N\ t'$, iff their neighbourhoods of places do not intersect, i.e. iff $({}^\bullet t\ \cup\ t^\bullet)\ \cap\ ({}^\bullet t'\ \cup\ t'^\bullet) = \emptyset$. We lift independence to sequences and sets of transitions as we did for lats'.

Proposition 2.1.1. *Let $\mathcal{N}$ be a net system.* $(Reach(\mathcal{N}), M_0, T'_N, I_N \cap (T'_N \times T'_N), Tran_N, l_N\lceil_{T'_N})$ *is a lats, where* $T'_N = \{t \in T_N \mid \exists M \in Reach(\mathcal{N}).\ M \xrightarrow{t}\}$*, and* $Tran_N$ *is the transition relation induced by the Petri net firing rule.*

Proof. This is straightforward to check. Note that axioms (3) and (4) of the definition of lats' easily follow by the definition of I_N and the transition firing rule. $\square$

Definition 2.1.9. Let $\mathcal{N}$ be a net system. We denote the lats associated with $\mathcal{N}$ in Prop. 2.1.1 by $lats(\mathcal{N})$.

The following is a consequence of the first axiom of independence:

Proposition 2.1.2. *Let $\mathcal{N}$ be a net system, $M \in Reach(\mathcal{N})$, and $w, w' \in T_N^*$ such that $w\ I_N\ w'$. If $M \xrightarrow{w} M' \xrightarrow{w'} M''$ for some M', M'', then there is M''' with $M \xrightarrow{w'} M''' \xrightarrow{w} M''$.*

Proof. Straightforward. $\square$

In studying hp and hhp bisimilarity, we will be concerned with the partial order behaviour of the systems under study. The *runs* of a system are defined as follows:

Definition 2.1.10. Let S be a system.

We say $r = t_1 \dots t_n \in T_S^*$, is a *transition-sequence* of S. We write $|r|$ for the length of r, that is $|r| = n$. For any $i \in [1, |r|]$ we denote the ith transition of r, t_i, by $r[i]$ or $tr(r, i)$; alternatively, if the context of r is clear we also write t_i or t^i (even if r is only given as $r \in T_S^*$). For any $t \in T_S$ we write $t \in r$ if $t = t_i$ for some $i \in [1, n]$. If $t \in r$ we let $last(r, t)$ denote the position of the last occurrence of t in r. That is $last(r, t) = i$ iff $t_i = t$ and $t_j \neq t$ for all $j \in [i + 1, n]$. Given $r' = t'_1 \dots t'_m \in T_S^*$, we write $r.r'$ for the concatenated sequence $t_1 \dots t_n t'_1 \dots t'_m$.

Let $s \in S_S$. A *run of S starting at s* is a possibly empty transition-sequence r such that $s \xrightarrow{r} s'$ for some s'. We let $Runs(S, s)$ denote the set of all runs of S starting at s. We also write $Runs_S(s)$, or simply $Runs(s)$ if the context of S is clear.

A *run of S* is a run of S starting at s_S^i. We let $Runs(S)$ denote the set of all runs of S. For r, $r' \in Runs(S)$, $w \in T_S^*$ we write $r \xrightarrow{w} r'$ iff $r' = r.w$.

Following [JM96] we associate a *pomset* with each run of a system:

Definition 2.1.11. A *pomset* is defined as a labelled partial order.[2] It is a tuple $p = (E_p, <_p, L_p, l_p)$, where E_p is a set of *events*, $<_p$ a partial order relation on E_p, L_p is a set of labels, and l_p a labelling function $l_p : E_p \to L_p$.

[2]We do not use the standard definition, but the convention used in [JM96].

A function f is an *isomorphism* between pomset p and pomset q iff $f : E_p \to E_q$ is a bijection, such that we have $l_p = l_q \circ f$, and $e <_p e'$ iff $f(e) <_q f(e')$ for all $e, e' \in E_p$.

Definition 2.1.12. Let S be a system, $s \in S_S$, and $r = t_1 \dots t_n \in Runs(S, s)$.

The *transition-pomset* of r, denoted by $trPom(r)$, has as events the integers from 1 to n, where the label of event i is t_i, and the partial ordering is the transitive closure of the following "proximate cause" relation: event i *proximately causes* event j, written $i <_r^{prox} j$, iff $i < j$ and t_i and t_j are *not* independent in S. We denote this partial ordering on [1,n] by '$<_r$'. For $i \in [1, n]$ we write $k\ co_r\ l$ short for $(k \not\leq_r l)$ & $(l \not\leq_r k)$, and $k\ dep_r\ l$ short for $(k \leq_r l) \vee (l \leq_r k)$.

The *pomset* of r, denoted by $pom(r)$, is the transition-pomset of r, where the label of each event i is $l_S(t_i)$, the *label* of t_i, rather than t_i itself.

We carry these concepts over to net systems in the obvious way. Given a net system $\mathcal{N}$ we write $Runs(\mathcal{N})$ for the runs of $\mathcal{N}$. Given a net N and a safe marking M of N, we write $Runs(N, M)$ or $Runs_N(M)$ for the runs of N starting at M; if N is clear from the context we also use $Runs(M)$.

2.1.3 Unfoldings

By abstracting away from their cyclic structure, lats' can be unfolded into *occurrence lats'*. We denote the *unfolding* of a system S by $unf(S)$.

Definition 2.1.13. Given a lats S, for r, $r' \in Runs(S)$ we define $r \cong r'$ iff $trPom(r)$ and $trPom(r')$ are isomorphic; this is clearly an equivalence relation.

An *occurrence lats* is a lats U, where

- if $s \xrightarrow{v} s$, for some $v \in T_U^*$, then v is empty (i.e. the lats is acyclic), and
- $s_U^i \xrightarrow{r} s$ & $s_U^i \xrightarrow{r'} s \implies r \cong r'$.

The *unfolding of a lats S* is the occurrence lats $unf(S) = (S_U, s_U^i, T_S, I_S, Tran_U, l_S)$, where

- S_U is defined to be $Runs(S)/\cong$,
- $s_U^i = [\varepsilon]_\cong$, and
- $(\tau, t, \tau') \in Tran_U$ iff $r \in \tau$ and $r.t \in \tau'$ for some run $r \in Runs(S)$.

Naturally, a system and its unfolding will have exactly the same behavioural properties. Hence, we also have:

Fact 2.1.1. *Let S_1 and S_2 be two systems. For any notion of behavioural equivalence, say $\sim_x$, we have: $S_1 \sim_x S_2 \iff \mathit{unf}(S_1) \sim_x \mathit{unf}(S_2)$.*

As we will see in Chapter 5, net systems can be unfolded into *occurrence net systems.*

2.1.4 Further Restrictions

In concurrency theory it is common to focus on systems that are *image-finite.* Furthermore, we will restrict our attention to *concurrency-degree finite* systems.

Definition 2.1.14. Let S be a system.

S is *image-finite* iff for each $w \in Act^*$ the set $\{s \in S_S \mid s^i \xrightarrow{w} s\}$ is finite.

Let $s \in S_S$. We define the *smallest upper bound on the number of transitions that can occur concurrently* at s by

$$\begin{aligned} \mathit{cbound}_S(s) = \;& \min\{\kappa \mid \forall r \in \mathit{Runs}_S(s). \\ & (\forall k, l \in [1, |r|].\ k \neq l \Rightarrow r[k]\ I_S\ r[l]) \implies |r| \leq \kappa\}. \end{aligned}$$

S is *concurrency-degree finite* iff for each $s \in S_S$, $\mathit{cbound}_S(s) \in \mathbb{N}_0$.

Restriction 2.1.2. We will only consider systems that are image-finite and concurrency-degree finite.

By the first axiom of independence finitely-branching processes are always concurrency-degree finite. Thus, finite-state systems, SBPP and BPP will naturally satisfy this condition. Ultimately, it only imposes a restriction on our decomposition theory and the coincidence result for S-systems in Section 4.3.

2.2 hp and hhp Bisimilarity

We now provide the formal definitions of the notions of bisimilarity that are central to this thesis, *hp* and *hhp bisimilarity*; we also define *chhp bisimilarity.* For examples and informal explanations we refer the reader to Section 1.2.1.

hp bisimilarity relates two systems whose behaviour can be bisimulated while ensuring that the matching history grows pomset isomorphic. Technically, this can be realized by basing hp bisimulation on pairs of *synchronous runs* [JM96]. Intuitively, two runs are synchronous if their induced pomsets are isomorphic, and both runs correspond to the same linearization of the associated pomset isomorphism class.

Definition 2.2.1 (synchronous runs). Let S_1 and S_2 be two systems. Let r_1 and r_2 be runs of S_1, and S_2 respectively. We say r_1 and r_2 are *synchronous* iff the identity function on $\{1, 2, \ldots, |r_1|\}$ is an isomorphism between the pomset of r_1 and the pomset of r_2. We denote the set of synchronous runs of S_1 and S_2 by $SRuns(S_1, S_2)$.

Definition 2.2.2 (hp bisimilarity). A *history preserving* (short: *hp*) *bisimulation* between two systems S_1 and S_2 consists of a set $\mathcal{H} \subseteq Runs(S_1) \times Runs(S_2)$ such that

1. Whenever $(r_1, r_2) \in \mathcal{H}$, then r_1 and r_2 are synchronous.
2. $(\varepsilon, \varepsilon) \in \mathcal{H}$.
3. Whenever $(r_1, r_2) \in \mathcal{H}$ and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, then there exists t_2 such that $r_2 \xrightarrow{t_2} r_2.t_2$ and $(r_1.t_1, r_2.t_2) \in \mathcal{H}$.
4. Vice versa.

We say S_1 and S_2 are *hp bisimilar*, written $S_1 \sim_{hp} S_2$, iff there is a hp bisimulation relating them.

hhp bisimilarity is obtained from hp bisimilarity by the addition of a *backtracking* requirement:

Definition 2.2.3 (backtracking). Let S be a system, $r = t_1 \ldots t_n \in Runs(S)$.

For $t \in T_S$, we say t is *backtrack enabled* in r, written $t \in BEn(r)$, iff there is $i \in [1, n]$ such that $t_i = t$, and $\forall j \in [i+1, n].\ t_j\ I_S\ t_i$. This means that i is a maximal element in $pom(r)$.

If $t \in BEn(r)$ we define $\delta(r, t)$ to be the result of deleting the last occurrence of t in r, i.e. $\delta(r, t) = t_1 \ldots t_{i-1} t_{i+1} \ldots t_n$ iff $last(r, t) = i$. Note that, given $t \in BEn(r)$, we have $\delta(r, t) \in Runs(S)$ due to the first axiom of independence.

Definition 2.2.4 (hhp bisimilarity). A hp bisimulation is *hereditary* (short: *h*) when it further satisfies

5. Whenever $(r_1, r_2) \in \mathcal{H}$ and $t_1 \in BEn(r_1)$, then $t_2 = r_2[last(r_1, t_1)] \in BEn(r_2)$ and $(\delta(r_1, t_1), \delta(r_2, t_2)) \in \mathcal{H}$.

   Whenever $(r_1, r_2) \in \mathcal{H}$ and $t_2 \in BEn(r_2)$, then $t_1 = r_1[last(r_2, t_2)] \in BEn(r_1)$ and $(\delta(r_1, t_1), \delta(r_2, t_2)) \in \mathcal{H}$.

We say S_1 and S_2 are *hereditary hp* (short: *hhp*) *bisimilar*, written $S_1 \sim_{hhp} S_2$, iff there is a hhp bisimulation relating them.

Furthermore, we define *chhp bisimilarity*, which complements the backtracking condition with a *padding* requirement:

Definition 2.2.5 (chhp bisimilarity). A hhp bisimulation is *coherent* (short: c) when it further satisfies

6. Whenever $(r_1.w_1, r_2.w_2)$, $(r_1.t_1, r_2.t_2) \in \mathcal{H}$ for some $w_1 \in T^*_{S_1}$, $w_2 \in T^*_{S_2}$, $t_1 \in T_{S_1}$, and $t_2 \in T_{S_2}$ such that $|w_1| = |w_2|$, $t_1\ I_{S_1}\ w_1$, and $t_2\ I_{S_2}\ w_2$, then $(r_1.t_1.w_1, r_2.t_2.w_2) \in \mathcal{H}$.

We say S_1 and S_2 are *coherent hhp* (short: *chhp*) *bisimilar*, written $S_1 \sim_{chhp} S_2$, iff there is a chhp bisimulation relating them.

It is straightforward to show that $\sim_{hp}$, $\sim_{hhp}$, and $\sim_{chhp}$ indeed define equivalence relations on systems. Sometimes, in the context of two systems S_1 and S_2, we shall also use $\sim_{(h)hp}$ to denote the set $\bigcup\{\mathcal{H} : \mathcal{H} \text{ is a (h)hp bisimulation}\}$. It is easy to see that hp and hhp bisimulations are closed under union. Thus, when used in this way $\sim_{(h)hp}$ denotes the largest (h)hp bisimulation relating S_1 and S_2.

A More Compact Definition. Given two systems S_1 and S_2, it is trivial that one can regard a relation $R \subseteq \{\, (r_1, r_2) \in T^*_{S_1} \times T^*_{S_2} \mid |r_1| = |r_2| \}$ as a language over the alphabet $T_{S_1} \times T_{S_2}$, and vice versa. For us, this means a pair of synchronous runs can be understood as a 'joint run' of pairs of transitions rather than a pair of separate runs.

Convention 2.2.1. We shall regard a pair of synchronous runs as an element of $(T_{S_1} \times T_{S_2})^*$ whenever it is convenient, and freely switch between the two views.

We carry over our notations for transition sequences of Def. 2.1.10 (e.g. $|r|$ and $r[i]$) to elements of $(T_{S_1} \times T_{S_2})^*$ in the obvious way.

For r, $r' \in \mathit{SRuns}(S_1, S_2)$, $t \in T_{S_1} \times T_{S_2}$, where we assume $r = (r_1, r_2)$ and $t = (t_1, t_2)$, we write $r \xrightarrow{t} r'$ iff $r' = (r_1.t_1, r_2.t_2)$. We refer to $(\mathit{SRuns}(S_1, S_2), T_{S_1} \times T_{S_2}, \rightarrow)$ as the *hp transition system*, and denote it by $\mathcal{T}_{hp}$. If $\rightarrow$ is not clear from the context we also write $\rightarrow_{hp}$.

In this spirit, backtracking can compactly be understood as the backtracking of pairs of transitions in joint runs with respect to a notion of *joint independence*. Formally, this is motivated by:

Proposition 2.2.1. *Let S_1, S_2 be two systems, and let $(r_1, r_2) \in \mathit{SRuns}(S_1, S_2)$.*

1. $\forall i \in [1, |r_1|].\ r_1[i] \in \mathit{BEn}(r_1) \iff r_2[i] \in \mathit{BEn}(r_2)$.

2. $t_1 \in BEn(r_1)$, $t_2 \in BEn(r_2)$ & $last(r_1, t_1) = last(r_2, t_2)$
$$\implies (\delta(r_1, t_1), \delta(r_2, t_2)) \in SRuns(S_1, S_2).$$

Proof. This is straightforward from the definitions. □

Definition 2.2.6 (joint independence, backtracking). Let S_1, S_2 be two systems.

The *(joint) independene relation of* S_1 *and* S_2, $I_{(S_1,S_2)} \subseteq (T_{S_1} \times T_{S_2}) \times (T_{S_1} \times T_{S_2})$, is defined by: $(t_1, t_2)\ I_{(S_1,S_2)}\ (t'_1, t'_2)$ iff $t_1\ I_{S_1}\ t'_1$ & $t_2\ I_{S_2}\ t'_2$. We lift $I_{(S_1,S_2)}$ to sequences and sets of pairs of transitions in the usual way.

Let $r \in SRuns(S_1, S_2)$, and $t \in T_{S_1} \times T_{S_2}$. t is *backtrack enabled* in r, written $t \in BEn(r)$, iff there is $i \in [1, |r|]$ such that $t_i = t$, and $\forall j \in [i+1, |r|].\ t_j\ I_{(S_1,S_2)}\ t$. We employ the notation $\delta(r, t)$ analogously to above.

Now we can define in a more compact fashion:

Definition 2.2.7 (hhp and chhp bisimulation, compact). A hp bisimulation is *hereditary* (short: h) when it further satisfies

5. Whenever $r \in \mathcal{H}$ and $t \in BEn(r)$ for some $t \in T_{S_1} \times T_{S_2}$, then $\delta(r, t) \in \mathcal{H}$.

A hhp bisimulation is *coherent* (short: c) when it further satisfies

6. Whenever $r.w$, $r.t \in \mathcal{H}$ for some $w \in (T_{S_1} \times T_{S_2})^*$, $t \in T_{S_1} \times T_{S_2}$ such that $t\ I_{(S_1,S_2)}\ w$, then $r.t.w \in \mathcal{H}$.

Prefix-closed Bisimulations. Throughout the thesis, it will often be convenient to work with bisimulations that are *prefix-closed*.

Definition 2.2.8 (prefix-closed). Let Σ be an alphabet. We say a language $L \subseteq \Sigma^*$ is *prefix-closed* iff $rt \in L$, where $t \in \Sigma$, implies $r \in L$.

By definition every hhp bisimulation is prefix-closed. This does not apply to hp bisimulations. However, since prefix-closed hp bisimulations correspond to bisimulations that have been built up inductively from $(\varepsilon, \varepsilon)$ without adding "any redundant tuples", we can extract from any given hp bisimulation one that is prefix-closed.

Fact 2.2.1. *Two systems are hp bisimilar iff there exists a prefix-closed hp bisimulation relating them.*

A Game Characterization for (h)hp Bisimilarity. It is common to view bisimulation equivalences in terms of two-player games (e.g. [Sti96]). This is particularly useful when exhibiting that two systems are non-bisimilar. An extension of the game characterization of classical bisimilarity that captures hhp bisimilarity was presented in [NC94]. We describe the game characterization of hhp bisimilarity informally (in the style of [Jan95]), incorporating a characterization for hp bisimilarity.

1. *Prerequisites.* There are two players, Opponent and Player, and a pair of systems S_1 and S_2. A configuration of a play is a pair $(r_1, r_2) \in \mathit{SRuns}(S_1, S_2)$. The initial configuration is $(\varepsilon, \varepsilon)$.

2. *Rule for hp bisimilarity.* Let (r_1, r_2) be the current configuration. Opponent chooses one of the two systems, say S_1 (S_2), and picks a transition t_1 (t_2) that is enabled at r_1 (r_2). Player has to respond by executing a transition, t_2 (t_1), in the opposite system such that the two extended runs stay synchronous. The new configuration is $(r_1.t_1, r_2.t_2)$.

3. *Additional rule for hhp bisimilarity.* Alternatively to a forwards move, having chosen one of the two systems, say S_1 (S_2), Opponent can pick a transition t_1 (t_2) that is backtrack enabled at r_1 (r_2). Player has to respond by backtracking $t_2 := r_2[\mathit{last}(r_1, t_1)]$ ($t_1 := r_1[\mathit{last}(r_2, t_2)]$) in r_2 (r_1). The new configuration is $(\delta(r_1, t_1), \delta(r_2, t_2))$.

4. *Result.* The play continues like this forever, in which case Player wins, or until either Player or Opponent is unable to move (being his or her turn), in which case the other participant wins.

Fact 2.2.2. *Player has a winning strategy in the (h)hp bisimulation game iff S_1 and S_2 are (h)hp bisimilar; in other words, Opponent has a winning strategy in the (h)hp bisimulation game iff S_1 and S_2 are not (h)hp bisimilar.*

Note that it immediately follows that hhp bisimilarity is co-semi decidable: it is easy to see that we can compute all counter-strategies in diagonal fashion step by step.

2.3 The Decidability of hp Bisimilarity

The key insight behind the proofs of the decidability of hp bisimilarity [Vog91, JM96] is the following fact: two synchronous runs stay synchronous after the addition of a pair of transitions iff the new transitions have the same label, and the *maximal causes* of the new transitions in the pomsets associated with the two runs are the same.

Definition 2.3.1. Let $p = (E_p, <_p, L_p, l_p)$ be a pomset and e, $e' \in E_p$. Event e' is a *maximal cause* of event e in p iff $e' <_p e$ and there is no event $e'' \in E_p$ such that $e' <_p e'' <_p e$.

Let S be a system, and $r \in Runs(S)$. For any $i \in [1, |r|]$, we denote the set of *maximal causes of* i in $pom(r)$ by $mcauses(r, i)$. Let $t \in T_S$ such that $r \xrightarrow{t}$. We define the *maximal causes of* t w.r.t. r as $mcauses(r, t) := mcauses(r.t, |r.t|)$.

Proposition 2.3.1. *Assume two systems S_1 and S_2. Let $(r_1, r_2) \in SRuns(S_1, S_2)$, and for $i \in 1, 2$ let $t_i \in T_{S_i}$ such that $r_i \xrightarrow{t_i}$. We have:*

$$(r_1.t_1, r_2.t_2) \in SRuns(S_1, S_2) \iff \begin{array}{l} l_{S_1}(t_1) = l_{S_2}(t_2) \\ \&\ mcauses(r_1, t_1) = mcauses(r_2, t_2). \end{array}$$

Proof. Straightforward. □

This means, to determine whether two runs grow synchronously, we do not need to keep the entire history, but it is sufficient to record only those events that can act as maximal causes. The next step is to find a notion that captures the corresponding segment of the history, but is finite in the sense that there are only finitely many instances of it. In any partial order run the events that can act as maximal causes correspond to distinct transitions. This is so because a transition cannot be independent of itself. Thus, as one possibility we can simply take the set of transition-pomsets restricted to the most-recent occurrences of their transitions. For finite nets there are clearly only finitely many such restricted pomsets. What we have just described is the notion of *growth-sites* defined by Jategaonkar and Meyer. Vogler develops a different concept specific to net systems: with *ordered markings* (*OMs*) the most-recent history is captured by imposing a pre-order on the places of each marking.

Instead of defining hp bisimulation on runs we can now base our notion on growth-sites or OMs; for this, Jategaonkar and Meyer define *growth-site correspondences* (short: *gsc's*), the compressed analogue to synchronous runs. The resulting bisimulations are called *gsc-bisimulation*, and *OM-bisimulation*, respectively. Jategaonkar and Meyer show that gsc-bisimilarity is indeed equivalent to hp bisimilarity. Vogler proves the analogue for OM-bisimilarity. As there are only finitely many growth-sites or OMs for a system, these bisimilarities can be decided by exhaustive search. The decidability of hp bisimilarity is then immediate.

Since we will refer to them later on, we provide the formal definition of growth-sites and gsc's:

Definition 2.3.2. Let S be a system. Let $r = t_1 \dots t_n$ be a transition-sequence of S. Event $i \in [1, n]$ is a *most recent occurrence* of transition t in r iff $t_i = t$ and $t_j \neq t$ for all $j \in [i{+}1, n]$. Let *growth-sites*(r) be the transition-pomset of r restricted to the most-recent occurrences of the transitions in r. We define the set of *growth-site states* (short: *gs-states*) of S as $GSs(S) := \{(s, g) \mid \exists r \in Runs(S).\ s^i \xrightarrow{r} s\ \&\ g = \mathit{growth\text{-}sites}(r)\}$. We carry over $\xrightarrow{w}$, $[w\rangle$, and *mcauses* to gs-states in the obvious way.

Let S_1, S_2 be two systems, and $(r_1, r_2) \in SRuns(S_1, S_2)$. We define the *growth-site correspondence* of (r_1, r_2), denoted by $gsc(r_1, r_2)$, to be the partial identity function $\beta : \mathit{growth\text{-}sites}(r_1) \to \mathit{growth\text{-}sites}(r_2)$ such that $\beta(i) = j$ iff $i = j$ and $i \in E_{\mathit{growth\text{-}sites}(r_1)} \cap E_{\mathit{growth\text{-}sites}(r_2)}$. We define the set of *gsc-states* of S_1 and S_2 as $GSCs(S_1, S_2) := \{(s_1, s_2, \beta) \mid \exists (r_1, r_2) \in SRuns(S_1, S_2).\ s_1^i \xrightarrow{r_1} s_1, s_2^i \xrightarrow{r_2} s_2\ \&\ \beta = gsc(r_1, r_2)\}$.

We consider growth-sites and gsc's only up to isomorphism. This ensures that $GSs(S)$ and $GSCs(S_1, S_2)$ are finite sets whenever S, S_1, and S_2 are finite-state.

We will not make active use of OMs, but we shall briefly bring out the insights underlying the OM quotienting. We will build on them and the associated concepts later on, in particular in Section 6.14.

If we consider net systems we can make use of the extra information provided by places. Place holdings mediate between transition occurrences and ultimately establish the causal dependencies between the events of a run. This induces a natural concept of *immediate cause*: let r be a run, and i, $j \in [1, |r|]$ be events of r; j is an immediate cause of i in r iff the firing of i has consumed a place that was generated by j. In the following, we fix a net system $\mathcal{N}$; the following convention will ensure that every event has an immediate cause.

Definition 2.3.3. Let $r \in Runs(\mathcal{N})$. We define *init* to be the *initial cause*, that is we define $init <_r i$ for all $i \in [1, |r|]$. We assume our definition of *mcauses* to be adapted accordingly. Extending the standard order on natural numbers, we set $init < i$ for all $i \in [1, |r|]$; we also set $init + 1 := 1$. We consider the initial cause *init* to be an occurrence of the *initial transition*, which we also call *init*; in other words, we define $r[init] := init$. Further, we define $init^\bullet := M_0$. In the context of a pair of synchronous runs, we use *init* short for $(init, init)$.

Definition 2.3.4. Let $r \in Runs(\mathcal{N})$, and M such that $M_0[r\rangle M$.

Let $p \in M$, and $i \in \{init\} \cup [1, |r|]$. i is the *generator (event) of p w.r.t. r* iff $p \in (r[i])^\bullet$ and $\forall j \in [i+1, |r|].\ p \notin {}^\bullet(r[j])$. It is easy to see that each $p \in M$ has exactly one generator w.r.t. r. We refer to the unique generator of p w.r.t. r by $gen(r,p)$. We lift the notation $gen(r,p)$ to sets of places $P \subseteq M$ in the obvious way.

Let $t \in T_N$ such that $r[t\rangle$. We say i is an *immediate cause of t w.r.t. r* iff there is $p \in {}^\bullet t$ such that $i = gen(r,p)$. In other words, we define $icauses(r,t) := gen(r, {}^\bullet t)$.

The concept of maximal causes can then be captured as follows: the maximal causes of an event i in a run r are exactly given by the *unsubsumed* immediate causes of i in r.

Proposition 2.3.2. *Let $r \in Runs(\mathcal{N})$, and $t \in T_N$ such that $r[t\rangle$. We have:*

$$mcauses(r,t) = \{i \in icauses(r,t) \mid \nexists j \in icauses(r,t).\ i <_r j\}.$$

Proof. Straightforward. □

2.4 Conventions

We explain some notation and conventions that will be employed throughout the thesis. Most of this is standard, but note that in the last paragraph we describe several conventions very specific to this thesis.

Sequences. Let Σ be a finite alphabet. We use Σ^* to denote the set of finite sequences over Σ, and Σ^ω to denote the set of ω-sequences over Σ, where $\omega = \{1, 2, \ldots\}$. We write ε for the empty sequence. Let $r \in \Sigma^*$. For $A \subseteq \Sigma$, let $r \uparrow A$ denote the projection of r onto A, i.e. the sequence obtained by erasing from r all occurrences of letters which are not in A. We use $set(r)$ to denote the set of letters occurring on r. $Prefixes(r)$ stands for the set of prefixes of r. Let $\gamma \in \Sigma^\omega$. $FinPrefixes(\gamma)$ is the set of finite prefixes of γ. Given $r \in FinPrefixes(\gamma)$, we use $\gamma - r$ to denote the ω-sequence obtained by deleting r from the beginning of γ.

Relations, Functions, Structures. Let A and B be sets. We use Id_A to denote the identity relation on A. Let R, R_1, and R_2 be relations on A. We use R^{-1} for the inverse of R, and R_1R_2 for the composition of R_1 and R_2. $\mathcal{P}(A)$ is the power set of A. Given $(a,b) \in A \times B$, the cartesian product of A and B, we use $proj_1(a,b)$ to refer to a and $proj_2(a,b)$ to refer to b. Given a function $f : A \to B$, we allow us to apply f to subsets of A as well as elements of A; that is we employ f as $f : \mathcal{P}(A) \to \mathcal{P}(B)$ with $f(A_s) = \{f(a) \mid a \in A_s\}$. Conversely, given $g : \mathcal{P}(A) \to \mathcal{P}(B)$, if it is clear that singleton sets are mapped to singleton sets, we allow us to employ g as $g : A \to B$ in the obvious way. Given a function f and a domain D, we use $f \upharpoonright_D$ for f restricted to D. Given two structures U and V, we write $U \cong V$ to say U and V are equivalent under the associated notion of isomorphism. Let $(P, \leq)$ be a partially ordered set. Given $x \in P$, we define $\uparrow x = \{y \in P \mid y \geq x\}$ and $\downarrow x = \{y \in P \mid y \leq x\}$.

Specific to this Thesis. If possible without ambiguity, for ease of notation, we leave indices away or simplify them, e.g.: in the context of a system S we write I for I_S; given a system S_1 we write T_1 for T_{S_1}. Conversely, we add indices to resolve ambiguity when we work with more than one system or net, e.g. we write $BEn_S(r)$, or $[t\rangle_N$.
If i is used to range over 1 and 2 then we define $\bar{i}$ as follows: if $i = 1$ then $\bar{i} = 2$, otherwise $\bar{i} = 1$.
Let S_1 and S_2 be two systems. For a pair $(r_1, r_2) \in \mathit{SRuns}(S_1, S_2)$ we use r as a short notation. Similarly, we write t for a pair $(t_1, t_2) \in T_1 \times T_2$. Further, we write $r \xrightarrow{t} r'$ when we have (r_1, r_2), $(r'_1, r'_2) \in \mathit{SRuns}(S_1, S_2)$ and $(t_1, t_2) \in T_1 \times T_2$ such that $r_1 \xrightarrow{t_1} r'_1$ and $r_2 \xrightarrow{t_2} r'_2$. In the same spirit we use S short for (S_1, S_2) to abbreviate indices such as in $BEn_{(S_1,S_2)}(r)$. Conversely, given $r \in \mathit{SRuns}(S_1, S_2)$ we write r_1 and r_2 short for $\mathit{proj}_1(r)$, and $\mathit{proj}_2(r)$ respectively. Given $t \in T_1 \times T_2$ we adopt the analogous convention.

2.5 Further Concepts

2.5.1 Bisimulation Approximations

For proving the soundness of tableaux systems that will be exhibited in Chapter 5, we give an alternative definition of hp and hhp bisimilarity based on bisimulation approximations. This is analogous to how classical bisimilarity was originally defined in [Mil80].

Definition 2.5.1. Let S_1, S_2 be two systems, and $n \in \mathbb{N}_0$. A set $\mathcal{H}$ is a *hp bisimulation approximation of degree* n for S_1, S_2 if

1. Whenever $r \in \mathcal{H}$ then $r \in \mathit{SRuns}(S_1, S_2)$.
2. $\varepsilon \in \mathcal{H}$.
3. Whenever $r \in \mathcal{H}$, $|r| < n$, and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, then there exists t_2 such that $r_2 \xrightarrow{t_2} r_2.t_2$, and $r.(t_1, t_2) \in \mathcal{H}$.

4. Vice versa.

For two systems S_1 and S_2, we write $S_1 \sim^n_{hp} S_2$ iff there is a hp bisimulation approximation of degree n relating them.

Definition 2.5.2. Let $n \in \mathbb{N}_0$. A *hhp bisimulation approximation of degree* n is a hp bisimulation approximation of degree n that further satisfies

5. Whenever $r \in \mathcal{H}$ and $t \in BEn(r)$ then $\delta(r,t) \in \mathcal{H}$.

For two systems S_1 and S_2, we write $S_1 \sim^n_{hhp} S_2$ iff there is a hhp approximation of degree n relating them.

With the standard argument we obtain:

Lemma 2.5.1. *For image-finite systems,*

$$\sim_{hp} = \bigcap_{n=0}^{\infty} \sim^n_{hp} .$$

Lemma 2.5.2. *For image-finite systems,*

$$\sim_{hhp} = \bigcap_{n=0}^{\infty} \sim^n_{hhp} .$$

Lemma 2.5.2 immediately implies that hhp bisimilarity is co semi-decidable: if two systems S_1 and S_2 are not hhp bisimilar then there must be an n such that $S_1 \not\sim^n_{hhp} S_2$. It is easy to see that checking whether $S_1 \sim^n_{hhp} S_2$ is decidable for each n.

2.5.2 Shuffle Product

From the first axiom of independence it is clear that concurrency has to do with being able to *shuffle* computations. Following [Pin97] we define:

Definition 2.5.3 (shuffle). Let A be an alphabet. The *shuffle* of n words $u_1, \ldots, u_n \in A^*$ is the set $u_1 \otimes \cdots \otimes u_n$ of all words of the form

$$u_{1,1}u_{2,1}\cdots u_{n,1}u_{1,2}u_{2,2}\cdots u_{n,2}\cdots u_{1,k}u_{2,k}\cdots u_{n,k}$$

with $k \geq 0$, $u_{i,j} \in A^*$, such that $u_{i,1}u_{i,2}\cdots u_{i,k} = u_i$ for $1 \leq i \leq n$. The *shuffle* of k languages $L_1, \ldots, L_k$ is the language

$$L_1 \otimes \cdots \otimes L_k = \bigcup \{u_1 \otimes \cdots \otimes u_k \mid u_1 \in L_1, \ldots, u_k \in L_k\}.$$

The following is a natural property of shuffle:

Proposition 2.5.1. *We assume a finite alphabet A, and* pairwise disjoint *sub-alphabets $A_1, \ldots, A_n$. Let $w_i \in A_i^*$ for $i \in [1,n]$, and $w \in w_1 \otimes \cdots \otimes w_n$. We have:*

$$\forall i \in [1,n].\ w_i = w \uparrow A_i.$$

Via shuffle we can infer from existing runs to new runs:

Proposition 2.5.2. *Assume a net system $\mathcal{N}$. Let $M \in Reach(\mathcal{N})$, M_s, $M'_s \subseteq M$ such that $M_s \cap M'_s = \emptyset$. Then for any $L \subseteq Runs(M_s)$, $L' \subseteq Runs(M'_s)$ we have:*

$$L \otimes L' \subseteq Runs(M_s \cup M'_s).$$

This is similar for synchronous runs:

Proposition 2.5.3. *Assume two net systems $\mathcal{N}_1$, $\mathcal{N}_2$. For $i \in \{1,2\}$ let $M_i \in Reach(\mathcal{N}_i)$ and M_i^s, $M_i^{s\prime}$ such that $M_i^s, M_i^{s\prime} \subseteq M_i$ and $M_i^s \cap M_i^{s\prime} = \emptyset$. Then for any $L \subseteq SRuns(M_1^s, M_2^s)$, $L' \subseteq SRuns(M_1^{s\prime}, M_2^{s\prime})$ we have:*

$$L \otimes L' \subseteq SRuns(M_1^s \cup M_1^{s\prime}, M_2^s \cup M_2^{s\prime}).$$

Chapter 3

Approximating hhp Bisimilarity

3.1 Introduction

In this chapter we examine the backtracking condition so as to understand how this seemingly small addition can give so much power to hhp bisimilarity. When considering the game characterization it is easy to see that backtracking has two dimensions:

1. the number of transitions over which Opponent may backtrack during a backtracking move (in other words the depth of backtracking), and

2. the number of backtracking moves Opponent is allowed to bring into play during a game.

In hhp bisimilarity backtracking is unbounded with respect to both of these parameters. It is, however, a priori not clear whether the distinguishing and computational power of the bisimilarity depends on this. As the main result of the chapter we prove that this is indeed the case.

By restricting the hereditary condition along the two dimensions we obtain two families of bounded backtracking bisimilarities. As we will see each of them forms a decreasing chain that approximates hhp bisimilarity from above, starting with hp bisimilarity. On the one hand, we show that both of these hierarchies are strict. This establishes that the distinguishing power of hhp bisimilarity can only be achieved by unbounded backtracking. On the other hand, we prove that in both hierarchies each level is decidable, which in turn implies that the computational power of hhp bisimilarity also depends on the unboundedness of the two dimensions.

We will see that the hierarchy insights can directly be applied to the decidability problem of hhp bisimilarity; in particular we obtain decidability for two subclasses: bounded asynchronous systems and systems with transitive independence relation.

The chapter is organized as follows: first, we explain a technicality concerning the definition of (h)hp bisimilarity, which we require later in this chapter. Then, we attend to the two hierarchies: for each of them we give the necessary description, show strictness, and prove the decidability result. After that follows the section on applications to the general decidability problem. We then conclude with some final remarks.

3.2 A Technicality

We now give an alternative way of defining (h)hp bisimilarity, which corresponds to how interleaving bisimilarities are usually defined. This will make it possible for us to show that the two hierarchies indeed approximate hhp bisimilarity.

It is clear that instead of comparing separate transition systems with specified initial state one can just as well compare the states of a large transition system which covers all the behaviour one is interested in. Typically, the large transition system will be given by SOS-rules as the semantics of some process algebra. Accordingly, a notion of bisimilarity can be defined as a relation on states of a specified transition system rather than as a relation between separate systems. This approach has the theoretical advantage that the bisimilarity can be defined as the union of all bisimulations, and it will typically amount to the largest bisimulation.

We take an analogous approach: we fix a lats (without initial state) S, and define hp and hhp bisimilarity as relations on runs, or more precisely synchronous runs over S. We adapt the notions of runs and synchronous runs to the fact that S does not have a designated initial state:

Definition 3.2.1. We define the set of *runs* (relative to S) by

$$\mathit{Runs} = \bigcup_{s \in S_S} \{(s, r) \mid r \in \mathit{Runs}(S, s)\},$$

and the set of *synchronous runs* (relative to S) by

$$\mathit{SRuns} = \{(r_1, r_2) \mid r_1, r_2 \in \mathit{Runs} \text{ such that } \mathit{proj}_2(r_1), \mathit{proj}_2(r_2) \text{ are synchronous}\}.$$

Further, we assume $\rightarrow$, BEn, δ, etc. to be adapted accordingly. To abbreviate, we set $T = T_S \times T_S$. Then, we are ready to define:

Definition 3.2.2 ((h)hp bisimulation, (h)hp bisimilarity). A *hp bisimulation* is a binary relation $\mathcal{H} \subseteq \mathit{SRuns}$ that satifies

1. If $(r_1, r_2) \in \mathcal{H}$ and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, then there is some t_2 so that $r_2 \xrightarrow{t_2} r_2.t_2$ and $(r_1.t_1, r_2.t_2) \in \mathcal{H}$.

2. Vice versa.

A hp bisimulation $\mathcal{H}$ is *hereditary* when it further satisfies

3. If $r \in \mathcal{H}$ and $t \in BEn(r)$, then $\delta(r, t) \in \mathcal{H}$.

We say two runs r_1 and r_2 are *(h)hp bisimilar*, written $r_1 \sim_{(h)hp} r_2$, iff $(r_1, r_2) \in \mathcal{H}$ for some (h)hp bisimulation $\mathcal{H}$. That is, we define:

$$\sim_{(h)hp} = \bigcup \{\mathcal{H} : \mathcal{H} \text{ is a (h)hp bisimulation}\}.$$

We also define *prefix-closed hp bisimilarity* by:

$$\sim'_{hp} = \bigcup \{\mathcal{H} : \mathcal{H} \text{ is a prefix-closed hp bisimulation}\}.$$

Hp and hhp bisimulation are preserved by various operations on relations:

Proposition 3.2.1. *Let $\mathcal{H}_i$ be a (h)hp bisimulation for $i = 1, 2, \ldots$. The following relations are all (h)hp bisimulations: (1) Id_{Runs}, (2) $\mathcal{H}_i^{-1}$, (3) $\mathcal{H}_1\mathcal{H}_2$, (4) $\bigcup_{i \in I} \mathcal{H}_i$.*

With this it is easy to prove that:

Proposition 3.2.2.

1. *$\sim_{(h)hp}$ is the largest (h)hp bisimulation.*

2. *$\sim_{(h)hp}$ is an equivalence relation.*

In this framework we define (h)hp bisimilarity between systems as a derived notion:

Definition 3.2.3. We say two systems S_1 and S_2 are (h)hp bisimilar iff we have $((s_1^i, \varepsilon), (s_2^i, \varepsilon)) \in \sim_{(h)hp}$, where $\sim_{(h)hp}$ is interpreted relative to the disjoint union of both systems.

It is straightforward to show that this relation on systems coincides with (h)hp bisimilarity as defined in Section 2.2. In the following, we will switch freely between the two technical frameworks, and work with whatever is more convenient. It will always be clear from the context which definition we use. Note that *prefix-closed* hp bisimilarity, $\sim'_{hp}$, induces the same relation on systems as $\sim_{hp}$: as explained in Section 2.2 from any given hp bisimulation between two systems it is always possible to extract one that is prefix-closed.

3.3 Bounding the Depth of Backtracking

3.3.1 Definition: (n)hhp Bisimilarity

We start with the first dimension of the backtracking condition, and constrain the number of transitions over which one may backtrack. For all $n \in \mathbb{N}_0$ we define:

Definition 3.3.1 ((n)hhp bisimulation, (n)hhp bisimilarity).

A hp bisimulation $\mathcal{H}$ is *(n)hereditary* (short: *(n)h*) when it further satisfies

3. Whenever $r \in \mathcal{H}$ and $t \in BEn(r)$ for some t such that $last(r,t) \geq |r| - n$, then $\delta(r,t) \in \mathcal{H}$.

We say two runs r_1 and r_2 are *(n)hhp bisimilar*, written $r_1 \sim_{(n)hhp} r_2$, iff $(r_1, r_2) \in \mathcal{H}$ for some (n)hhp bisimulation $\mathcal{H}$. That is, we define:

$$\sim_{(n)hhp} = \bigcup\{\mathcal{H} : \mathcal{H} \text{ is a (n)hhp bisimulation}\}.$$

It is easy to verify that analogously to Prop. 3.2.1 and 3.2.2 we have:

Proposition 3.3.1. *For all $n \in \mathbb{N}_0$ the following holds:*

1. *Let $\mathcal{H}_i$ be a (n)hhp bisimulation for $i = 1, 2, \ldots$. The following relations are all (n)hhp bisimulations: (1) $Id_{\mathcal{P}}$, (2) $\mathcal{H}_i^{-1}$, (3) $\mathcal{H}_1\mathcal{H}_2$, (4) $\bigcup_{i \in I} \mathcal{H}_i$.*

2. *$\sim_{(n)hhp}$ is the largest (n)hhp bisimulation.*

3. *$\sim_{(n)hhp}$ is an equivalence relation.*

We now show that the (n)hhp bisimilarities indeed approximate hhp bisimilarity from above, starting with prefix-closed hp bisimilarity[1].

Clearly, the (0)hhp bisimulations correspond to the prefix-closed hp bisimulations, and so we confirm:

Proposition 3.3.2. $\sim_{(0)hhp} = \sim'_{hp}$.

It is also immediate that any (m)hhp bisimulation satisfies the conditions to be a (k)hhp bisimulation for all $k \leq m$; this gives us:

Proposition 3.3.3. *Let $k, m \in \mathbb{N}_0$. If $k \leq m$ then $\sim_{(k)hhp} \supseteq \sim_{(m)hhp}$.*

[1] We could define the (n)hhp bisimulations in a way such that the hierarchy would start at hp bisimilarity just as well.

In other words, our equivalence relations form a decreasing chain:

$$\sim_{(0)hhp} \supseteq \sim_{(1)hhp} \supseteq \sim_{(2)hhp} \supseteq \ \cdots \ \supseteq \sim_{(k)hhp} \supseteq \ \cdots$$

We will denote this chain by $\langle \sim_{(n)hhp} \rangle_{n \in \omega}$.

Further, any hhp bisimulation is certainly an (n)hhp bisimulation for all $n \in \mathbb{N}_0$, and hence:

Proposition 3.3.4. *For all* $n \in \mathbb{N}_0$*,* $\sim_{hhp} \subseteq \sim_{(n)hhp}$.

Finally, we show that the chain $\langle \sim_{(n)hhp} \rangle_{n \in \omega}$ properly approximates $\sim_{hhp}$, in that $\sim_{hhp}$ is the limit of the (n)hhp bisimilarities. Note that the proof is only valid in our context of image-finite systems.

Proposition 3.3.5. $\sim_{hhp} = \bigcap_{n \in \omega} \sim_{(n)hhp}$.

Proof. For the sake of shorter notation, we set $\mathcal{H}_{lim} = \bigcap_{n \in \omega} \sim_{(n)hhp}$.

$\sim_{hhp} \subseteq \mathcal{H}_{lim}$ follows directly from Prop. 3.3.4. To prove the other direction, we will show that $\mathcal{H}_{lim}$ is a hhp bisimulation. Since $\sim_{hhp}$ is the largest hhp bisimulation, this is clearly sufficient to establish $\mathcal{H}_{lim} \subseteq \sim_{hhp}$. $\mathcal{H}_{lim}$ is obviously a relation on synchronous runs, and so we can go ahead and check whether properties (1) - (3) of Def. 3.2.2 hold.

To prove (1) we let $r \in \mathcal{H}_{lim}$ and assume $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1. We need to find t_2 such that $r_2 \xrightarrow{t_2} r_2.t_2$ and $r.t \in \mathcal{H}_{lim}$. Consider that for all n, $\sim_{(n)hhp}$ is a hp bisimulation, and $r \in \sim_{(n)hhp}$ by assumption. Then, it is clear that each $\sim_{(n)hhp}$ contains some match for t_1 at r, i.e. some t_2 so that $r_2 \xrightarrow{t_2} r_2.t_2$ and $r.t \in \sim_{(n)hhp}$.

For each $\sim_{(n)hhp}$ such matches come from the following set of candidates: $Cands = \{t_2 \in T_S \mid r_2 \xrightarrow{t_2} \ \& \ l(t_2) = l(t_1)\}$. By image-finiteness $Cands$ must be a finite set. But then there is at least one $t_2 \in Cands$ that appears infinitely often in the chain $\langle \sim_{(n)hhp} \rangle_{n \in \omega}$ as a match of t_1 at r. By Prop. 3.3.3 we infer that $r.(t_1, t_2) \in \sim_{(n)hhp}$ for all n, and hence $r.(t_1, t_2) \in \mathcal{H}_{lim}$. Altogether, t_2 is a match as required.

Property (2) follows from the symmetric argument.

(3) is easy to prove. Assume $r \in \mathcal{H}_{lim}$ and $t \in BEn(r)$. Surely, $last(r, t) \geq |r| - k$ for some $k \in \mathbb{N}_0$, and so $\delta(r, t) \in \sim_{(n)hhp}$ for all $n \geq k$. With Prop. 3.3.3 we clearly obtain $\delta(r, t) \in \sim_{(n)hhp}$ for all $n < k$, and hence $\delta(r, t) \in \mathcal{H}_{lim}$. $\square$

3.3.2 Strictness of the Hierarchy

Having seen that the (n)hhp bisimilarities form a hierarchy that approximates hhp bisimilarity, we would like to know whether this hierarchy is strict in that the chain of (n)hhp bisimilarities is *strictly* decreasing.

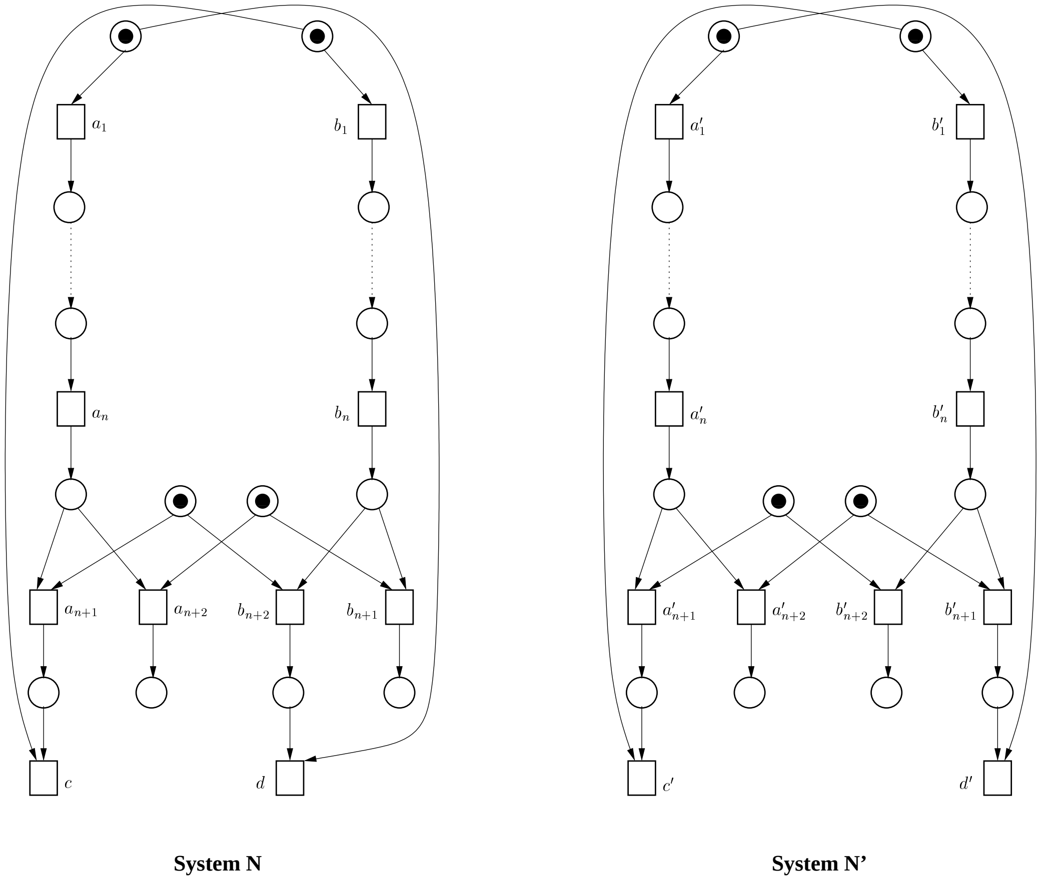


Figure 3.1: Two nets N and N' that are (n)hhp but not (n+1)hhp bisimilar. Note that for $n = 0$ one obtains the two systems of counter-example 1 (Figure 1.7)

The standard example of Figure 1.7 shows that (0)hhp bisimilarity is strictly weaker than (1)hhp bisimilarity. There is an elegant generalization of this counter-example, which discriminates between (n)hhp and (n+1)hhp bisimilarity:

Lemma 3.3.1. *For any $n \in \mathbb{N}_0$, N and N' of Figure 3.1 are (n) but not (n+1)hhp bisimilar.*

Proof. Let us first argue that N and N' are not hhp bisimilar. In any hp bisimulation we must match a_i with a'_i, and b_i with b'_i for $1 \leq i \leq n$. Then, one option in N' is to perform a'_{n+1} and b'_{n+1}. These transitions have to be matched with either a_{n+1} and b_{n+1}, or a_{n+2} and b_{n+2} respectively. Suppose we choose the match a_{n+1}, b_{n+1}. We can now backtrack all the a-transitions such that d' becomes enabled in N'. But no d-action is possible in N. If we choose a_{n+2}, b_{n+2} as our match, we can backtrack all the b-transitions. Then a c-action becomes possible in N', but not in N. So, indeed N and N' are not hhp bisimilar. It is easy to see that using this strategy we never need to backtrack over more than $n + 1$ transitions (in fact, in N and N' we can never backtrack over more than $n + 1$ transitions).

Thus, N and N' are not (n+1)hhp bisimilar either.

The above counter-strategy does not apply for (n)hhp bisimilarity; instead, we can proceed as follows to match the critical $n+1$ transitions. Say we have to match a'_{n+1}, and b'_{n+1} has not been fired yet, i.e. we can still choose between a_{n+1} and a_{n+2} as a match. We make our match dependent on the first transition in the history. Assume it is an a-transition. Then, it is safe to match a'_{n+1} with a_{n+1}, which determines that b'_{n+1} is later matched with b_{n+1}. For d' to become enabled in N' we need to backtrack all the a-transitions; however, there will be $n+1$ b-transitions following the first a, so this is not possible. A symmetrical argument applies if the first action was a b-action, and similarly for the remaining cases. □

It follows that the hierarchy is strict, and together with Prop. 3.3.3 we have:

Theorem 3.3.1. *Let $k, m \in \mathbb{N}_0$. If $k < m$ then $\sim_{(k)hhp} \supset \sim_{(m)hhp}$.*
In other words, $\langle \sim_{(n)hhp} \rangle_{n \in \omega}$ is a strictly decreasing chain.

3.3.3 Decidability of (n)hhp Bisimilarity

We now show that for any $n \in \mathbb{N}_0$, (n)hhp bisimilarity is decidable for finite-state systems. The idea behind our proof is that we can define hhp and (n)hhp bisimulation in a 'forward fashion'. At each tuple we keep a matching directive that prescribes how transitions are going to be matched from this point onwards. The matching directive allows us to express the backtracking requirement as a property of the matching directives of two 'connected' tuples.

To characterize hhp bisimulation in this manner we need to record the matching of the entire future. Because of this the forwards characterization merely shifts the difficulty of the decidability of hhp bisimilarity from the past to the future: now we are confronted with an infinity of possible futures. This is not the case for (n)hhp bisimilarity. But we shall see that it is sufficient to record future matchings of length n. Our proof builds on this fact and the insights gained in the proofs of the decidability of hp bisimilarity.

Below is the definition of (n)Dhp bisimulation, our forwards characterization of (n)hhp bisimulation.

Definition 3.3.2. A *(n)Dhp bisimulation* between two systems S_1 and S_2 consists of a set $\mathcal{H}_D$ of triples (r_1, r_2, D) such that

(i) r_1 is a run of S_1, r_2 is a run of S_2, and r_1 and r_2 are synchronous. The matching directive D is a *non-empty* and *prefix-closed* set of pairs of words

(w_1, w_2), such that w_1 is a transition-sequence of S_1, w_2 of S_2 respectively, and $|w_1| = |w_2| \leq n$.

(ii) For some D, $(\varepsilon, \varepsilon, D) \in \mathcal{H}_D$.

(iii) Whenever $(r_1, r_2, D) \in \mathcal{H}_D$, and $w \in D$ for some w, such that $|w| < n$, and for some t_1 we have $r_1.w_1 \xrightarrow{t_1} r_1.w_1.t_1$, then there is some t_2 such that $(w_1.t_1, w_2.t_2) \in D$.

Note that $(\varepsilon, \varepsilon) \in D$ because D is prefix-closed and non-empty.

(iv) Vice versa.

(v) Whenever $(r_1, r_2, D) \in \mathcal{H}_D$, and $(t_1, t_2) \in D$, then there is some D', such that $(r_1.t_1, r_2.t_2, D') \in \mathcal{H}_D$ and

(a) $\forall w$ s.t. $|w| < n$. $tw \in D \Leftrightarrow w \in D'$.

(b) $\forall w'$. $w' \in D' \wedge t\ I\ t'$ for all $t' \in w' \Rightarrow w' \in D$.

We say two systems S_1 and S_2 are *(n)Dhp bisimilar*, written $S_1 \sim_{(n)Dhp} S_2$, iff there is a (n)Dhp bisimulation relating them.

We show that for all $n \in \mathbb{N}_0$ (n)Dhp bisimilarity is indeed equivalent to (n)hhp bisimilarity. In the proof we make use of the fact that it is sufficient to consider only *prefix-closed* (n)Dhp bisimulations: they correspond to bisimulations that are built up inductively from the empty runs without adding any "redundant tuples".

Definition 3.3.3. We say a (n)Dhp bisimulation $\mathcal{H}_D$ is *prefix-closed* iff whenever $(r_1.t_1, r_2.t_2, D') \in \mathcal{H}_D$, then there is $(r_1, r_2, D) \in \mathcal{H}_D$ for some D such that $t \in D$ and

1. $\forall w$ s.t. $|w| < n$. $tw \in D \Leftrightarrow w \in D'$.

2. $\forall w'$. $w' \in D' \wedge t\ I\ t'$ for all $t' \in w' \Rightarrow w' \in D$.

Fact 3.3.1. *If two systems are (n)Dhp bisimilar then there exists a* prefix-closed *(n)Dhp bisimulation relating them.*

Lemma 3.3.2. *For any $n \in \mathbb{N}_0$, two systems are (n)hhp bisimilar iff they are (n)Dhp bisimilar.*

Proof. For the 'only if'-direction let $\mathcal{H}$ be a (n)hhp bisimulation relating two systems S_1 and S_2. Note that by definition $\mathcal{H}$ is prefix-closed. We define a set $\mathcal{H}_D$ by assigning a matching directive D to every $r \in \mathcal{H}$ as follows:

$$\mathcal{H}_D = \{(r, D) \mid r \in \mathcal{H} \ \& \ D = \{w \mid |w| \leq n \ \& \ r.w \in \mathcal{H}\}\}$$

Our claim is that $\mathcal{H}_D$ is a (n)Dhp bisimulation for S_1 and S_2. Prefix-closure of D is given by prefix-closure of $\mathcal{H}$; then property (i) of Def. 3.3.2 clearly holds. Properties (ii), (iii), and (iv) are also trivial.

To see that property (v) holds, let $(r_1, r_2, D) \in \mathcal{H}_D$ and $(t_1, t_2) \in D$. Then, due to the way D is defined there is D' such that $(r.t, D') \in \mathcal{H}_D$. Condition (a) is clearly satisfied, also due the way matching directives are added to the tuples. To check condition (b) assume we have $w' \in D' \wedge t \ I \ t'$ for all $t' \in w'$. But then we have $r.t.w' \in \mathcal{H}$ with t being backtrack enabled. The fact that $|w'| \leq n$ together with the (n)hereditary condition (Def. 3.3.1) implies that $r.w' \in \mathcal{H}$. Hence, by definition of D we have $w' \in D$ as required.

For the 'if'-direction assume $\mathcal{H}_D$ to be a prefix-closed (n)Dhp bisimulation relating two systems S_1 and S_2. Define $\mathcal{H}$ by simply ignoring the matching directive D of triples $(r_1, r_2, D) \in \mathcal{H}_D$. It is easy to verify that $\mathcal{H}$ is a hp bisimulation for S_1 and S_2 (Def. 2.2.2). We show that in addition $\mathcal{H}$ is (n)hereditary (Def. 3.3.1). Let $r.t.w \in \mathcal{H}$ such that t is backtrack enabled, and $|w| \leq n$. By prefix-closure of $\mathcal{H}_D$ we have $(r, D), (r.t, D') \in \mathcal{H}_D$ for some D, D' such that $t \in D$, $w \in D'$, and the two conditions of property (v) of Def. 3.3.2 are satisfied. But then we have $w \in D$ by condition (b), and hence $(r.w, D'') \in \mathcal{H}_D$ for some D'' as required. $\square$

Now that we have expressed the backtracking condition in a forwards fashion, we can proceed along the lines of the decidability proofs for hp bisimilarity (see Section 2.3). Instead of defining (n)Dhp bisimulation on synchronous runs we can base the notion on gsc's (or OMs for net systems) just as well; we call the resulting equivalence (n)Dgsc bisimilarity. The proof that (n)Dgsc bisimilarity indeed coincides with (n)Dhp bisimilarity is a straightforward adaptation of the proof in [JM96]. Since there are only finitely many matching directives of size n, (n)Dgsc bisimilarity can also be decided by exhaustive search. Consequently, (n)Dhp bisimilarity is decidable, and with it (n)hhp bisimilarity.

Theorem 3.3.2. *For any $n \in \mathbb{N}_0$, it is decidable whether two finite-state systems are (n)hhp bisimilar.*

3.4 Bounding the Number of Backtrack Moves

3.4.1 Definition: (n)nhp Bimilarity

We now turn to the second dimension of backtracking, and constrain the number of backtracking moves. Transferred to the relational framework this amounts to restricting the recursive depth of the backtracking requirement. So, we inductively define for all $n \in \mathbb{N}_0$:

Definition 3.4.1 ((0)nhp bisimulation, (0)nhp bisimilarity). Every hp bisimulation is *(0)nested* (short: *(0)n*).

Two runs r_1 and r_2 are *(0)nhp bisimilar*, written $r_1 \sim_{(0)nhp} r_2$, iff r_1 and r_2 are hp bisimilar. That is, we define $\sim_{(0)nhp} = \sim_{hp}$.

Definition 3.4.2 ((n+1)nhp bisimulation, (n+1)nhp bisimilarity). A hp bisimulation $\mathcal{H}$ is *(n+1)nested* (short: *(n+1)n*) when it further satisfies

3. If $r \in \mathcal{H}$ and $t \in BEn(r)$ for some t, then $\delta(r,t) \in \sim_{(n)nhp}$.

We say r_1 and r_2 are *(n+1)nhp bisimilar*, written $r_1 \sim_{(n+1)nhp} r_2$, iff $(r_1, r_2) \in \mathcal{H}$ for some (n+1)nhp bisimulation $\mathcal{H}$. That is, we define:

$$\sim_{(n+1)nhp} = \bigcup \{\mathcal{H} : \mathcal{H} \text{ is a (n+1)nhp bisimulation}\}.$$

Similarly to before, we have:

Proposition 3.4.1. *For all $n \in \mathbb{N}_0$ the following holds:*

1. *Let $\mathcal{H}_i$ be a (n)nhp bisimulation for $i = 1, 2, \ldots$. The following relations are all (n)nhp bisimulations: (1) $Id_{\mathcal{P}}$, (2) $\mathcal{H}_i^{-1}$, (3) $\mathcal{H}_1\mathcal{H}_2$, (4) $\bigcup_{i \in I} \mathcal{H}_i$.*
2. *$\sim_{(n)nhp}$ is the largest (n)nhp bisimulation.*
3. *$\sim_{(n)nhp}$ is an equivalence relation.*

The (n)nhp bisimilarities form another chain of equivalences which approximates hhp bisimilarity from above, this time starting with hp bisimilarity. Analogously to the observations in Section 3.3.1 the following can easily be read from the definition:

Proposition 3.4.2. *1. $\sim_{(0)nhp} = \sim_{hp}$.*

2. *Let $k, m \in \mathbb{N}_0$. If $k < m$ then $\sim_{(k)nhp} \supseteq \sim_{(m)nhp}$.*
3. *For all $n \in \mathbb{N}_0$, $\sim_{hhp} \subseteq \sim_{(n)nhp}$.*

The second clause shows that the (n)nhp bisimilarities form a decreasing chain:

$$\sim_{(0)nhp} \supseteq \sim_{(1)nhp} \supseteq \sim_{(2)nhp} \supseteq \ \cdots \ \supseteq \sim_{(k)nhp} \supseteq \ \cdots$$

We denote this second chain by $\langle \sim_{(n)nhp} \rangle_{n\in\omega}$. Presupposing image-finiteness we obtain:

Proposition 3.4.3. $\sim_{hhp} = \bigcap_{n\in\omega} \sim_{(n)nhp}$.

Proof. To abbreviate we set $\mathcal{H}_{lim} = \bigcap_{n\in\omega} \sim_{(n)nhp}$.

The direction $\sim_{hhp} \subseteq \mathcal{H}_{lim}$ is immediate with Prop. 3.4.2(3). The other direction follows if we can establish that $\mathcal{H}_{lim}$ is a hhp bisimulation. To prove that $\mathcal{H}_{lim}$ is a hp bisimulation we can employ the same argumentation as used in the proof of Prop. 3.3.5. Then, it only remains to verify that property (3) of Def. 3.2.2 holds. This is readily done: assume $r \in \mathcal{H}_{lim}$ and $t \in BEn(r)$. We show that for all $n \in \mathbb{N}_0$ $\delta(r,t) \in \sim_{(n)nhp}$, and hence $\delta(r,t) \in \mathcal{H}_{lim}$. Let $n \in \mathbb{N}_0$. By assumption we have $r \in \sim_{(n+1)nhp}$; but then by the (n+1)nested condition (Def. 3.4.2) we obtain $r \in \sim_{(n)nhp}$ as required. □

3.4.2 Strictness of the Hierarchy

The family of counter-examples which demonstrates that the first hierarchy is strict proves strictness for the second hierarchy just as well:

Lemma 3.4.1. *For any $n \in \mathbb{N}_0$, N and N' of Figure 3.1 are (n) but not (n+1)nhp bisimilar.*

Proof. In the proof of Lemma 3.3.1 consider the counter-strategy which demonstrates that N and N' are not hhp bisimilar. It is easy to check that we employ no more than $n+1$ backtracking moves: to backtrack $n+1$ a-transitions, or $n+1$ b-transitions. Thus, N and N' are not (n+1)nhp bisimilar.

To show that N and N' are (n)nhp bisimilar the strategy of Lemma 3.3.1 does not apply; instead we simply proceed as follows. If we encounter a_{n+1} or a_{n+2} before we have matched b_{n+1} or b_{n+2} we will take care of the c: then we match a_{n+1} to a'_{n+1} and a_{n+2} to a'_{n+2}. This implies that later on we will have to match b_{n+1} to b_{n+1} and b_{n+2} to b_{n+2}. The b-match will not be in accord with the d-actions, but it is still safe: to expose the respective d-action we would have to backtrack all $n+1$ a-transitions, while we are only allowed $n+1$ backtracking moves. Symmetrical arguments apply in the remaining cases. □

Theorem 3.4.1. *Let $k, m \in \mathbb{N}_0$. If $k < m$ then $\sim_{(k)nhp} \supset \sim_{(m)nhp}$.*

In other words, $\langle \sim_{(n)nhp} \rangle_{n\in\omega}$ is a strictly decreasing chain.

3.4.3 Decidability of (n)nhp Bisimilarity

We shall now demonstrate that for any $n \in \mathbb{N}_0$, (n)nhp bisimilarity is decidable for finite-state systems. Similarly to Section 3.3.3 our proof relies on an alternative characterization of hhp and (n)nhp bisimilarity, which is 'forwards' in that there is no explicit backtracking, and the histories can be compressed to gsc's. Again, for hhp bisimilarity the alternative characterization merely shifts the complexity of the decidablity problem somewhere else; for (n)nhp bisimilarity, however, the gsc-version of the alternative characterization is based on finite domains only. Thus, this equivalence is decidable, and so is (n)nhp bisimilarity.

Let us be more concrete now: how can we obtain a forwards characterization suitable for deciding the (n)nhp bisimilarities? The idea is to record for each run (or more exactly pair of synchronous runs) r all the runs that are reachable by backtracking from r. We can conveniently keep this information in a 'backtrack tree': r constitutes the root; nodes of depth one correspond to runs reachable by one backtracking move from r; nodes of depth two to runs reachable by one backtracking move from their immediate parent node, or by backtracking twice from r, and so on. To record backtrack information that is relevant to hhp bisimilarity in this manner there is no bound on the depth of the trees required. On the other hand, it is clear that for (n)nhp bisimilarity we only need to keep trees of depth n.

Thus, backtrack trees seem to provide a suitable domain to base our forwards characterization on. There is, however, a further requirement: we need to be able to compute the backtrack tree of a continuation run $r.t$ without inspecting the history, but merely from t and the backtrack tree of r. This is possible due to the following insight: the one-move backtrack runs of a continuation run $r.t$ are fully determined by t and the one-move backtrack runs of r. Formally, and in more detail we have:

Definition 3.4.3. Let $r \in \mathit{SRuns}$.

For $t_b \in T$ we define the set of *one-move t_b-backtrack runs of r* by:

$$\mathit{1btRuns}(t_b, r) = \{r_b \mid \exists i \in \mathit{BEn}(r).\ r_b = \delta(r,i)\ \&\ t_i = t_b\}.^2$$

The set of *one-move backtrack runs of r* is defined by:

$$\mathit{1btRuns}(r) = \{r_b \mid \exists i \in \mathit{BEn}(r).\ r_b = \delta(r,i)\}\ (= \textstyle\bigcup_{t_b \in T} \mathit{1btRuns}(t_b, r)).$$

Proposition 3.4.4. *Let $t, t_b \in T$, and $r', r'.t \in \mathit{SRuns}$.*
We have $r_b \in \mathit{1btRuns}(t_b, r'.t)$ iff one of the following conditions holds:

[2] We always have $|\mathit{1btRuns}(t_b, r)| \leq 1$, but this is not crucial here.

1. $t = t_b$ & $r_b = r'$, *or*

2. $t \; I \; t_b$ & $\exists r'_b \in 1btRuns(t_b, r')$. $r'_b \xrightarrow{t}$ & $r_b = r'_b.t$.[3]

Proof. Easy to see from the definitions. $\square$

Altogether we then proceed as follows: for each $n \in \mathbb{N}_0$ we define a domain $BTTs^{[n]}$, which corresponds to the data structure required to record a backtrack tree of depth n. We associate a partial transition function $tr^{[n]} : BTTs^{[n]} \times T \to BTTs^{[n]}$ with each of these domains, so that $tr^{[n]}$ implements the above insight in recursive fashion. Based on this transition system we then define a family of (n)btt bisimilarities, and show that for each $n \in \mathbb{N}_0$ (n)btt bisimilarity corresponds to (n)nhp bisimilarity.

The family of domains is defined as follows. Note that for technical reasons we additionally keep a transition t at each node excluding the root node; this is intended to specify that the run of the respective node is obtained by backtracking t from the run of the parent node.

Definition 3.4.4. For each $n \in \mathbb{N}_0$, we define the domain of *inner backtrack trees of depth n* inductively as follows:

$$\begin{array}{lcl} IBTTs^{[0]} & = & T \times SRuns, \\ IBTTs^{[n+1]} & = & T \times SRuns \times \mathcal{P}(IBTTs^{[n]}). \end{array}$$

The domain of *(outer) backtrack trees of depth n* is then defined by:

$$\begin{array}{lcl} BTTs^{[0]} & = & SRuns, \\ BTTs^{[n+1]} & = & SRuns \times \mathcal{P}(IBTTs^{[n]}). \end{array}$$

Let ρ range over $BTTs^{[n]}$ and $IBTTs^{[n]}$, and R over $\mathcal{P}(BTTs^{[n]})$ and $\mathcal{P}(IBTTs^{[n]})$; if the domains are not specified it will be clear from the context whether we are concerned with $BTTs^{[n]}$ or $IBTTs^{[n]}$.

Let $\rho \in IBTTs^{[n+1]}$ with $\rho = (t_b, r, R)$. We refer to t_b by t_b^ρ, to r by r^ρ, and to R by R^ρ. We will use similar conventions for $\rho \in IBTTs^{[0]}$, and $\rho \in BTTs^{[n]}$.

We will need the following two operations on (inner) backtrack trees:

Definition 3.4.5. By discarding their innermost level we can prune inner and outer backtrack trees of depth $n+1$ into ones of depth n. For all $n \in \mathbb{N}$, we define a function $prune^{[n]} : IBTTs^{[n]} \to IBTTs^{[n-1]}$ by:

$$\begin{array}{lcl} prune^{[1]}(t, r, R^{[0]}) & = & (t, r), \\ prune^{[n+1]}(t, r, R^{[n]}) & = & (t, r, prune^{[n]}(R^{[n]})), \end{array}$$

[3] $r'_b \xrightarrow{t}$ is always satisfied, but we prefer to make this explicit.

and correspondingly a function $prune^{[n]} : BTTs^{[n]} \to BTTs^{[n-1]}$ by:

$$\begin{array}{lcl} prune^{[1]}(r, R^{[0]}) & = & r, \\ prune^{[n+1]}(r, R^{[n]}) & = & (r, prune^{[n]}(R^{[n]})). \end{array}$$

For all $n \in \mathbb{N}_0$, we define a function $cast^{[n]} : IBTTs^{[n]} \to BTTs^{[n]}$ to typecast inner backtrack trees into outer ones:

$$\begin{array}{lcl} cast^{[0]}(t, r) & = & r, \\ cast^{[n+1]}(t, r, R^{[n]}) & = & (r, R^{[n]}). \end{array}$$

Now comes the partial transition function for backtrack trees; note how it is inspired by Prop. 3.4.4.

Definition 3.4.6. For all $n \in \mathbb{N}_0$, we define a partial transition function for inner backtrack trees, $tr^{[n]} : IBTTs^{[n]} \times T \to IBTTs^{[n]}$, as follows:

$$tr^{[n]}(\rho, t) = \begin{cases} new^{[n]}(\rho, t) & \text{if } r^{\rho} \xrightarrow{t} \ \& \ t \ I \ t_b^{\rho}, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

where $new^{[n]} : IBTTs^{[n]} \times T \to IBTTs^{[n]}$ is inductively defined by:

$$\begin{array}{lcl} new^{[0]}((t_b, r), t) & = & (t_b, r.t), \\ new^{[n+1]}((t_b, r, R), t) & = & (t_b, r.t, tr^{[n]}(R, t) \cup (t, r, prune(R)). \end{array}$$

Correspondingly, we define for all $n \in \mathbb{N}_0$ a partial transition function for backtrack trees, $tr^{[n]} : BTTs^{[n]} \times T \to BTTs^{[n]}$, by:

$$tr^{[n]}(\rho, t) = \begin{cases} new^{[n]}(\rho, t) & \text{if } r^{\rho} \xrightarrow{t}, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

where $new^{[n]} : BTTs^{[n]} \times T \to BTTs^{[n]}$ is inductively given by:

$$\begin{array}{lcl} new^{[0]}(r, t) & = & r.t, \\ new^{[n+1]}((r, R), t) & = & (r.t, tr^{[n]}(R, t) \cup (t, r, prune(R)). \end{array}$$

We use $\rho \xrightarrow{t}$ to express '$tr(\rho, t)$ is defined', and $\rho \xrightarrow{t} \rho'$ as short notation for '$\rho \xrightarrow{t}$ & $\rho' = tr(\rho, t)$'.

We define analogues of *1btRuns* for our family of domains:

Definition 3.4.7. Let $n \in \mathbb{N}$, and $\rho \in BTTs^{[n]}$.

For $t_b \in T$ we define the set of *first-level t_b-backtrack trees of ρ* by:
$1btTrees(t_b, \rho) = \{cast(\rho') \mid \rho' \in R^{\rho} \text{ with } t_b^{\rho'} = t_b\}$.

The set of *first-level backtrack trees of ρ* is defined by:
$1btTrees(\rho) = cast(R^{\rho})$, that is $1btTrees(\rho) = \bigcup_{t_b \in T} 1btTrees(t_b, \rho)$.

Analogously to Prop. 3.4.4 we have:

Proposition 3.4.5. *Let $n \in \mathbb{N}$, $t, t_b \in T$, and $\rho', \rho \in BTTs^{[n]}$ with $\rho = tr(\rho', t)$. We have $\rho_b \in 1btTrees(t_b, \rho)$ iff one of the following conditions holds:*

1. *$t = t_b$ & $\rho_b = prune(\rho')$, or*

2. *$t\ I\ t_b$ & $\exists \rho'_b \in 1btTrees(t_b, \rho')$. $\rho'_b \xrightarrow{t}$ & $\rho_b = tr(\rho'_b, t)$.*

Proof. Easy to see from the definitions. □

At last, we come to define our family of (n)btt bisimilarities:

Definition 3.4.8 ((0)btt bisimulation, (0)btt bisimilarity). A *(0)btt bisimulation* is a relation $\mathcal{B} \subseteq BTTs^{[0]}$ that satisfies

1. If $\rho \equiv (r_1, r_2) \in \mathcal{B}$ and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, then there is some t_2 so that $r_2 \xrightarrow{t_2} r_2.t_2$, $\rho \xrightarrow{t}$, and $tr(\rho, t) \in \mathcal{B}$.

2. Vice versa.

Two runs r_1 and r_2 are *(0)btt bisimilar*, written $r \in \sim_{(0)btt}$, iff $r \in \mathcal{B}$ for some (0)btt bisimulation $\mathcal{B}$. That is, we define: $\sim_{(0)btt} = \bigcup\{\mathcal{B} : \mathcal{B}$ is a (0)btt bisimulation$\}$.

Definition 3.4.9 ((n+1)btt bisimulation, (n+1)btt bisimilarity).
A *(n+1)btt bisimulation* is a relation $\mathcal{B} \subseteq BTTs^{[n+1]}$ that satisfies

1. If $\rho \equiv ((r_1, r_2), R) \in \mathcal{B}$ and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, then there is some t_2 so that $r_2 \xrightarrow{t_2} r_2.t_2$, $\rho \xrightarrow{t}$, and $tr(\rho, t) \in \mathcal{B}$.

2. Vice versa.

3. If $\rho \in \mathcal{B}$ then $1btTrees(\rho) \subseteq \sim_{(n)btt}$.

Two runs r_1 and r_2 are *(n+1)btt bisimilar* w.r.t. $R \subseteq IBTTs^{[n]}$, written $(r, R) \in \sim_{(n+1)btt}$, iff $(r, R) \in \mathcal{B}$ for some (n+1)btt bisimulation $\mathcal{B}$. That is, we define: $\sim_{(n+1)btt} = \bigcup\{\mathcal{B} : \mathcal{B}$ is a (n+1)btt bisimulation$\}$.

We now show that for all $n \in \mathbb{N}_0$, (n)btt bisimilarity and (n)nhp bisimilarity indeed correspond to each other. For this, we translate every pair of synchronous runs into a corresponding backtrack tree.

Definition 3.4.10. For all $n \in \mathbb{N}_0$, and $r \in SRuns$ we inductively define a map $btt^{[n]} : SRuns \to BTTs^{[n]}$ to translate a pair of synchronous runs into a corresponding backtrack tree:

$$\begin{array}{lcl} btt^{[0]}(\varepsilon) & = & \varepsilon, \\ btt^{[n+1]}(\varepsilon) & = & (\varepsilon, \emptyset), \\ btt^{[n]}(r.t) & = & tr^{[n]}(btt^{[n]}(r), t). \end{array}$$

Next, we present some straightforward properties about this map. Note that by clause (2) of Prop. 3.4.6, $btt^{[n]}$ is a function. Clause (1) and (3) are important in view of our plan to show that the (n)btt bisimilarities coincide with the (n)nhp bisimilarities: the two properties ensure that the 'bisimulation'-conditions of the two notions correspond to each other.

Proposition 3.4.6. *For all $n \in \mathbb{N}_0$, and $r \in SRuns$ we have:*

1. *Let $\rho \in BTTs^{[n]}$ with $\rho = btt^{[n]}(r)$, and let $t \in T$.*
   (a) $r \xrightarrow{t} \implies \rho \xrightarrow{t}$ & $tr(\rho, t) = btt^{[n]}(r.t)$.
   (b) $\rho \xrightarrow{t} \implies r \xrightarrow{t}$ & $btt^{[n]}(r.t) = tr(\rho, t)$.
2. *$btt^{[n]}(r)$ is defined.*
3. $r^{btt^{[n]}(r)} = r$.
4. $btt^{[n]}(r) = prune(btt^{[n+1]}(r))$.

Proof. (1), (2), and (3) can be proved in one go by induction on the length of r and inspecting the definitions. (4) can also be shown by induction on the length of r; employ the fact $tr^{[n]}(prune^{[n+1]}(\rho^{[n+1]}), t) = prune^{[n+1]}(tr^{[n+1]}(\rho^{[n+1]}, t))$, which follows by induction on n. □

Now comes the crux of the proof: using Prop. 3.4.4 and 3.4.5 we show that *1btRuns* and *1btTrees* correspond to each other in the following way:

Lemma 3.4.2. *For any $n \in \mathbb{N}_0$, and $r \in SRuns$ we have:*

1. *Let $t_b \in T$. $btt^{[n]}(1btRuns(t_b, r)) = 1btTrees(t_b, btt^{[n+1]}(r))$, and so*
2. $btt^{[n]}(1btRuns(r)) = 1btTrees(btt^{[n+1]}(r))$.

Proof. We only need to prove (1); (2) follows as an immediate consequence. Let $n \in \mathbb{N}_0$, $r \in SRuns$, and $t_b \in T$. We proceed by induction on the length of r. *Base case* $r = \varepsilon$: the property clearly holds since $btt^{[n]}(1btRuns(t_b, r)) = \emptyset = 1btTrees(t_b, btt^{[n+1]}(r))$. *Inductive case* $r = r'.t$: we prove the two inclusions separately.

For the '$\subseteq$'-direction assume $r_b \in 1btRuns(t_b, r)$. Then, by Prop. 3.4.4 one of the following two conditions holds: (1) $t = t_b$ & $r_b = r'$, or (2) $t \ I \ t_b$ & $\exists r'_b \in 1btRuns(t_b, r')$. $r'_b \xrightarrow{t}$ & $r_b = r'_b.t$. We show that in both cases $btt^{[n]}(r_b) \in 1btTrees(t_b, btt^{[n+1]}(r))$.

Case (1): Prop. 3.4.6(1a) implies $btt^{[n+1]}(r') \xrightarrow{t}$, and further by Prop. 3.4.5 we infer $prune(btt^{[n+1]}(r')) \in \mathit{1btTrees}(t, tr(btt^{[n+1]}(r'), t))$. Since $tr(btt^{[n+1]}(r'), t) = btt^{[n+1]}(r'.t)$ (Prop. 3.4.6(1a)), and $prune(btt^{[n+1]}(r')) = btt^{[n]}(r')$ (Prop. 3.4.6(4)) this immediately establishes $btt^{[n]}(r') \in \mathit{1btTrees}(t, btt^{[n+1]}(r))$ as required.

Case (2): we can apply the induction hypothesis to r' and r'_b, and obtain $btt^{[n]}(r'_b) \in \mathit{1btTrees}(t_b, btt^{[n+1]}(r'))$. By Prop. 3.4.6(1a) we have $btt^{[n+1]}(r') \xrightarrow{t}$, and $btt^{[n]}(r'_b) \xrightarrow{t}$. Altogether, with Prop. 3.4.5 we then get $tr(btt^{[n]}(r'_b), t) \in \mathit{1btTrees}(t_b, tr(btt^{[n+1]}(r'), t))$. But with Prop. 3.4.6(1a) this immediately gives us the required fact: $btt^{[n]}(r'_b.t) \in \mathit{1btTrees}(t_b, btt^{[n+1]}(r'.t))$.

To prove the '$\supseteq$'-direction assume $\rho_b \in \mathit{1btTrees}(t_b, btt^{[n+1]}(r))$. Consider $\rho' = btt^{[n+1]}(r')$; by Prop. 3.4.6(2) ρ' is defined, and by Prop. 3.4.6(1a) we have $\rho' \xrightarrow{t}$ & $btt^{[n+1]}(r) = tr(\rho', t)$. Then, by Prop. 3.4.5 there are the following two possible cases: (1) $t = t_b$ & $\rho_b = prune(\rho')$, or (2) $t \; I \; t_b$ & $\exists \rho'_b \in \mathit{1btTrees}(t_b, \rho')$. $\rho'_b \xrightarrow{t}$ & $\rho_b = tr(\rho'_b, t)$. We show that in both cases there is $r_b \in \mathit{1btRuns}(t_b, r)$ with $btt^{[n]}(r_b) = \rho_b$.

Case (1): r' is a run as required: obviously (formally by Prop. 3.4.4), we have $r' \in \mathit{1btRuns}(t_b, r)$, and since $prune(\rho') = btt^{[n]}(r')$ (by Prop. 3.4.6(4)) we certainly have $btt^{[n]}(r') = \rho_b$.

Case (2): we can apply the induction hypothesis to ρ' and ρ'_b, and obtain $r'_b \in \mathit{1btRuns}(t_b, r')$ such that $btt^{[n]}(r'_b) = \rho'_b$. Since $\rho'_b \xrightarrow{t}$ we also have $r'_b \xrightarrow{t}$ (by Prop. 3.4.6(1b)). But then $r'_b.t$ provides a run as required: by Prop. 3.4.4 $r'_b.t \in \mathit{1btRuns}(t_b, r'.t)$, and with $btt^{[n]}(r'_b.t) = tr(\rho'_b, t)$ (by Prop. 3.4.6(1b)) we obtain $btt^{[n]}(r'_b.t) = \rho_b$. □

Now, it is straightforward to prove that (n)btt bisimilarity coincides with (n)nhp bisimilarity in the following sense:

Lemma 3.4.3. *For all $n \in \mathbb{N}_0$, and $r \in SRuns$ we have:*

$$r \in \sim_{(n)nhp} \iff btt^{[n]}(r) \in \sim_{(n)btt} .$$

Proof. We prove the lemma by induction on n. *Base case* $n = 0$: from the definitions it is clear that $\forall r \in SRuns.\ btt^{[0]}(r) = r$, and $\sim_{(0)nhp} = \sim_{hp} = \sim_{(0)btt}$. Hence, the property is immediate. *Inductive Case* $n > 0$: Let $r \in SRuns$. We prove the two directions separately.

To establish the '$\Rightarrow$'-direction we assume a (n)nhp bisimulation $\mathcal{H}$ with $r \in \mathcal{H}$, and show that $\mathcal{B} = btt^{[n]}(\mathcal{H})$ is a (n)btt bisimulation. For this, we need to verify that $\mathcal{B}$ satisfies the three conditions of Def. 3.4.9. With Prop. 3.4.6(3) and (1a) it is easy to see that conditions (1) and (2) indeed hold; condition (3) in turn follows from the induction hypothesis and Lemma 3.4.2.

For the '⇐'-direction let $\mathcal{B}$ be a (n)btt bisimulation such that $btt^{[n]}(r) \in \mathcal{B}$. We define a set $\mathcal{H} \subseteq SRuns$ as follows: $\mathcal{H}$ is the least set satisfying

1. $r \in \mathcal{H}$.
2. Let $t \in T$. If $r \in \mathcal{H}$, and
   (a) there is $\rho, \rho' \in \mathcal{B}$ such that $\rho = btt^{[n]}(r)$ & $\rho \xrightarrow{t} \rho'$, and
   (b) $r \xrightarrow{t}$ & $btt^{[n]}(r.t) = \rho'$

   then $r.t \in \mathcal{H}$.

Fact 3.4.1. *If $r \in \mathcal{H}$ then $btt^{[n]}(r) \in \mathcal{B}$.*

We claim that $\mathcal{H}$ is a (n)nhp bisimulation. Clearly, $r \in \mathcal{H}$; so if this is true then $r \in \sim_{(n)nhp}$. With the above fact, and clause (3) and (1b) of Prop. 3.4.6 it is easy to see that $\mathcal{H}$ is a hp bisimulation (Def. 3.2.2). Further, it follows from the induction hypothesis and Lemma 3.4.2 that $\mathcal{H}$ satisfies the (n)nested condition (Def. 3.4.2). □

This proves that the (n)btt bisimilarities indeed provide an alternative characterization of the (n)nhp bisimilarities. Let us now exploit that the characterization is 'forwards'.

Instead of keeping runs in backtrack trees, we can record their gsc's just as well. Accordingly, we compress each domain $BTTs^{[n]}$ to a domain $gsc\text{-}BTTs^{[n]}$ by substituting $GSCs$ for $SRuns$. The partial transition function carries over easily to this domain. It is then straightforward to check that a corresponding family of (n)gsc-btt bisimilarities coincides with the (n)btt bisimilarities. The $gsc\text{-}BTTs^{[n]}$ domains are surely finite for finite-state systems, and so the (n)gsc-btt bisimilarities can be decided by exhaustive search. To decide (n)nhp bisimilarity we then simply translate the respective runs into their gsc-based backtrack tree (which clearly can be done effectively), and test for (n)gsc-btt bisimilarity.

Theorem 3.4.2. *For any $n \in \mathbb{N}_0$, it is decidable whether two finite-state systems are (n)nhp bisimilar.*

3.5 Application to the Decidability Problem of hhp Bisimilarity

We will now see how the insights about the hierarchies can be applied to the decidability problem of hhp bisimilarity. First, we review how they might have helped to solve the general problem. Then, we identify two system classes for which the hierarchy insights indeed help to establish results: we show that for bounded asynchronous systems and systems with transitive independence deciding hhp bisimilarity reduces to deciding (n)hhp bisimilarity.

3.5.1 Retrospection

The two hierarchies were identified, and the material of the first elaborated when the decidability of hhp bisimilarity was still an open problem. At that time we hoped that the decidability of the (n)hhp bisimilarities would directly help to solve the general problem.

From the strictness result we already knew that hhp bisimilarity does not coincide with (n)hhp bisimilarity for any $n \in \mathbb{N}_0$ (now, this also follows from the undecidability of the former). However, for two fixed systems hhp bisimilarity does fall together with (n)hhp bisimilarity from some $n \in \mathbb{N}_0$ onwards: if two systems are not hhp bisimilar then by Prop. 3.3.5 this will show within some bound n. Thus, the decidability of the general problem would immediately follow, if this bound could be effectively computed for any two finite-state systems. Via the undecidability result it is now clear that this is not possible.

Analogous considerations are valid for the second hierarchy.

Proposition 3.5.1. 1. *For any two (image-finite) systems S_1 and S_2 there is a bound $n \in \mathbb{N}_0$ such that $S_1 \sim_{hhp} S_2$ iff $S_1 \sim_{(n)hhp} S_2$.*

This bound is not effectively computable for finite-state systems.

2. *For any two (image-finite) systems S_1 and S_2 there is a bound $n \in \mathbb{N}_0$ such that $S_1 \sim_{hhp} S_2$ iff $S_1 \sim_{(n)nhp} S_2$.*

This bound is not effectively computable for finite-state systems.

3.5.2 Bounded Asynchronous Systems

For each $n \in \mathbb{N}_0$, we define a behavioural system class for which hhp bisimilarity naturally coincides with (n)hhp bisimilarity. With Theorem 3.3.2 it is immediate that hhp bisimilarity is decidable for the finite-state subsets of these classes.

Definition 3.5.1. Let $n \in \mathbb{N}_0$. A system S is *(n)bounded asynchronous* iff for all $r \in Runs(S)$ we have: $i \in BEn(r) \implies |r| - i \leq n$.

Proposition 3.5.2. *For any $n \in \mathbb{N}_0$ we have:*

1. *Two (n)bounded asynchronous systems are hhp bisimilar iff they are (n)hhp bisimilar.*

2. *It is decidable whether two finite-state (n)bounded asynchronous systems are hhp bisimilar.*

Proof. (1) is immediate by definition; (2) follows from (1) and Theorem 3.3.2. □

We can also consider the union of this family of system classes.

Definition 3.5.2. A system S is *bounded asynchronous* iff S is (n)bounded asynchronous for some $n \in \mathbb{N}_0$.

Note that we have:

Proposition 3.5.3.

1. *It is decidable whether a finite-state system is bounded asynchronous.*

2. *For any finite-state bounded asynchronous system S it is possible to compute the smallest n for which S is (n)bounded asynchronous.*

Proof. (1.) It is easy to check that a finite-state system S fails to be bounded asynchronous if and only if there is $t \in T_S$, $s \in \mathit{Reach}(S)$, and a loop $s = s_0 \xrightarrow{t_1} s_1 \cdots \xrightarrow{t_n} s_n = s$ such that $s \xrightarrow{t}$ & $\forall i \in [1, n].\ t\ I_S\ t_i$. Clearly, this condition can be decided for finite-state systems.

(2.) To compute the smallest bound n for which S is (n)bounded asynchronous we can then simply test for all $t \in T_S$, $s, s' \in \mathit{Reach}(S)$ with $s \xrightarrow{t} s'$ how many transitions independent of t can be computed from s' onwards; as the bound we take the maximum. □

With this fact, decidability for finite-state bounded asynchronous systems is also immediate.

Theorem 3.5.1. *It is decidable whether two finite-state bounded asynchronous systems are hhp bisimilar.*

This is in fact a strong result: hhp bisimilarity becomes decidable as soon as we disallow 'threads' being left behind indefinitely; this is already achieved by systems which satisfy suitably defined criteria of 'concurrency-fairness' and 'thread-liveness'. Also note that bounded asynchrony seems to be closely related (if not equivalent) to the property 'strongly synchronized' of [Maz89].

3.5.3 Systems with Transitive Independence Relation

In trace theory, there are several decision problems which are undecidable in the general case, but which can be decided as soon as one assumes the independence relation to be transitive [DM97]. We will now see that, analogously, it is fruitful for us to consider systems with transitive independence relation.

Definition 3.5.3. An independence relation I over an alphabet Σ is *transitive* if, for every *distinct* $t, t', t'' \in \Sigma$, $t\ I\ t'$ & $t'\ I\ t''$ implies $t\ I\ t''$.

Let S be a system. A transition $t \in T_S$ is a *self-loop* iff $\exists s \in Reach(S).\ s[t\,t\rangle$. (For 1-safe net systems a static condition is: ${}^\bullet t = t^\bullet$.) Intuitively, a self-loop is a transition that can be repeated immediately, i.e. independently of the occurrence of other transitions.

Let us first draw our attention to systems with transitive independence relation that do not contain any self-loops. It is easy to see that for such systems the number of transitions over which we can backtrack is bounded by the size of the maximal independence clique. In other words, a system with maximal independence clique of size k is (k)bounded asynchronous, and hence decidability for finite-state systems of this subclass is immediate.

If a system contains a self-loop that can occur concurrently with another transition, then this system is clearly not bounded asynchronous. However, we can transfer the decidability result to the full class of finite-state systems with transitive independence with the help of another key observation. In every (h)hp bisimulation between two systems with transitive independence, concurrently occurring self-loop transitions have always to be matched with self-loops. Hence, we do not need to consider the unfoldings of such self-loops: it is sufficient to match the first occurrence of such a transition when we make sure that the match is indeed a self-loop. But then the number of transitions over which one can backtrack is again bounded by the size of the maximal independence clique, and so we have established decidability. The precise definition of what it means for a self-loop to occur concurrently in a given context, and the details of the proof can be found in Appendix A.

Theorem 3.5.2. *It is decidable whether two finite-state systems with transitive independence relation are hhp bisimilar.*

For systems with transitive independence and no self-loops it can be proved that hhp bisimilarity coincides with coherent hhp bisimilarity. Based on this insight there is an alternative decidability proof for the finite-state subset of this class, and thereby for the the full class of finite-state systems with transitive independence. Both of the proofs can be found in [Frö98].

3.6 Final Remarks

We have approximated hhp bisimilarity by two hierarchies of bounded backtracking bisimilarities corresponding to the two dimensions of backtracking. By analysing these hierarchies we have found that the distinguishing and computational power of hhp bisimilarity can only be achieved by leaving the two dimensions unbounded. With the help of our hierarchy insights we have also obtained two partial results: decidability of hhp bisimilarity for bounded asynchronous systems and transition systems with independence.

There are some interesting points for further research. One point is to relate the hierarchy insights further to the undecidability result. Inspired by the systems that are employed in [JN00], a second family of counter-examples has been developed that demonstrates strictness for the second hierarchy. The counter-example explicates an aspect of the undecidability proof: it illustrates how the second ingredient of unboundedness gives the power to propagate a piece of information, such as tiling information, indefinitely. This connection has to be worked out in more detail. It still needs to be analysed how the first ingredient of unboundedness manifests itself in the undecidability proof, and whether there is an equally intuitive interpretation. So far, one connection is clear: one central part of the reduction is to encode the two-dimensional grid by two sets of independent transitions; it is obvious that in the corresponding systems Opponent has the *opportunity* to backtrack transitions of arbitrary depth. Note how this connection agrees with our decidability result for bounded asynchronous systems: such systems do not have the expressive power to encode the grid.

This brings us to a symmetric point. For the first hierarchy, bounded asynchronous systems provide a natural system class for which bounded backtracking achieves the same power as hhp bisimilarity. This also helped to establish the decidability of systems with transitive independence. It still has to be checked whether a corresponding restriction for the second hierarchy gives rise to a natural system class, or whether it helps to find results for other interesting subclasses.

Further, one could analyse whether the two hierarchies interact with each other in any way; one could also consider a hierarchy which reflects both dimensions. A general idea for further research is to study the relationship between tiling games and independence systems and/or backtracking; this could lead to useful intuitions and new results. An obvious technical point is to check whether the two hierarchies still approximate hhp bisimilarity in the general case, when lifting our restriction to image-finite systems.

Chapter 4

The Interplay of Causality, Concurrency & Conflict

4.1 Introduction

In this chapter we lay the foundations for the remaining material of the thesis. Having examined the backtracking condition in the previous chapter we now begin with the second and major part of our analysis: we study the coincidence and decidability problem of (hp and) hhp bisimilarity on system classes with restricted behaviour. Thereby, we hope to identify which behavioural aspects of concurrent systems are crucial to the increased expressive and computational power of hhp bisimilarity. We have put forward that backtracking is so powerful because it can expose subtle differences arising from the mixture of the three fundamental situations: causality, concurrency, and conflict. We will substantiate this intuition by exhibiting concrete coincidence and decidability results. This chapter is our starting point: it identifies important behavioural situations, and delivers first insights on the coincidence problem. We shall also prove our first composition and decomposition results. Two of the themes identified here will be continued in the following two chapters: in Chapter 5 we study a structural class with restricted *synchronization*, and in Chapter 6 we investigate a structural characterization of *confusion-free* systems. After concluding this introduction with a first insight, we will proceed as follows:

Section 4.2. Naturally, we start by investigating the basic situations causality, concurrency, and conflict. We find that concurrency and conflict are both crucial to keep hp and hhp bisimilarity distinct, but that this is not the case for causality. Furthermore, we identify *(L&C)-nondeterminism* as a crucial situation. Investigating it will lead us to an interesting counter-example.

Section 4.3. We prove that hp and hhp bisimilarity are composable with respect to *decompositions of systems into independent components*, and that for *bsc-decomposable systems* the two bisimilarities are decomposable with respect to the *set of prime components*. With this it will be straightforward to show that hp and hhp bisimilarity coincide for *parallel compositions of sequential systems*, confirming our intuition that the increased power of hhp bisimilarity relies on the interplay of concurrrency with conflict and/or causality.

Section 4.4. Motivated by the previous section we identify and investigate behavioural situations that witness structural synchronization: three *'synchronization witness'* (short: *SW*) *situations*. On the one hand, we find that taken by themselves each of them is significant but not essential to distinguish hp and hhp bisimilarity. On the other hand, we show that in their entirety they are a necessary condition for *bounded-degree systems*. We conclude that hp and hhp bisimilarity coincide for bounded-degree *communication-free* net systems.

Section 4.5. We consider a well-known behavioural situation which results from the interplay of concurrency and conflict, the situation of *confusion*. We find that confusion is significant for distinguishing hp and hhp bisimilarity, but we also show that the two bisimilarities do *not* coincide for confusion-free systems; this disproves a long-standing conjecture. However, our counter-example leads us to identifying a new kind of confusion, so-called *syn-confusion*.

Section 4.6. Finally, we introduce the well-known concept of *liveness*. This behavioural property will be particularly interesting later on, when we study it in combination with the structural condition of *free choice*: free choice net systems are confusion-free, while *live* free choice net systems additionally appear to exclude syn-confusion.

As in the previous chapter we work at the behavioural level here, and employ lats' as our primary semantic model. We will refer to lats' simply as systems.

4.1.1 A First Intuition

In studying the distinction between hp and hhp bisimilarity we will seek to capture which behavioural aspects are significant or even necessary to construct a counter-example. It is immediate to identify two conditions that any counter-example must provide:

Insight 4.1.1.

1. We must be given the *opportunity* to match in a non-hereditary way: it must be possible to match two interleavings of the same partial order run differently depending on the order in which independent transitions are linearized.

2. The non-hereditary matching must be made *necessary* in that: if we do not seize the provided opportunity to make the matching dependent on the linearization then a difference between the two systems will be exhibited, and a hp bisimulation cannot be obtained.

Accordingly, we can distinguish between two types of scenarios which will be present in any counter-example. Let A and B be two systems that are hp but not hhp bisimilar.

Frame Scenario. In a hp bisimulation relating A and B the matching of some transitions of A, and B respectively, will depend on the order in which they or other transitions are linearized. We call the specific arrangement of such key transitions the *frame scenario* of the counter-example.

MNH Situations. There must be a second kind of key transitions: the ones that together ensure that the non-hereditary matching is made necessary. Typically, this will involve two parallel transitions such that one of them has an indirect effect on the other, in that it can change the 'behavioural environment' of the first. We call such scenarios *MNH situations* ('MNH' is short for 'Match Non-Hereditary').

The difficulty in constructing counter-examples is that such situations must be combined while preserving that the two systems are still hp bisimilar. We illustrate our concepts with the help of the two standard counter-examples:

Counter-example 1 (Figure 1.7). The counter-example's frame situation consists of two conflicting 'independence squares' in each system. a_1 and b_1 can either be matched by a_1' and b_1', or by a_2' and b_2'. Which of the two squares is employed can be made dependent on the order in which a_1 and b_1 are linearized. This is similar for a_2 and b_2, and symmetric in B. The following describes one of the MNH situations employed in A: a_1 and b_1 are independent of each other but a_1 holds an influence over the 'behavioural environment' of b_1: if b_1 occurs first then a transition d can occur causally dependent on b_1. If a_1 occurs before b_1 then the d option is no longer available.

Counter-example 2 (Figure 1.8). Here, the frame situation is made up of three conflicting 'independence squares' in system A, and two conflicting 'independence squares' in system B. Similarly to counter-example 1 the matching of each square can be made dependent on the order in which its transitions are linearized. Since the 'independence squares' are in conflict the third square can easily be incorporated into this scheme. The following describes one of the MNH situations employed in A: a_1 and b_1 are in parallel, but a_1 has an indirect influence on b_1:

assume a_1 and b_1 are still both enabled. If b_1 occurs first then there is the option to execute c_1 in parallel to b_1. If a_1 occurs before b_1 then this opportunity is taken away.

4.2 A Minimum of Behavioural Situations

In this first section we will settle a minimum of behavioural situations which must be allowed to keep hp and hhp bisimilarity distinct from each other. These situations are *concurrency*, *conflict*, and a specific notion of nondeterminism, which we shall call *(L&C)-nondeterminism.* Causality is not a necessary condition. We will also find that (L&C)-nondeterminism can be classified into *(L&C)-nondet. conflict* and *(L&C)-nondet. concurrency*, but that the availability of one of the two is sufficient for non-coincidence. The three essential situations will, of course, also constitute a minimum for keeping hhp bisimilarity undecidable.

4.2.1 Concurrency and Conflict but not Causality

The fact that concurrency and conflict are both necessary to keep hp and hhp bisimilarity apart becomes immediately clear with Insight 4.1.1 and the following observations. Without concurrency there will never be an occasion to match two distinct linearizations of the same computation in a different way, trivially because there will only be one linearization per computation. Thus, if there is no concurrency we cannot fulfill requirement (1.). On the other hand, in the absence of conflict requirement (2.) cannot be met: the behaviour of a conflict-free system consists of a single (possibly infinite) partial order run, and therefore there will not be any alternative behaviour that could make it necessary to match in a non-hereditary way. More detailed proofs of both results can be found in [Frö00b]; here we simply state:

Definition 4.2.1 (sequential systems, conflict-free systems).

A system S is *sequential* iff

$$\forall s \in \mathit{Reach}(S).\ \forall t_1, t_2 \in T_S.\ s[t_1\rangle \ \&\ s[t_2\rangle \quad \Longrightarrow \quad t_1\ D_S\ t_2.$$

A system S is *conflict-free* iff

$$\forall s \in \mathit{Reach}(S).\ \forall t_1, t_2 \in T_S.\ t_1 \neq t_2 \ \&\ s[t_1\rangle \ \&\ s[t_2\rangle \quad \Longrightarrow \quad t_1\ I_S\ t_2.$$

Theorem 4.2.1.

1. *hp and hhp (and classical) bisimilarity coincide for sequential systems.*

2. hp and hhp bisimilarity coincide for conflict-free systems.

The result for sequential systems also follows from the fact that hp bisimilarity naturally coincides with classical bisimilarity in this case, and that adding 'linear' backtracking to classical bisimilarity does not increase its distinguishing power. The latter has been shown in [DNMV90].

Interestingly, a look at the second counter-example (Figure 1.8) tells us that causality is not a necessary condition for non-coincidence: it is easy to see that the two systems are entirely causality-free.

Definition 4.2.2 (causality-free systems). A system S is *causality-free* iff

$$\forall s \in Reach(S).\ \forall t_1, t_2 \in T_S.\ s[t_1 t_2\rangle \implies t_1\ I_S\ t_2.$$

Theorem 4.2.2. *hp and hhp bisimilarity do not coincide in general for causality-free systems.*

S-systems and T-systems. There are two well-studied subclasses of Petri nets which give rise to structural characterizations of sequential and, respectively, conflict-free systems. These so-called *S-systems* and *T-systems* will also be interesting later on with regard to free-choice net systems (cf. Chapter 6).

In S-systems every transition has exactly one pre-place and one post-place, and thereby a transition can neither be used to join a number of input threads, nor to fork an input thread into several output strands.

Definition 4.2.3 (S-graphs, S-systems).

A net N is an *S-graph* iff $\forall t \in T_N.\ |{}^\bullet t| = 1\ \&\ |t^\bullet| = 1$.

A net system $\mathcal{N}$ is an *S-system* iff its underlying net is an S-graph.

Dually, in T-systems every place has exactly one pre-transition and one post-transition. This ensures that such systems do not contain any backwards or forwards branched places, and hence no conflict at all.

Definition 4.2.4 (T-graphs, T-systems).

A net N is a *T-graph* iff $\forall s \in S_N.\ |{}^\bullet s| = 1\ \&\ |s^\bullet| = 1$.

A net system $\mathcal{N}$ is a *T-system* iff its underlying net is a T-graph.

It is clear that all T-systems are conflict-free. On the other hand, S-systems are not necessarily sequential since they can contain several independent components. Instead we have that all *connected* S-systems are sequential.[1] Then with our results of above (Theorem 4.2.1) we immediately get:

[1]Remember that we always assume a net system to be 1-safe.

Theorem 4.2.3.

1. *hp and hhp (and classical) bisimilarity coincide for the class of connected S-systems.*

2. *hp and hhp bisimilarity coincide for the class of T-systems.*

More to the theory of S-systems and T-systems can be found in [RT86] and [DE95].

4.2.2 (L&C)-nondeterminism

Apart from concurrency there is yet another basic behavioural situation which is needed to satisfy requirement (1.) (of Insight 4.1.1). Certainly, it is only possible to make the matching of a transition dependent on the order in which independent behaviour is linearized if there are suitable alternative matches available in the respective systems. It could well be the case that we have so few options of how to build up a bisimulation that we are forced to match in a hereditary way.

By the axioms of independence (see Section 2.1.2) and the fact that our matching preserves independence this need for variety translates into the requirement for a certain kind of nondeterministic choice: at a respective state there must be at least two transitions available that are 'matching-equivalent' in that if one of them is suited to match some transition of the opposite system, say t_2, then the other one will make a suitable match for t_2 just as well. With the insights of Section 2.3 it is not difficult to see that two transitions are 'matching-equivalent' at some gsc state g iff they have the same label and the same maximal causes at g. Accordingly, we define the behavioural situation of *(L&C)-nondeterministic choice*.

Definition 4.2.5 (choice, nondeterministic choice). Let S be a system, t_1, $t_2 \in T_S$, $s \in \mathit{Reach}(S)$, and $g \in \mathit{GSs}(S)$.

The triple (s, t_1, t_2) is a *choice situation (at s)* iff $t_1 \neq t_2$, $s[t_1\rangle$ & $s[t_2\rangle$. Analogously, the triple (g, t_1, t_2) is a *choice situation (at g)* iff $t_1 \neq t_2$, $g[t_1\rangle$ & $g[t_2\rangle$.

Let $c = (s, t_1, t_2)$, $c' = (g, t_1, t_2)$ be choice situations. We say

- c or c' is *nondeterministic w.r.t. the labelling* (short: *(L)-nondet.*) iff $l(t_1) = l(t_2)$,

- c' is *nondeterministic w.r.t. the maximal causes* (short: *(C)-nondet.*) iff $\mathit{mcauses}(g, t_1) = \mathit{mcauses}(g, t_2)$, and

- c' is *nondeterministic w.r.t. labelling and maximal causes* (short: *(L&C)-nondet.*) iff c' is both, (L)- and (C)-nondet..

We can then add (L&C)-nondet. choice as our third basic requirement for non-coincidence. Formally, we have:

Definition 4.2.6 ((L/C)-det. systems). A system S is *deterministic w.r.t. labelling or maximal causes* (short: *(L/C)-det.*) iff for all $g \in GSs(S)$ there is no (L&C)-nondet. choice at g.

Theorem 4.2.4. *hp and hhp bisimilarity coincide for (L/C)-det. systems.*

Proof. Let S_1, S_2 be two systems such that S_1 is (C/L)-det., and let $\mathcal{H}$ be a prefix-closed hp bisimulation relating S_1 and S_2.

Assume $rtw \in \mathcal{H}$ with $t\ I\ w$. By prefix-closure of $\mathcal{H}$, $r \in \mathcal{H}$. Clearly, w_2 is enabled at r_2 just as well, and thus there must be a match for w_2 at r: there is w_1^* such that $(r_1 w_1^*, r_2 w_2) \in \mathcal{H}$. By induction on the length of w we show $w_1 = w_1^*$. This certainly proves $rw \in \mathcal{H}$ as required. Base case $|w| = 0$: There is nothing to prove. Inductive case $|w| > 0$: Assume $w_1 = w_1' t_1'$, $w_2 = w_2' t_2'$, and, integrating the induction hypothesis, $w_1^* = w_1' t_1^*$. On the one hand, we have $l(t_1') = l(t_2')$ and $mcauses(r_1 w_1', t_1') = mcauses(r_2 w_2', t_2')$ because clearly rw must be a pair of synchronous runs. On the other hand, since t_1^* is a match for t_2' at rw', t_1^* must be such that $l(t_1^*) = l(t_2')$ and $mcauses(r_1 w_1', t_1^*) = mcauses(r_2 w_2', t_2')$, and thereby a transition with the same label, and the same maximal causes at $r_1 w_1'$ as t_1'. Since S_1 is (C/L)-det. this means t_1^* must be t_1' as required. □

In fact, the proof shows that the two bisimilarities also coincide if only one of two related systems is (L/C)-det., and this in the strong sense that every prefix-closed hp bisimulation is hereditary.

4.2.3 (L&C)-nondet. Conflict or (L&C)-nondet. Concurrency

In concurrent systems two simultaneously enabled transitions are either in conflict with each other or concurrently executable. Accordingly, we can distinguish between the following two types of choice.

Definition 4.2.7 (#-choice, co-choice). Let S be a system, $t_1, t_2 \in T_S$, $s \in Reach(S)$, $g \in GSs(S)$, and let $c = (s, t_1, t_2)$, $c' = (g, t_1, t_2)$ be choice situations. We say

- c or c' is a *choice situation due to conflict* (short: *#-choice*) iff $t_1\ D_S\ t_2$, and

- c or c' is a *choice situation due to concurrency* (short: *co-choice*) iff $t_1\ I_S\ t_2$.

This classification gives us the opportunity to further analyse whether to keep hp and hhp bismilarity distinct we require (L&C)-nondeterminism in the disguise of *(L&C)-nondet. #-choice*, *(L&C)-nondet. co-choice*, either, or both of the two.

A look at the two counter-examples (Figure 1.7, 1.8) shows that (L&C)-nondet. co-choice is *not* necessary for non-coincidence: the systems do not contain any (L&C)-nondet. co-choice, indeed they do not even contain (L)-nondet. co-choice (or auto-concurrency as it is usually called).

Definition 4.2.8 (auto-concurrency free, (L&C)-co free systems). A system S is *auto-concurrency free* iff for all $s \in \mathit{Reach}(S)$ there is no (L)-nondet. co-choice at s.

A system S is *(L&C)-concurrency free* (short: *(L&C)-co free*) iff for all $g \in \mathit{GSs}(S)$ there is no (L&C)-nondet. co-choice at g.

Theorem 4.2.5. *hp and hhp bisimilarity do not coincide in general for auto-concurrency free systems, and hence not for (L&C)-co free systems.*

At the same time this means (L&C)-nondeterminism can be employed in the disguise of (L&C)-nondet. #-choice to distinguish between hp and hhp bisimilarity. It is more difficult to resolve whether (L&C)-nondet. #-choice is necessary for non-coincidence, or whether alternatively we can employ (L&C)-nondet. co-choice. The two standard counter-examples (Figure 1.7, 1.8) do not give us any help: their frame situations are based on (L&C)-nondet. conflict, and there is no obvious way of converting them into counter-examples that use (L&C)-nondet. co-choice instead (cf. Section 4.1.1). However, we have come up with a new counter-example, which indeed employs (L&C)-nondet. co-choice rather than #-choice. The two systems are presented in Figure 4.1 and 4.2. They are clearly (L&C)-nondet. conflict free, and even (L)-nondet. conflict free.

In both of them there is a bundle of parallel e-transitions and a bundle of parallel f-transitions such that depending on how the conflict between the a- and b-transition, and respectively the c- and d-transition, is resolved, either (1)(b, c) the two bundles can occur in parallel, (2)(a, d) none of the bundles can occur, (3)(b, d) the f-bundle can occur but the e-bundle is prevented from occurring, or (4)(a, c) vice versa. In case (1) the e-threads can synchronize with the f-threads

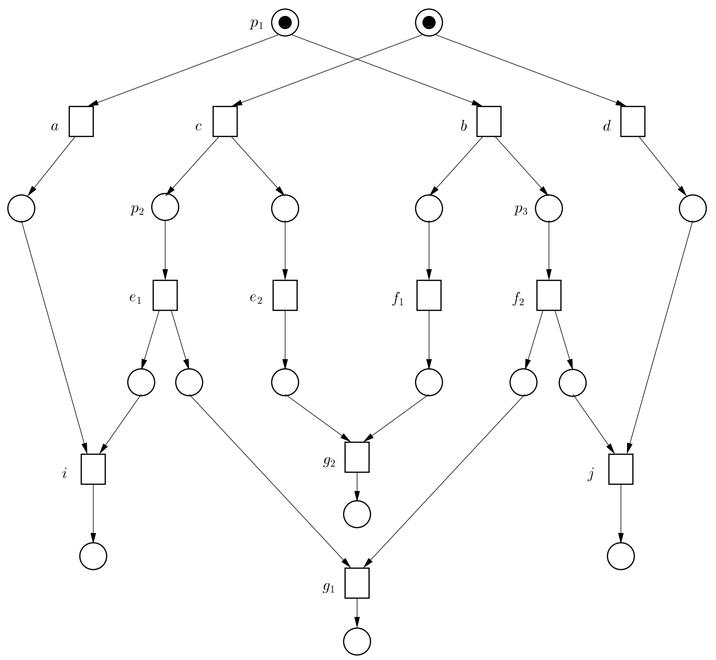


Figure 4.1: Counter-example 3, System S

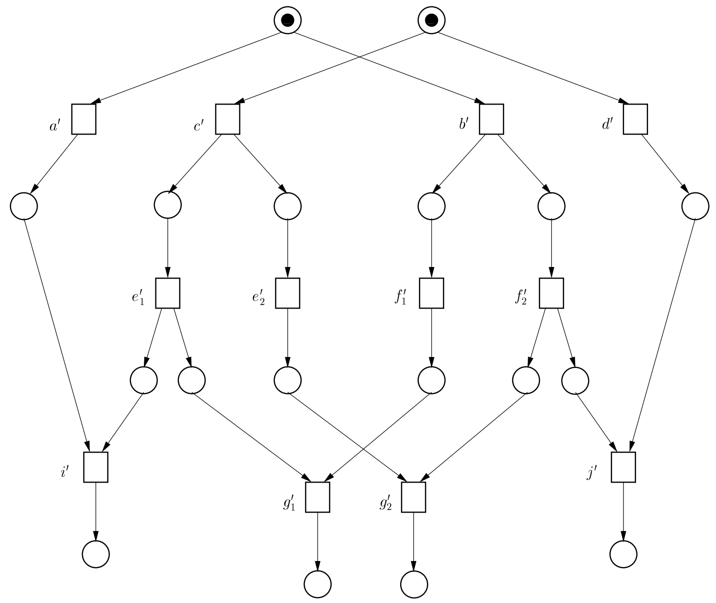


Figure 4.2: Counter-example 3, System S'

via g-transitions; in case (3) one particular thread of the f-bundle can do a j-synchronization; and similarly in case (4) one particular thread of the e-bundle can do an i-synchronization. The only difference between the two systems is that in S the e- and f-transition with the i-, and respectively, j-synchronization option are g-synchronization partners, whereas in S' these transitions have g-synchronizations with the alternative e- or f-transition.

We will see that with the help of backtracking this difference can easily be detected, but that without it we can use non-hereditary matching to disguise it. That is we claim:

Lemma 4.2.1 (counter-example 3). *S and S' of Figure 4.1 and 4.2 are hp bisimilar but not hhp bisimilar.*

Proof. We shall first argue why the two systems are not hhp bisimilar, and then explain why they are hp bisimilar still.

(1.) Opponent (short: *Op*) has the following counter-strategy against Player (short: *Pl*). *Op* opens the game by choosing the c-transition of system S; *Pl* has to answer this move by c'. Then, *Op* picks e_1, and *Pl* can choose between e'_1 and e'_2. We first consider the latter case and assume to be at configuration $(c, c')(e_1, e'_2)$. *Op* does the a-transition, and *Pl* has to react with the a'-transition. But note that *Op* has tricked *Pl* into a losing position: *Op* can do the i-transition, but there is no i-transition enabled in system S', and so *Pl* is stuck.

Let's go back and allow *Pl* to try out the other possibility. We assume configuration $(c, c')(e_1, e'_1)$. *Pl* could now perfectly match the above sequence. But of course, *Op* has something different in mind: *Op* chooses b; *Pl* has to react with b'. Then *Op* picks f_2; *Pl* can now choose between f'_1, and f'_2. Let's try the latter option first. We are at $(c, c')(e_1, e'_1)(b, b')(f_2, f'_2)$. *Op* does g_1, but there is no g-transition *Pl* could use to match this move, and so he is stuck again.

Previously, *Pl* could have reacted with f'_1 though, and then he would have managed this sequence easily. But of course, *Op* will employ a different strategy in this case. We presume configuration $(c, c')(e_1, e'_1)(b, b')(f_2, f'_1)$. *Op* finally brings the backtracking capability into play and backtracks the e_1- and c-transition; *Pl* has to react with the corresponding backtracking move. This brings us to configuration $(b, b')(f_2, f'_1)$. *Op* chooses the d-transition, which is now available; *Pl* reacts with d'. But then *Op* can do the j-synchronization, and *Pl* is definitely stuck this time.

(2.) To see that the two systems are hp bisimilar first note that without *Op*'s power to backtrack *Pl*'s last strategy would have been perfectly successful. In general, *Pl* can take the following strategy to win the game.

For all transitions apart from the e- and f-labelled ones, the matching will be forced by the labelling. *Pl* can match the e-transitions either straight across, that is e_1 to e'_1 and e_2 to e'_2, or diagonally, that is e_1 to e'_2 and e_2 to e'_1 (and vice versa). We call the first option E_s, and the second E_d. This is analogous for the f-transitions, and we shall refer to the corresponding strategies by F_s and F_d. With regard to the g-synchronizations, the strategies E_s and F_d are compatible, and the combination (E_d, F_s) also works fine. For the i-synchronization E_s will do, but not E_d. Similarly, F_s is suitable for the j-synchronization, but F_d is not.

Because $a\#b$ and $c\#d$ ($a'\#b'$ and $c'\#d'$), *Pl* will never have to consider the i-, or respectively, j-transition, and the g-synchronizations in the same game, and thus he can adopt the following strategy: if he has to match c (or c') before b (or b') has occurred then he will orientate himself by the i-synchronization, and employ the matching strategy (E_s, F_d). This is perfectly safe: the i-synchronization is well taken care of, and *Pl* is in no danger from the j-transitions, since they have been put out of action by the c-transitions. Analogously, *Pl* will adopt strategy (E_d, F_s) if he encounters transition b (or b') before c (or c'). Note that there is an overlap when b and c have both occurred before any e- or f-transition has: then naturally both strategies will work. □

Together with our observation that S and S' do not contain any (L&C)-nondet. #-choice nor (L)-nondet. #-choice, Lemma 4.2.1 immediately proves:

Definition 4.2.9 ((L)-conflict free, (L&C)-conflict free systems). A system S is *(L)-conflict free* iff for all $s \in \mathit{Reach}(S)$ there is no (L)-nondet. #-choice at s.

A system S is *(L&C)-conflict free* iff for all $g \in \mathit{GSs}(S)$ there is no (L&C)-nondet. #-choice at g.

Theorem 4.2.6. *hp and hhp bisimilarity do not coincide in general for (L)-conflict free systems, and hence not for (L&C)-conflict free systems.*

Thus, we have shown that (L&C)-nondeterminism may come as either (L&C)-nondet. #-choice *or* (L&C)-nondet. co-choice; the availability of both or one in particular of these situations is not necessary to distinguish hp and hhp bisimilarity.

4.3 Composition and Decomposition Results

We now make a digression into something more structural, and prove our first composition and decomposition results: hp and hhp bisimilarity are composable with respect to *decompositions of systems into independent components* (short: *decompositions*), and for *systems that are decomposable into components each of which is behaviourally strongly connected* (short: *bsc-decomposable systems*) the two bisimilarities are decomposable with respect to the *set of prime components*. As we will see in this section and later on (cf. Section 4.4 and 5.3) these insights provide the key to several coincidence results. Here we will derive that hp and hhp bisimilarity coincide for parallel compositions of sequential systems. We can therefore add: to distinguish the two bisimilarities it is essential that the sequential parts of a system can interact, or communicate, with each other. Behaviourally this means: the increased power of hhp bisimilarity relies on the interplay of concurrency with conflict and/or causality.

The section is organized as follows: we begin with some preliminaries, then follow the composition, and in turn the decomposition, results, and finally we discuss their consequences, which include the coincidence result for parallel compositions of sequential systems.

4.3.1 Preliminaries

First, we present the concept of *concurrent step*; it will be needed for the decomposition result, and also later on (in Section 4.4.3, 5.4.5 and 6.14). Secondly, we introduce the notions central to this section: the concept of *decomposition*, *set of prime components*, and *bsc-decomposable system*.

4.3.1.1 Concurrent Steps

Concurrent steps and *maximal concurrent steps* are defined as follows:

Definition 4.3.1 (csteps, mcsteps). Let S be a system.

A run $r \in Runs(S)$ is a *concurrent step of S* iff we have: $\forall k, l \in [1, |r|]. \; k \neq l \implies k \; co_r \; l$ (or equivalently $r[k] \; I_S \; r[l]$). We denote the *set of concurrent steps of S* by $csteps(S)$.

r is a *maximal concurrent step* (short: *mc step*) *of S* iff $r \in csteps(S)$ and $\forall t \in T_S. \; r.t \notin csteps(S)$. We denote the *set of maximal concurrent steps of S* by $mcsteps(S)$.

Proposition 4.3.1 (facts about csteps and mcsteps). *Let S be a system.*

1. $r.t \in csteps(S) \implies r \in csteps(S)$.
2. *Let* $r, r' \in csteps(S)$. $(\forall t \in r. \; \forall t' \in r'. \; t \; I_S \; t') \implies r.r' \in csteps(S)$.

3. *Let S be non-empty.* $\exists r \in \mathit{csteps}(S).\ |r| \geq 1$.

4. *Let S be non-empty.* $r \in \mathit{mcsteps}(S) \implies |r| \geq 1$.

Proof. (1) and (3) are obvious. (2) follows with the second axiom of independence, and (4) with (3). □

We will make use of two straightforward but important insights about the matching of concurrent steps: in synchronous runs, and hence in hp bisimilarity, concurrent steps are always matched against concurrent steps; furthermore, in hp bisimilarity mc steps are always matched against mc steps.

Proposition 4.3.2 (hp bisimilarity respects csteps and mcsteps). *Let S_1 and S_2 be two systems.*

1. *Let* $(r_1, r_2) \in \mathit{SRuns}(S_1, S_2)$. $r_1 \in \mathit{csteps}(S_1) \iff r_2 \in \mathit{csteps}(S_2)$.

2. *Let* $(r_1, r_2) \in \sim_{hp}$. $r_1 \in \mathit{mcsteps}(S_1) \iff r_2 \in \mathit{mcsteps}(S_2)$.

Proof. (1.) This is a consequence of the fact that synchronous runs are partial order preserving.

(2.) We show that the contrary leads to a contradiction. Let S_1 and S_2 be two systems, and w.l.o.g. assume $r \equiv (r_1, r_2) \in \sim_{hp}$ such that (a) $r_1 \in \mathit{mcsteps}(S_1)$ but (b) $r_2 \notin \mathit{mcsteps}(S_2)$. By (1) and (a) we have $r_2 \in \mathit{csteps}(S_2)$, and considering (b) there must be $t_2 \in T_{S_2}$ with $r_2.t_2 \in \mathit{csteps}(S_2)$. Clearly, t_2 must have a match at r, that is there must be t_1 such that $r.(t_1, t_2) \in \sim_{hp}$. But this contradicts either (1) or the maximality of r_1. □

4.3.1.2 Decomposed Systems

The primary notion of decomposition employed in this section is very simple: we deal with decompositions of systems into components that are completely independent of each other.

Definition 4.3.2 (sub-system terminology). Let S be a system.

We say a system S' is a *sub-system* of S iff

1. $S_{S'} \subseteq S_S$ with $s^i_S \in S_{S'}$,

2. $s^i_{S'} = s^i_S$,

3. $T_{S'} \subseteq T_S$,

4. $\rightarrow_{S'} = \rightarrow_S \cap (S_{S'} \times T_{S'} \times S_{S'})$,

5. $I_{S'} = I_S \cap (T_{S'} \times T_{S'})$,

6. $l_{S'} = l_S \restriction_{T_{S'}}$.

Let S_1 and S_2 be two sub-systems of S. We say S_1 and S_2 are *independent*, written $S_1 \; I_S \; S_2$, iff $T_{S_1} \; I_S \; T_{S_2}$.

The *empty sub-system of* S is defined by $c^S_{empty} = (\{s^i_S\}, s^i_S, \emptyset, \emptyset, \emptyset, \emptyset)$.

The *sub-system of* S *induced by* $T \subseteq T_S$ is defined by:

$$(S_T, s^i_S, T, \rightarrow_S \cap (S_T \times T \times S_T), I_S \cap (T \times T), l_S \restriction_T),$$

where $S_T = \{s \in S_S \mid \exists w.\; s^i_S \xrightarrow{w} s \;\&\; set(w) \subseteq T\}$.

Definition 4.3.3 (decomposition terminology). Let S be a system.

A set $\mathcal{D} = \{c_1, \dots, c_n\}$, $n \in \mathbb{N}$, of sub-systems of S is a *decomposition of* S *into independent components* (short: *decomposition of* S*)* iff

1. $\forall i, j \in [1, n].\; (i \neq j \implies c_i \; I_S \; c_j)$, and

2. $Runs(S) = \bigcup\{\tau_1 \otimes \cdots \otimes \tau_n \mid \tau_i \in Runs(c_i) \text{ for } i \in [1, n]\}$.

A pair $S = (S, \mathcal{D})$ is a *system decomposed into independent components* (short: *decomposed system*) iff S is a system, and $\mathcal{D}$ a decomposition of S. In the context of a decomposed system $(S, \mathcal{D})$ we use the following decomposition functions:

- $K : T_S \rightarrow \mathcal{D}$, defined by $K(t) = c_i \iff t \in T_{c_i}$, and

- $Ks : T^*_S \rightarrow \mathcal{P}(\mathcal{D})$, defined by $Ks(w) = \bigcup_{t \in w} K(t)$.

(K is a function by clause (1) of the definition of decomposition, and the irreflexivity of independence.)

A sub-system c of S is a *divisor of* S iff there exists a decomposition $\mathcal{D}$ of S such that $c \in \mathcal{D}$. If c is a divisor of S we define $S \backslash c$ to be the sub-system of S induced by $T_S \backslash T_c$.

Every system S has at least one decomposition: the one consisting of S itself. Moreover, a system might have many different decompositions. Every *non-empty* system will, however, uniquely decompose into a set of *prime components.*

Definition 4.3.4 (prime systems). Let S be a non-empty system. S is *prime* iff c^S_{empty} and S are the *only* divisors of S.

Fact 4.3.1. *Each non-empty system* S *has a unique decomposition* $\mathcal{D}$ *such that for all* $c \in \mathcal{D}$ c *is prime.*

For the proof we require the following observations on decompositions.

Proposition 4.3.3. *Let $(S, \{c_1, \ldots, c_n\})$ be a decomposed system.*

1. *If C_i is a decomposition of c_i then $\{c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n\} \cup C_i$ is a decomposition of S, for all $i \in [1, n]$.*

2. *$\{c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n\}$ is a decomposition of $S \backslash c_i$, for all $i \in [1, n]$.*

3. *Let p be a divisor of S. If p is prime then p must be a divisor of c_i for some $i \in [1, n]$.*

Proof. (1) and (2) follow easily from the definitions. (3) seems more involved but this is also straightforward if one considers that divisors are defined as concrete sub-systems of the respective system: with the help of the definitions and the two axioms of independence one can show that p must be a divisor of the c_i that satisfies $T_p \cap T_{c_i} \neq \emptyset$ (such c_i must clearly exist). □

Now Fact 4.3.1 can be established following the standard proof of unique prime factorization of natural numbers. (We follow [Nor86].)

Proof (Fact 4.3.1). Let S be a non-empty system. Denote the smallest upper bound on the number of transitions that can occur concurrently at the initial state by $ib(S)$. Since we only consider concurrency-degree finite systems $ib(S) \in \mathbb{N}_0$ (cf. Section 2.1.4). We prove the fact by induction on $ib(S)$, say k.

If $k = 0$ the fact vacuously holds: this case is only possible for empty systems. Assume $k > 0$ and suppose the fact is true for systems S' with $ib(S') \leq k - 1$. First we show that S is decomposable into a set of primes. If S is prime this is immediate. If S is not prime then there must be a decomposition of S, say $\mathcal{D} = \{c_1, \ldots, c_n\}$, such that $n > 1$, and for all $i \in [1, n]$ c_i is non-empty. This means for all $i \in [1, n]$ we must have $ib(c_i) < k$. But then by the induction hypothesis each c_i is decomposable into a set of primes, and so must be S (Prop. 4.3.3(1)).

Secondly, we prove the uniqueness of the factorization. Suppose S can be decomposed into $\{b_1, \ldots, b_n\}$, and $\{c_1, \ldots, c_m\}$ respectively, where $b_1, \ldots, b_n$, and $c_1, \ldots, c_m$ are prime. By Prop. 4.3.3(3) b_1 divides one of the factors in the decomposition $\{c_1, \ldots, c_m\}$. Say b_1 is a divisior of c_1. Since c_1 is prime we conclude $b_1 = c_1$, and so $\{b_2, \ldots, b_n\}$ and $\{c_2, \ldots, c_m\}$ are decompositions of $S \backslash b_1$ (Prop 4.3.3(2)). But clearly $ib(S \backslash b_1) < k$, and thus by the induction hypothesis we can assume that $n = m$ and (possibly on renumbering $c_1, \ldots, c_m$) $b_2 = c_2, \ldots, b_n = c_m$. Hence the two prime decompositions of S are identical. □

Definition 4.3.5 (prime components). Let S be a system. We define the *prime components of* S, denoted by $PComps(S)$, as follows: if S is empty we set $PComps(S) = \emptyset$, otherwise we define $PComps(S)$ to be the decomposition associated with S by Fact 4.3.1. When it is clear that $PComps(S) \neq \emptyset$ we also use the term *prime decomposition*.

Convention 4.3.1. Let S be a system. If it is clear from the context that S is non-empty and there is no other decomposition specified, we shall understand S as the decomposed system $S = (S, PComps(S))$. In particular, we shall then use the decomposition functions K and Ks. If S is empty (or equivalently $PComps(S) = \emptyset$) then we define $Ks(\varepsilon) = \emptyset$. The intuition behind this is: ε is the only possible computation of S, and the set of prime components involved in it is the empty set.

Fact 4.3.2. *Let S be a finite-state system. Then $PComps(S)$ is computable.*[2]

Proof. Given a finite-state system S, we construct a set C of sub-systems of S in three steps, which are clearly computable.

1. We partition T_S into non-empty subsets $T_1, \ldots, T_n$ such that each T_i is a connected component with respect to the dependence relation D, that is two transitions t, t' belong to the same T_i iff $t \; D \; t_1 \; D \cdots D \; t_m \; D \; t'$ for some $t_1, \ldots, t_m \in T_S$.

2. For $i \in [1, n]$ we compute c_i, the sub-system of S induced by T_i.

3. We define $C = \{c_i\}_{i \in [1,n]}$.

We claim that $C = PComps(S)$. If S is empty then $C = \emptyset$ as required. Assume S is non-empty. First, we check that C is a decomposition of S. By definition it is clear that for all distinct $c, c' \in C, c \; I_S \; c'$. For $i \in [1, n]$ let $\tau_i \in Runs(c_i)$. Clearly $\forall i \in [1, n], \tau_i \in Runs(S)$. Then, by repeated use of the second axiom of independence we obtain $\tau_1 \otimes \cdots \otimes \tau_n \subseteq Runs(S)$. Conversely, let $r \in Runs(S)$. Clearly $r \in r \restriction_{T_1} \otimes \cdots \otimes r \restriction_{T_n}$. Further, by repeated use of the first axiom of independence we obtain $\forall i \in [1, n], r \restriction_{T_i} \in Runs(T_i)$. To see that each $c_i \in C$ is prime consider that by definition each c_i does not contain any independent factors. □

Our decomposition results concern systems which are decomposable into *behaviourally strongly connected* (short: *bsc*) systems. In bsc systems every concurrent step has an 'observable link' with any further concurrently enabled transition.

[2] My thanks to Monika Maidl, who has helped to clarify a related question.

This feature will enable us to prove a key insight later on (Lemma 4.3.2), which will act as one of the pillars of the proof of the decomposition results.

Definition 4.3.6 (bsc-decomposition terminology). Let S be a system.

Let $r \in \mathit{Runs}(S)$, and $k, l \in [1, |r|]$. We say $w \in T_S^+$ is a *run at* r *that is causally dependent on the events* k *and* l, denoted by $w \in \mathit{depRuns}_S(r, k, l)$, iff

1. $r.w \in \mathit{Runs}(S)$, and
2. $\exists m \in [|r+1|, |r.w|].\ k <_{r.w} m \ \&\ l <_{r.w} m$.

S is *behaviourally strongly connected* (short: *bsc*) iff for all $r \in \mathit{csteps}(S)$ with $|r| \geq 1$ we have: $\forall t \in T_S.\ r.t \in \mathit{csteps}(S) \implies \exists k \in [1, |r|], w \in T_S^+.\ w \in \mathit{depRuns}_S(r.t, k, |r.t|)$.

A set $\mathcal{D}$ of sub-systems of S is a *decomposition of* S *into independent components that are behaviourally strongly connected* (short: *bsc-decomposition of* S) iff $\mathcal{D}$ is a decomposition of S, and for all $c \in \mathcal{D}$ c is a bsc system.

S is *decomposable into independent components of which each is behaviourally strongly connected* (short: *bsc-decomposable*) iff S has a bsc-decomposition.

It is clear that not every system is bsc-decomposable; for example system B of Figure 4.3 (page 119) is not. If a system is bsc-decomposable then its prime components are bsc. In more detail, we have:

Fact 4.3.3.

1. *Non-empty bsc systems are prime.*
2. *Let $\mathcal{C}$ be a class of bsc systems, and $(S, \mathcal{D})$ be a decomposed system such that for all $c \in \mathcal{D}$ c is a $\mathcal{C}$ system. Then the prime components of S are $\mathcal{C}$ systems.*
3. *The prime components of bsc-decomposable systems are bsc.*

Proof. (1) Let S be a non-empty bsc system. To the contrary suppose S is not prime. Then we can assume a decomposition $\mathcal{D}$ of S such that $|\mathcal{D}| > 1$ and for all $c \in \mathcal{D}$ c is non-empty. Select two distinct components c_1, $c_2 \in \mathcal{D}$, and two transitions $t_1 \in T_{c_1}$, $t_2 \in T_{c_2}$ such that $t_i \in \mathit{Runs}(c_i)$ for $i = 1$, and 2. By definition of decomposition we obtain t_1, $t_1 t_2 \in \mathit{csteps}(S)$. Then, since S is bsc there must be a run at $t_1 t_2$ that is causally dependent on the events corresponding to t_1 and t_2. But this contradicts that t_1 and t_2 belong to distinct independent factors of S.

(2) If S is empty $PComps(S) = \emptyset$, and (2) vacuously holds. If S is non-empty then $\mathcal{D} \backslash c^S_{empty}$ is still a decomposition, and by (1) and uniqueness of prime decomposition we must have $\mathcal{D} \backslash c^S_{empty} = PComps(S)$.

(3) is a consequence of (2) and the definition of bsc-decomposable. $\square$

Finally, we introduce terminology and observations, which will be employed later on.

Definition 4.3.7. Let $(S, \mathcal{D})$ be a decomposed system.

Given $c \in \mathcal{D}$ and an entity x of S, we use $x \uparrow c$ short for $x \uparrow (S_c \cup T_c)$.

Let $r \in T_S^*$, $c \in \mathcal{D}$, and $k \in [1, |r \uparrow c|]$. We say $k' \in [1, |r|]$ is the *kth c-event of r* iff k' is the kth event e in r satisfying $K(r[e]) = c$.

Proposition 4.3.4 (basic observations). *Let $(S, \mathcal{D})$ be a decomposed system.*

1. *Assume $c \in \mathcal{D}$. Let $r \in T_S^*$, $w \in T_c^*$, $k \in [1, |(r \uparrow c).w|]$, and k' be the kth c-event of $r.w$.*
   (a) $k \in [1, |(r \uparrow c)|] \implies k' \in [1, |r|]$.
   (b) $k \in [|r \uparrow c| + 1, |(r \uparrow c).w|] \implies k' \in [|r| + 1, |r.w|]$.
2. $\forall t, t' \in T_S.\ K(t) \neq K(t') \implies t\ I_S\ t'$.
3. *Let $r \in Runs(S)$. For all events $k, l \in [1, |r|]$ we have:*
   (a) $K(r[k]) \neq K(r[l]) \implies k\ co_r\ l$*, or equivalently*
   (b) $k\ dep_r\ l \implies K(r[k]) = K(r[l])$.
4. *Let $\mathcal{D} = \{c_1, \ldots, c_n\}$, $r_i \in T^*_{c_i}$ for $i \in [1, n]$, and $r \in r_1 \otimes \cdots \otimes r_n$. We have: $\forall i \in [1, n].\ r_i = r \uparrow c_i$.*

Proof. (1) is obvious. (2) is immediate from clause (1) of the definition of decomposition. (3) follows with (2) by induction on the length of r. To see (4) consider: by clause (1) of the definition of decomposition and the irreflexivity of independence we have $T_{c_i} \cap T_{c_j} = \emptyset$ for all distinct i, $j \in [1, n]$; then (4) is a consequence of Prop. 2.5.1. $\square$

Proposition 4.3.5 (inferring behaviour). *Let $(S, \mathcal{D})$ be a decomposed system, and $c \in \mathcal{D}$.*

1. $r \in Runs(S) \implies r \uparrow c \in Runs(c)$.
2. (a) $r \in Runs(c) \implies r \in Runs(S)\ \&\ Ks(r) \subseteq \{c\}$.

(b) $r \in \mathit{Runs}(S)$ & $(r\!\uparrow\! c).w \in \mathit{Runs}(c) \implies$
$r.w \in \mathit{Runs}(S)$ & $\mathit{Ks}(w) \subseteq \{c\}$.

3. *Let $r \in \mathit{Runs}(S)$. For all $k, l \in [1, |r\!\uparrow\! c|]$ and $k', l' \in [1, |r|]$ such that k' is the kth, and l' the lth, c-event of r, we have: $k <^{c}_{r\uparrow c} l \iff k' <^{S}_{r} l'$.*

Proof. We write (D2) short for "clause (2) of the definition of decomposition". (1) follows from (D2)'$\subseteq$' and Prop. 4.3.4(4). (2a) is immediate with (D2)'$\supseteq$' and the fact that for all $c' \in \mathcal{D}$, $\varepsilon \in \mathit{Runs}(c')$. To see (2b) consider both directions of (D2) and Prop. 4.3.4(4). (3) follows by induction on the length of r while considering Prop. 4.3.4(2) and the following fact: $\forall t, t' \in T_c.\ t\ I_c\ t' \iff t\ I_S\ t'$. The latter is immediate by the definition of sub-system. $\square$

Proposition 4.3.6 (on decomposed systems and csteps). *Let S be a system, and $\mathcal{D}$ be a decomposition of S; we also allow $\mathcal{D} = \mathit{PComps}(S)$.*

1. *Let $c \in \mathcal{D}$, and $r \in T_S^*$.*
   (a) $r \in \mathit{csteps}(c) \implies r \in \mathit{csteps}(S)$ & $\mathit{Ks}(r) \subseteq \{c\}$.
   (b) $r \in \mathit{csteps}(S) \implies r\!\uparrow\! c \in \mathit{csteps}(c)$.
   (c) *Let $r \in \mathit{csteps}(S)$, and $t \in T_S$.*
   $(r\!\uparrow\! c).t \in \mathit{csteps}(c) \iff r.t \in \mathit{csteps}(S)$ & $K(t) = c$.
2. *Let $r, r' \in \mathit{csteps}(S)$. $\mathit{Ks}(r) \cap \mathit{Ks}(r') = \emptyset \implies r.r' \in \mathit{csteps}(S)$.*
3. *Let $C \subseteq \mathcal{D}$ such that $\forall c \in C.\ c$ is non-empty. $\exists r \in \mathit{csteps}(S).\ \mathit{Ks}(r) = C$.*

Proof. (1a) follows by Prop. 4.3.5(2a) and (3), (1b) with Prop. 4.3.5(1) and (3). (1c)'$\Leftarrow$' is immediate with (1b), whereas '$\Rightarrow$' follows with Prop. 4.3.5(2b),(3) and Prop. 4.3.4(3a). (2) is a consequence of Prop. 4.3.1(2) and Prop 4.3.4(2). (3) is immediate with Prop. 4.3.1(3), and clause (1a) and (2) of this proposition. $\square$

4.3.2 Composition Results

We now show that hp and hhp bisimilarity are composable with respect to decompositions in the following sense: whenever we can exhibit a one-to-one correspondence between the components of two decomposed systems such that related components are hp (hhp) bisimilar then the two systems are hp (hhp) bisimilar.

Theorem 4.3.1 (composition result). *Let $(S_1, \mathcal{D}_1)$ and $(S_2, \mathcal{D}_2)$ be two decomposed systems.*

1. *If there exists a bijection $\beta : \mathcal{D}_1 \to \mathcal{D}_2$ such that $c_1 \sim_{hp} \beta(c_1)$ for each $c_1 \in \mathcal{D}_1$ then we have $S_1 \sim_{hp} S_2$.*

2. *If there exists a bijection $\beta : \mathcal{D}_1 \to \mathcal{D}_2$ such that $c_1 \sim_{hhp} \beta(c_1)$ for each $c_1 \in \mathcal{D}_1$ then we have $S_1 \sim_{hhp} S_2$.*

Proof. Let $(S_1, \mathcal{D}_1)$ and $(S_2, \mathcal{D}_2)$ be two decomposed systems.

(1.) Assume we are given a bijection $\beta : \mathcal{D}_1 \to \mathcal{D}_2$, say $\beta = \{c_1, \ldots, c_n\}$, and a family $\{\mathcal{H}_{c_i}\}_{i=1}^{n}$ such that for $i \in [1, n]$ $\mathcal{H}_{c_i}$ is a hp bisimulation relating $proj_1(c_i)$ and $proj_2(c_i)$. We define $\mathcal{H} = \bigcup \{r_{c_1} \otimes \cdots \otimes r_{c_n} \mid r_{c_i} \in \mathcal{H}_{c_i} \text{ for } i \in [1, n]\}$. It is straightforward to check that $\mathcal{H}$ is a hp bisimulation relating S_1 and S_2. To see that clause (1) of Def. 2.2.2 holds assume $r \equiv (r_1, r_2) \in \mathcal{H}$, and consider: by clause (2)'$\supseteq$' of the definition of decomposition $r_1 \in Runs(S_1)$ and $r_2 \in Runs(S_2)$; since for all $k \in [1, |r|]$, $i \in [1, n]$ we have $K(r_1[k]) = proj_1(c_i) \iff K(r_2[k]) = proj_2(c_i)$, with Prop. 4.3.4(3a), and Prop. 4.3.5(3) combined with Prop. 4.3.4(4) we infer r_1 and r_2 must be synchronous. Clause (2) follows since clearly for all $i \in [1, n]$ $\varepsilon \in \mathcal{H}_{c_i}$. (3) and (4) are easy to verify with Prop. 4.3.5(1) and Prop. 4.3.4(4).

(2.) Let $r \in \mathcal{H}$ and $t \equiv (t_1, t_2) \in BEn_S(r)$. For $i \in [1, n]$ set $r_{c_i} \in \mathcal{H}_{c_i}$ such that $r \in r_{c_1} \otimes \cdots \otimes r_{c_n}$. Further, set j such that $K(t_1) = proj_1(c_j)$ and $K(t_2) = proj_2(c_j)$. It is easy to see: we also have $t \in BEn_c(r_{c_j})$. But then we can easily infer: if for all $i \in [1, n]$ $\mathcal{H}_{c_i}$ is hereditary then $\mathcal{H}$ will also be hereditary. □

4.3.3 Decomposition Results

On the other hand, for the class of bsc-decomposable systems, hp and hhp bisimilarity are decomposable with respect to the set of prime components: whenever two bsc-decomposable systems are hp (hhp) bisimilar then there is a one-to-one correspondence between their prime components such that related components are hp (hhp) bisimilar. The proof of this statement is more involved than that of the composition result. In addition to two crucial insights about hp (hhp) bisimilarity we will require the combinatorial argument of Hall's Marriage Theorem[3] (e.g. see [Tru91]).

First of all, building on our concepts of concurrent step and mc step, we introduce the notion of *concurrent step that is maximal and exclusive with respect to a set of components* (short: *mec step*). Such concurrent steps are defined for decomposed systems as follows.

[3] My thanks to Walter Vogler for correcting an earlier version of a similar proof, and pointing out to me that this theorem has to be applied.

Definition 4.3.8 (mec steps). Let $S = (S, \mathcal{D})$ be a decomposed system, and $C \subseteq \mathcal{D}$. A concurrent step $r \in csteps(S)$ is *maximal and exclusive w.r.t.* C iff $\forall c \in C.\ r{\uparrow}c \in mcsteps(c)$ and $Ks(r) \subseteq C$. We denote the *set of concurrent steps of* S *that are maximal and exclusive w.r.t.* C by $mecsteps(S, C)$.

Proposition 4.3.7 (facts about mec steps). *Let* $S = (S, \mathcal{D})$ *be a decomposed system.*

1. *Let* $c \in \mathcal{D}$*, and* $r \in mcsteps(c)$. $r \in mecsteps(S, \{c\})$.
2. *Let* $C, C' \subseteq \mathcal{D}$ *with* $C \cap C' = \emptyset$*,* $r \in mecsteps(S, C)$*, and* $r' \in mecsteps(S, C')$. $r.r' \in mecsteps(S, C \cup C')$.
3. $\forall C \subseteq \mathcal{D}.\ \exists r.\ r \in mecsteps(S, C)$.
4. *Let* $C \subseteq \mathcal{D}$*,* $r \in mecsteps(S, C)$*, and* $t \in T_S$.
   $r[t\rangle_S \;\&\; K(t) \notin C \iff r.t \in csteps(S)$.
5. *Let* $C \subseteq \mathcal{D}$*, and* $r \in mecsteps(S, C)$.
   $(\forall c \in C.\ c$ *is non-empty*$) \implies Ks(r) = C$.

Proof. (1) follows with Prop. 4.3.6(1a), (2) with Prop. 4.3.6(2), and (3) with (1), (2), and the fact that for any system S $\exists r.\ r \in mcsteps(S)$; (4)'$\Leftarrow$' is immediate with Prop. 4.3.6(1b) and the definition of mc step, and the '$\Rightarrow$'-direction with Prop. 4.3.4(3a). (5) follows from Prop. 4.3.1(4). □

Let $(S_1, \mathcal{D}_1)$, $(S_2, \mathcal{D}_2)$ be two decomposed systems such that S_1 and S_2 are hp (hhp) bisimilar. With our concept of mec steps it is easy to identify a situation which will allow us to infer that two components $c_1 \in \mathcal{D}_1$, $c_2 \in \mathcal{D}_2$ are hp (hhp) bisimilar: whenever there is $r \in \sim_{(h)hp}$ such that for $i = 1$, and 2, $proj_i(r)$ is a mec step w.r.t. $\mathcal{D}_i \backslash c_i$, then we can extract a hp (hhp) bisimulation that relates c_1 and c_2 from any hp (hhp) bisimulation containing r. This is so because: (1) the full behaviour of c_1 and c_2 has still to be matched at r, and (2) the causal dependencies will force that behaviour of c_1 has to be matched against c_2, and vice versa. Formally, we have:

Lemma 4.3.1. *Let* $(S_1, \mathcal{D}_1)$*,* $(S_2, \mathcal{D}_2)$ *be two decomposed systems,* $c_1 \in \mathcal{D}_1$*, and* $c_2 \in \mathcal{D}_2$.

1. *If there exists* $r \in \sim_{hp}$ *such that* $proj_i(r) \in mecsteps(S_i, \mathcal{D}_i \backslash c_i)$ *for* $i = 1$*, and* 2 *then we have* $c_1 \sim_{hp} c_2$.

2. *If there exists* $r \in \sim_{hhp}$ *such that* $proj_i(r) \in mecsteps(S_i, \mathcal{D}_i \backslash c_i)$ *for* $i = 1$, *and* 2 *then we have* $c_1 \sim_{hhp} c_2$.

Proof. (1.) Let $(S_1, \mathcal{D}_1)$, $(S_2, \mathcal{D}_2)$, c_1, and c_2 be given as above. Assume $r \equiv (r_1, r_2) \in \sim_{hp}$ such that $r_i \in mecsteps(S_i, \mathcal{D}_i \backslash c_i)$ for $i = 1$, and 2. Consider the following statement, say (S):

For all r' such that $r.r' \in \sim_{hp}$ we have:

$$(*)\ \forall k \in [1, |r'|].\ r'_1[k] \in T_{c_1} \iff r'_2[k] \in T_{c_2}.$$

Given a hp bisimulation $\mathcal{H}$ for S_1 and S_2 such that $r \in \mathcal{H}$, define $\mathcal{H}_c = \{r'' \uparrow (T_{c_1} \times T_{c_2}) \mid r'' \in \mathcal{H}\ \&\ r'' \equiv r.r' \text{ for some } r'\}$. Considering (S) it is easy to check that $\mathcal{H}_c$ is a hp bisimulation relating c_1 and c_2: clause (1) of Def. 2.2.2 follows with Prop. 4.3.5(1) and (3); clause (2) holds because $r \in \mathcal{H}$ and $Ks(r_i) \subseteq \mathcal{D}_i \backslash c_i$ for $i \in \{1, 2\}$ by definition of mec step; (3) and (4) follow with Prop. 4.3.5(2b).

By induction on the length of r' we shall now prove that (S) indeed holds. Base case $r' \equiv \varepsilon$: there is nothing to prove. Inductive case $r' \equiv r''.(t_1, t_2)$: set $r''_i = proj_i(r'')$ for $i \in \{1, 2\}$. By prefix-closure of $\sim_{hp}$ and the induction hypothesis we can assume that $(*)$ holds for r''. Thus, we only need to show: $t_1 \in T_{c_1} \Leftrightarrow t_2 \in T_{c_2}$.

Suppose $t_1 \in T_{c_1}$. We prove $t_2 \in T_{c_2}$ by case analysis. First, assume there is $k \in [1, |r''|]$ such that $k <_{r'_1} |r'|$. By Prop. 4.3.4(3b) this implies $K(r'_1[k]) = c_1 (= K(r'_1[|r'|]))$. Since we can assume that $(*)$ holds for r'', we furthermore obtain $K(r'_2[k]) = c_2$. On the other hand, considering that $\sim_{hp}$ is partial order preserving, we infer $k <_{r'_2} |r'|$. But then, Prop. 4.3.4(3b) immediately implies $K(r'_2[|r'|]) = c_2$, or $t_2 \in T_{c_2}$ as required.

If there is no k as above then it is easy to derive that $r.(t_1, t_2) \in SRuns(S_1, S_2)$. Together with $r_1 \in mecsteps(S_1, \mathcal{D}_1 \backslash c_1)$ and $t_1 \in T_{c_1}$ this means that we can apply Prop. 4.3.7(4)'$\Rightarrow$' to obtain $r_1.t_1 \in csteps(S_1)$, which by Prop. 4.3.2(1) implies $r_2.t_2 \in csteps(S_2)$. But then by $r_2 \in mecsteps(S_1, \mathcal{D}_2 \backslash c_2)$ and Prop. 4.3.7(4)'$\Leftarrow$' it is immediate that $t_2 \in T_{c_2}$.

The opposite direction follows from the symmetric argument.

(2.) Let $r_c \in \mathcal{H}_c$ and $r'' \in \mathcal{H}$ such that $r'' \equiv r.r'$ for some r', and $r_c = r'' \uparrow (T_{c_1} \times T_{c_2})$. With (S) and Prop. 4.3.4(2) it is easy to see: $\forall t \in T_c.\ t \in BEn_c(r_c) \implies t \in BEn_S(r'')$. But then we can easily infer: if $\mathcal{H}$ is hereditary then $\mathcal{H}_c$ will also be hereditary. □

We now come to our second crucial insight. Let S be a bsc-decomposable system, and c be a prime component of S. By Fact 4.3.3(3) c is bsc, and thus in c every concurrent step has an 'observable link' with any further concurrently enabled transition. Naturally, we have: (1) The 'observable links' of each c can also be computed in the context of S. (2) In S 'observable links' always connect events that belong to the same prime component. Then, considering that concurrent steps of S exhaustively fall into concurrent steps of prime components, we obtain: *in hp bisimilarity on bsc-decomposable systems the matching of concurrent steps respects prime decompositions.* Otherwise, the observable links could not be matched in a partial order preserving fashion.

Proposition 4.3.8. *Let $(S, \mathcal{D})$ be a decomposed system, and $r \in Runs(S)$.*

1. *Let $c \in \mathcal{D}$, $k, l \in [1, |r{\uparrow}c|]$, and $w \in depRuns_c(r{\uparrow}c, k, l)$. Then we have: $w \in depRuns_S(r, k', l')$, where k' denotes the kth, and l' the lth, c-event of r.*

2. *Let $k, l \in [1, |r|]$. $K(r[k]) \neq K(r[l]) \implies \nexists w.\ w \in depRuns_S(r, k, l)$.*

Proof. (1) is immediate with Prop. 4.3.5(2b),(3), Prop. 4.3.4(1), and the definition of *depRuns*. (2) follows from Prop. 4.3.4(3b) and the definition of *depRuns*. ☐

Lemma 4.3.2. *Let S_1 and S_2 be two bsc-decomposable systems. For all $r \equiv (r_1, r_2) \in \sim_{hp}$ such that $r_i \in csteps(S_i)$ for $i = 1$, or 2 equivalently (Prop. 4.3.2(1)), we have:*

$$(*) \quad \forall k, l \in [1, |r|].\ K(r_1[k]) = K(r_1[l]) \iff K(r_2[k]) = K(r_2[l]).$$

Proof. Let S_1, S_2, and $r \equiv (r_1, r_2)$ be given as above. We prove the lemma by induction on the length of r. Base case $r \equiv \varepsilon$: there is nothing to prove. Inductive case $r \equiv r'.t$: for $i \in \{1, 2\}$ set $r'_i = proj_i(r')$, and $t_i = proj_i(t)$. Since $r'_i \in csteps(S_i)$ for $i = 1$, and 2 (Prop. 4.3.1(1)), by prefix-closure of $\sim_{hp}$ and the induction hypothesis we can assume that $(*)$ holds for r'. Thus, we only have to prove: $\forall k \in [1, |r'|].\ K(t_1) = K(r_1[k]) \iff K(t_2) = K(r_2[k])$. We establish this by *reductio ad absurdum.*

Assume there is $k \in [1, |r'|]$ such that (a) $K(t_1) = K(r_1[k])$ but (b) $K(t_2) \neq K(r_2[k])$. Set $c_1 = K(t_1)$, and $r'_{c_1} = r'_1{\uparrow}c_1$. Clearly, $r_1{\uparrow}c_1 = r'_{c_1}.t_1$. Further, it is easy to see: (c) $|r|$ is the $|r'_{c_1}.t_1|$th c_1-event in r_1. (d) If for $l \in [1, |r'_{c_1}|]$ m is the lth c_1-event in r_1 then $m \in [1, |r'|]$ and $K(r_1[m]) = c_1$.

By Prop. 4.3.6(1b) we obtain r'_{c_1}, $r'_{c_1}.t_1 \in csteps(c_1)$. Then, by Fact 4.3.3(3) and the definition of bsc there exist $w_1 \in T^+_{c_1}$, $l \in [1, |r'_{c_1}|]$ such that $w_1 \in depRuns_{c_1}(r'_{c_1}.t_1, l, |r'_{c_1}.t_1|)$. Employing Prop. 4.3.8(1), (c), and (d), we infer $w_1 \in depRuns_{S_1}(r_1, m, |r|)$, where m satisfies (e) $m \in [1, |r'|]$ & $K(r_1[m]) = c_1$. Clearly, there must be a match for w_1 at r, that is we can assume w with $r.w \in \sim_{hp}$

and $proj_1(w) = w_1$. Set $w_2 = proj_2(w)$. Since $\sim_{hp}$ is partial order preserving we must have $w_2 \in depRuns_{S_2}(r_2, m, |r|)$. But considering Prop. 4.3.8(2) this is not possible: since we can assume that $(*)$ holds for r', by (e), (a), and $k \in [1, |r'|]$ we obtain $K(r_2[k]) = K(r_2[m])$, and furthermore with (b), $K(r_2[m]) \neq K(t_2)(= K(r_2[|r|])$.

The other direction follows from the symmetric argument. □

Based on Lemma 4.3.2 we could now prove that more generally we have: for bsc-decomposable systems, *every* match $r \in \sim_{hp}$ respects prime decompositions. However, since this statement only concerns the matching *within* and not *across* linearizations it does not bring us any closer to our decomposition results. Instead, we shall derive the following two corollaries.

On the one hand, we infer that in hp bisimilarity on bsc-decomposable systems mec steps are matched against mec steps. This is important in view of Lemma 4.3.1.

Corollary 4.3.1. *Let S_1, S_2 be two bsc-decomposable systems, $r \equiv (r_1, r_2) \in \sim_{hp}$, $i \in \{1, 2\}$, and $C_i \subseteq PComps(S_i)$. We have:*

$$r_i \in mecsteps(S_i, C_i) \implies r_{\bar{i}} \in mecsteps(S_{\bar{i}}, Ks(r_{\bar{i}})).$$

Proof. Let S_1, S_2, $r \equiv (r_1, r_2)$, i, and C_i be given as above. Assume (A) $r_i \in mecsteps(S_i, C_i)$. We need to show: (1) $r_{\bar{i}} \in csteps(S_{\bar{i}})$, (2) $Ks(r_{\bar{i}}) \subseteq Ks(r_{\bar{i}})$, and (3) $\forall c_{\bar{i}} \in Ks(r_{\bar{i}}).\ r_{\bar{i}}{\uparrow}c_{\bar{i}} \in mcsteps(c_{\bar{i}})$. (1) is given by (A) and Prop. 4.3.2(1), and (2) is trivial.

For (3) let $c_{\bar{i}} \in Ks(r_{\bar{i}})$. Clearly, there must be $k \in [1, |r|]$ such that $K(r_{\bar{i}}[k]) = c_{\bar{i}}$. Set $c_i = K(r_i[k])$. To the contrary, suppose (B) $r_{\bar{i}}{\uparrow}c_{\bar{i}} \notin mcsteps(c_{\bar{i}})$. Considering $r_{\bar{i}} \in csteps(S_{\bar{i}})$ and Prop. 4.3.6(1b) we have $r_{\bar{i}}{\uparrow}c_{\bar{i}} \in csteps(c_{\bar{i}})$, and thus by (B) and the definition of mc step we can assume $t_{\bar{i}}$ such that $(r_{\bar{i}}{\uparrow}c_{\bar{i}}).t_{\bar{i}} \in csteps(c_{\bar{i}})$. By Prop. 4.3.6(1c)'$\Rightarrow$' we obtain (a) $r_{\bar{i}}.t_{\bar{i}} \in csteps(S_{\bar{i}})$ and (b) $K(t_{\bar{i}}) = c_{\bar{i}}$. There must be a match for $t_{\bar{i}}$ at r, that is we can assume t_i such that $r.(t_1, t_2) \in \sim_{hp}$. By (a) and Prop. 4.3.2(1) $r_i.t_i \in csteps(S_i)$, and by (b) and Lemma 4.3.2 we infer $K(t_i) = c_i$. Then by Prop. 4.3.6(1c)'$\Leftarrow$' $(r_i{\uparrow}c_i).t_i \in csteps(c_i)$. But this is a contradiction to our assumption $r_i \in mecsteps(S_i, C_i)$, which, since obviously $c_i \in Ks(r_i)$, entails $r_i{\uparrow}c_i \in mcsteps(c_i)$. □

On the other hand, Lemma 4.3.2 implies that in hp bisimilarity on bsc-decomposable systems the matching of concurrent steps respects the number of prime components involved. Furthermore, we obtain that whenever two bsc-decomposable systems are hp bisimilar then they have the same number of prime components.

Corollary 4.3.2. *Let S_1 and S_2 be two bsc-decomposable systems.*

1. *For all $r \equiv (r_1, r_2) \in \sim_{hp}$ such that $r_i \in csteps(S_i)$ for $i = 1$, or 2 equivalently (Prop. 4.3.2(1)), we have: $|Ks(r_1)| = |Ks(r_2)|$.*

2. *If $S_1 \sim_{hp} S_2$ then $|PComps(S_1)| = |PComps(S_2)|$.*

Proof. (1) is a direct consequence of Lemma 4.3.2. (2) is easy with (1) and Prop. 4.3.6(3). □

Together with the combinatorial argument of Hall's Marriage Theorem these corollaries make it possible that we can employ Lemma 4.3.1 to obtain our decomposition result. Finally, we prove:

Theorem 4.3.2 (decomposition result). *Let S_1, S_2 be two bsc-decomposable systems.*

1. *If $S_1 \sim_{hp} S_2$ then there exists a bijection $\beta : PComps(S_1) \to PComps(S_2)$ between the prime components of S_1 and those of S_2 such that $c_1 \sim_{hp} \beta(c_1)$ for each $c_1 \in PComps(S_1)$.*

2. *If $S_1 \sim_{hhp} S_2$ then there exists a bijection $\beta : PComps(S_1) \to PComps(S_2)$ between the prime components of S_1 and those of S_2 such that $c_1 \sim_{hhp} \beta(c_1)$ for each $c_1 \in PComps(S_1)$.*

Proof. Let S_1 and S_2 be two bsc-decomposable systems.

(1.) Assume $S_1 \sim_{hp} S_2$. We shall prove that a bijection β exists as required. By Corollary 4.3.2(2) it is clear that (A) $|PComps(S_1)| = |PComps(S_2)|$, and it only remains to show that an injective map can be found. For each $c_1 \in PComps(S_1)$ let $C_{2_{c_1}}$ be the set of prime components of S_2 which are hp bisimilar to c_1. By Hall's Marriage Theorem (see e.g. [Tru91]) the required injection exists if and only if the following condition is fulfilled:

$$(*) \quad \forall C_1 \subseteq PComps(S_1).\, |\bigcup_{c_1 \in C_1} C_{2_{c_1}}| \geq |C_1|.$$

Choose an arbitrary subset C_1 of $PComps(S_1)$. Let $\bar{C}_1 = PComps(S_1)\backslash C_1$, and consider a run $r_1 \in mecsteps(S_1, \bar{C}_1)$ (this is possible by Prop. 4.3.7(3)); by Prop. 4.3.7(5) we have (B) $Ks(r_1) = \bar{C}_1$. Clearly, there must be $r \in \sim_{hp}$ with $proj_1(r) = r_1$; set $r_2 = proj_2(r)$, $\bar{C}_2 = Ks(r_2)$, and $C_2 = PComps(S_2)\backslash \bar{C}_2$. By Corollary 4.3.1 we obtain (C) $r_2 \in mecsteps(S_2, \bar{C}_2)$. On the other hand, (B) and Corollary 4.3.2(1) give us $|\bar{C}_1| = |\bar{C}_2|$, and considering (A) we further gain (D) $|C_1| = |C_2|$. Next we show that for each remaining component $c_2 \in C_2$ there

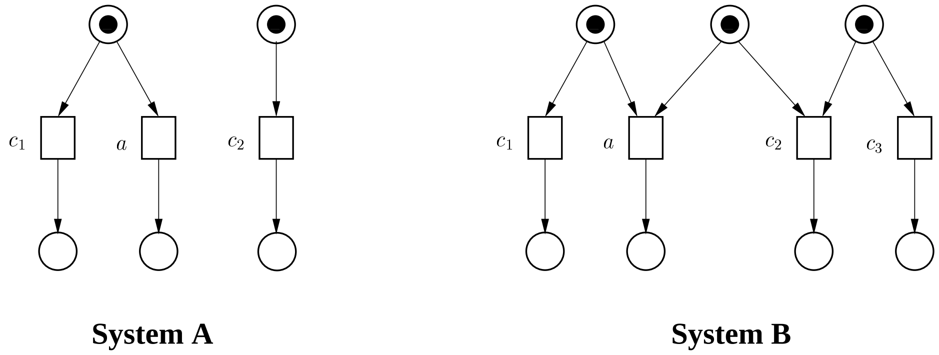


Figure 4.3: hp bisimilarity is *not* decomposable with respect to the set of prime components in general

is a component $c_1 \in C_1$ such that $c_1 \sim_{hp} c_2$. With (D) this will immediately establish $|\bigcup_{c_1 \in C_1} C_{2_{c_1}}| \geq |C_1|$, and thereby $(*)$.

Assume C_2 is non-empty, and choose any $c_2 \in C_2$. Further, consider a run $r_2' \in \mathit{mecsteps}(S_2, C_2 \backslash c_2)$ (again this is possible by Prop. 4.3.7(3)). By (C) and Prop. 4.3.7(2) we have $r_2.r_2' \in \mathit{mecsteps}(S_2, \mathit{PComps}(S_2) \backslash c_2)$, and with Prop. 4.3.7(5) we obtain (E) $\mathit{Ks}(r_2.r_2') = \mathit{PComps}(S_2) \backslash c_2$. Clearly, there must be r' such that $r.r' \in \sim_{hp}$ and $\mathit{proj}_2(r') = r_2'$; set $r_1' = \mathit{proj}_1(r')$. Corollary 4.3.2(1) gives us $|\mathit{Ks}(r_1.r_1')| = |\mathit{Ks}(r_2.r_2')|$, and by (E), (A), and (B) this implies $\mathit{Ks}(r_1.r_1') = \mathit{PComps}(S_1) \backslash c_1$ for some $c_1 \in C_1$. Then, by Corollary 4.3.1 we obtain $r_1.r_1' \in \mathit{mecsteps}(S_1, \mathit{PComps}(S_1) \backslash c_1)$. But altogether this means we can apply Lemma 4.3.1(1) to infer $c_1 \sim_{hp} c_2$. Thus, c_1 provides a component exactly as required.

(2.) This can be proved analogously to (1); simply employ Lemma 4.3.1(2) instead of (1). But there is also a simpler proof: the required bijection can be obtained by employing backtracking instead of the argument of Hall's Marriage Theorem. $\square$

Is it possible to strengthen this decomposition result? Clearly, hp and hhp bisimilarity are not decomposable with respect to arbitrary decompositions, but maybe they are with respect to the set of prime components, for all systems? I conjecture that this is indeed the case for hhp bisimilarity. The crucial step would be to prove an analogue of Lemma 4.3.2, which should be possible with the help of backtracking. It is certain that in general hp bisimilarity is *not* decomposable with respect to the set of prime components: Figure 4.3 shows an example of two hp bisimilar systems such that a bijection between their prime components can clearly not be found. Note that the two systems are *not* hhp bisimilar.

One could hold the view that truly-concurrent bisimilarities should respect independent components in the sense of being decomposable with respect to the set of prime components. Under this aspect the negative result for hp bisimilarity underlines the non truly-concurrent character of the notion. Thus, provided we indeed obtain a positive result for hhp bisimilarity, we can put forward a new piece of evidence for our thesis of Section 1.2.1: hhp bisimilarity is a notion for true-concurrency whereas hp bisimilarity only captures causality.

4.3.4 Consequences

What are the gains of our composition and decomposition results? First of all, they provide us with the following proof technique:

Definition 4.3.9 (parallel compositions of $\mathcal{C}$ systems). Let $\mathcal{C}$ be a system class. A system S is a *parallel composition of* $\mathcal{C}$ *systems* iff S has a decomposition $\mathcal{D}$ such that for all $c \in \mathcal{D}$ c is a $\mathcal{C}$ system.

Corollary 4.3.3 (proof technique). *Let $\mathcal{C}$ be a class of bsc systems.*

1. *If hp and hhp bisimilarity coincide for the class of $\mathcal{C}$ systems then they also coincide for the class of parallel compositions of $\mathcal{C}$ systems.*

2. *In addition, let the systems of $\mathcal{C}$ be finite-state. If hhp bisimilarity is decidable for the class of $\mathcal{C}$ systems then it is also decidable for the class of parallel compositions of $\mathcal{C}$ systems.*

Proof. Let $\mathcal{C}$ be a class of bsc systems, and let S_1, S_2 be parallel compositions of $\mathcal{C}$ systems. Note that by Fact 4.3.3(2) the prime components of S_1 and S_2 are contained in $\mathcal{C}$.

(1.) Assume hp and hhp bisimilarity coincide for the class of $\mathcal{C}$ systems, and let $S_1 \sim_{hp} S_2$. By Theorem 4.3.2(1) we obtain a bijection between $PComps(S_1)$ and $PComps(S_2)$ such that two related components are hp, and hence, hhp bisimilar. Either $PComps(S_1)$ and $PComps(S_2)$ both are empty sets, or both are non-empty. In the first case $S_1 \sim_{hhp} S_2$ is trivial, in the second case this follows from Theorem 4.3.1(2).

(2.) Assume the systems of $\mathcal{C}$ are finite-state, and that hhp bisimilarity is decidable for $\mathcal{C}$. We can decide whether $S_1 \sim_{hhp} S_2$ as follows. First, compute the set of prime components of S_1 and S_2; this is possible by Fact 4.3.2. Then, check by exhaustive search whether there is a bijection between $PComps(S_1)$ and $PComps(S_2)$ such that two related components are hhp bisimilar. Note that for $PComps(S_1) = \emptyset = PComps(S_2)$ this method is sound because then

$S_1 \sim_{hhp} S_2$ is trivially given. Otherwise soundness follows from Theorem 4.3.1(2). By Theorem 4.3.2(2) the method is complete. □

Hence, whenever we investigate the two problems of hhp bisimilarity for a class of bsc-decomposable systems we can restrict our attention to the class of the bsc factors. Besides, we can apply the method to extend one of our previous results:

Theorem 4.3.3. *hp and hhp bisimilarity coincide for parallel compositions of sequential systems.*

Proof. Clearly, sequential systems are bsc, and so the result follows from Theorem 4.2.1(1) and Corollary 4.3.3(1). □

Note that the class of parallel compositions of sequential systems provides all of the behavioural situations which we have identified to be essential for the non-coincidence of hp and hhp bisimilarity in the previous section. We can now add: it is not sufficient if concurrency comes as the juxtaposing of sequential parts, but we need the sequential parts to interact, or communicate, with each other. Behaviourally this means: concurrency has to be mixed with conflict and/or causality.

We will attend to this in more detail in the next section, where we shall investigate behavioural situations that witness synchronization. There, our composition and decomposition insights will once more provide the key to a coincidence result; this time they will be employed in an inductive argument. A third application will follow in Section 5.3.2: the composition and decomposition insights will give soundness and completeness of a tableau system, and thereby help to establish a coincidence and decidability result.

Results for Structural Subclasses. To complete the picture we record that the coincidence result of Theorem 4.3.3 can be carried over to two structural subclasses: *S-systems* are parallel compositions of sequential systems, and this is also true for *systems with transitive dependence relation*, the counterpart of systems with transitive independence (cf. Section 3.5.3). In each case, this is an easy consequence of the definition. Thus, we obtain:

Corollary 4.3.4.

1. *hp and hhp bisimilarity coincide for S-systems.*
2. *hp and hhp bisimilarity coincide for systems with transitive dependence relation.*

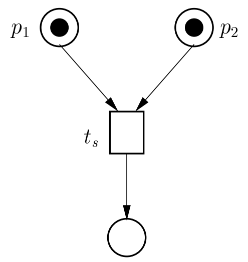


Figure 4.4: t_s is a synchronization at $\{p_1, p_2\}$

4.4 Synchronization Witness Situations

In the previous section we have established that to keep hp and hhp bisimilarity distinct concurrency has to be mixed with conflict and/or causality. Since such interplay can ultimately be put down to the interaction of sequential components it seems promising to analyse mixtures of concurrency with conflict and/or causality which manifest natural forms of structural interaction. In this section, we examine situations which witness synchronization: we identify three '*synchronization witness*' (short: *SW*) *situations*, and investigate their importance to the coincidence problem. On the one hand, we find: taken by themselves all of the SW situations are significant but not essential to keep hp and hhp bisimilarity distinct. On the other hand, we show: in their entirety they constitute a necessary condition to distinguish the two bisimilarities on *bounded-degree systems*. The proof of the latter fact makes use of a characterization of *SW-free systems* in terms of decomposition properties, which allows us to employ the composition and decomposition results of the previous section. We also conclude that hp and hhp bisimilarity coincide for bounded-degree *communication-free net systems*.

4.4.1 Definition

We define our synchronization witness situations with net systems as the underlying structural model in mind. In net theory it is natural to understand synchronization as the firing of a transition that has several preplaces, thereby uniting several concurrent 'components' (see Figure 4.4).

Definition 4.4.1 (synchronization). Let $\mathcal{N}$ be a net system, $M \in Reach(\mathcal{N})$, and $t \in T_N$. We say t is a *synchronization at* M iff $M[t\rangle$ and $|{}^{\bullet}t| > 1$.

We identify three basic ways in which such synchronization can manifest itself behaviourally (illustrated in Figure 4.5):

1. There is a state with two concurrently enabled transitions t_1 and t_2 such that a third transition t_s can happen causally dependent on t_1 and t_2.

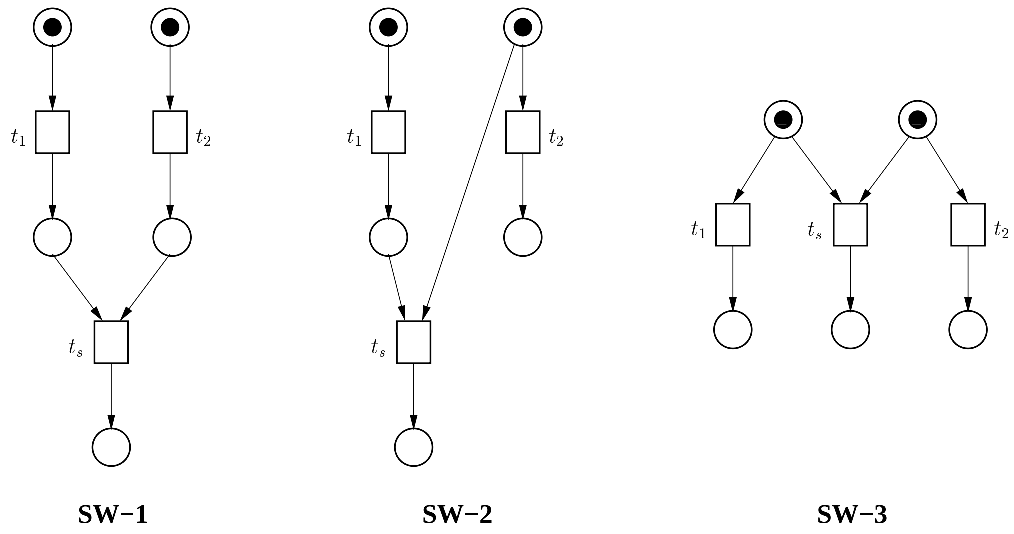


Figure 4.5: Illustration of the three SW situations

2. There is a state with two concurrently enabled transitions t_1 and t_2 such that t_2 is in conflict with a third transition t_s that can occur causally dependent on t_1.

3. There is a state with two concurrently enabled transitions t_1 and t_2 such that both of them are in conflict with a third enabled transition t_s.

With Def. 2.1.7, Def. 2.1.8, and safeness it is easy to check that these situations are sound with respect to synchronization in the sense of Def. 4.4.1: in any of the three scenarios we can infer that t_s must occur as synchronization at the structural level. Formally, we define the following three synchronization witness situations:

Definition 4.4.2 (synchronization witness situations). Let S be a system, $s \in \mathit{Reach}(S)$, $t_s \in T_S$, and $t_1, t_2 \in T_S$ with $t_1 \; co_s \; t_2$.

The tuple (s, t_1, t_2, t_s) is a *synchronization witness situation of type 1* (short: *SW-1 situation*) (at s) iff we have $\{t_1, t_2\} <_s t_s$.

The tuple (s, t_1, t_2, t_s) is a *synchronization witness situation of type 2* (short: *SW-2 situation*) (at s) iff we have $t_1 <_s t_s$ & $t_2 \#_{s'} t_s$, where s' is such that $s \xrightarrow{t_1} s'$.

The tuple (s, t_1, t_2, t_s) is a *synchronization witness situation of type 3* (short: *SW-3 situation*) (at s) iff we have $t_1 \#_s t_s$ & $t_2 \#_s t_s$.

We say there is a *synchronization witness situation* (short: *SW situation*) at s iff there is a SW-1, SW-2, or SW-3 situation at s.

In the following two sections, we shall consider systems with restricted synchronization. In preparation, we define:

Definition 4.4.3. Let X, Y range over $\{1, 2, 3\}$. A system S is *SW-X free* iff for all $s \in \mathit{Reach}(S)$ there is no SW-X situation at s. S is *SW-$\{X, Y\}$ free* iff for all $s \in \mathit{Reach}(S)$ there is neither an SW-X nor an SW-Y situation at s. Finally, S is *SW-free* iff for all $s \in \mathit{Reach}(S)$ there is no SW situation at s.

There is a well-known net class, which provides a structural characterization of SW-free systems: the *communication-free nets* of [Hir94] and [Esp97b].

Definition 4.4.4. A net N is *communication-free* (short: *comm-free*) iff we have: $\forall t \in T_N.\ |{}^\bullet t| = 1$. A net system $\mathcal{N}$ is *comm-free* iff N is comm-free.

Proposition 4.4.1. *Every comm-free net system is SW-free.*

Proof. Clearly, in comm-free net systems there is no synchronization in the sense of Def. 4.4.1. Then, the proposition follows from the soundness of the SW situations with respect to synchronization. □

4.4.2 The SW-X Situations and the Coincidence Problem

A look at the counter-examples shows that all of the three SW situations are significant for keeping hp and hhp bisimilarity distinct:

Counter-example 1 (Figure 1.7). Consider system B. The tuples (s', a'_2, b'_2, c') and (s', b'_2, a'_2, d') are both SW-2 situations. They correspond to the two MNH situations employed in the system. The tuple (s', a'_2, b'_2, a'_1) is a SW-3 situation. Together with further SW-3 situations, it originates from the counter-example's frame situation. There are corresponding SW-2 and SW-3 situations in system A.

Counter-example 2 (Figure 1.8). There are several SW-3 situations in both systems, e.g. the tuple (s, a_1, b_1, a_2). They all stem from the frame situation employed in the two systems. The counter-example's MNH situations do not give rise to any SW situation.

Counter-example 3 (Figure 4.1, 4.2). We regard system S. Set $s = \{p_1, p_2\}$, and $s' = \{p_2, p_3\}$. The tuples (s, e_1, a, i) and (s', e_1, f_2, g_1) are SW-1 situations. There are further SW-1 situations in S, and corresponding ones in system S'; all are due to MNH situations employed in the systems. The counter-example's frame does not involve any SW situations.

However, the counter-examples also demonstrate that none of the three SW situations is essential for non-coincidence, and even more they show that we can do without SW-1 *and* SW-2, and also without SW-2 *and* SW-3.

Theorem 4.4.1.

1. *hp and hhp bisimilarity do not coincide for SW-X free systems, where X ranges over* $\{1,2,3\}$, *and even more:*

2. *hp and hhp bisimilarity do not coincide for* SW-$\{1,2\}$ *free systems, nor for* SW-$\{2,3\}$ *free systems.*

Proof. Simply observe: counter-example 2 does not contain SW-1 nor SW-2; counter-example 3 manages without SW-2 and SW-3. □

It is probably also true that hp and hhp bisimilarity do not coincide for SW-$\{1,3\}$ free systems: it seems possible to construct a counter-example without SW-1 and SW-3 by employing the framework of counter-example 3 together with the MNH scenario of counter-example 1.

On the other hand, all of the three counter-examples employ at least one type of SW situation. So we ask: are the SW situations necessary to distinguish hp and hhp bisimilarity when taken together? In the following we prove that this is indeed the case for *bounded-degree* systems.

4.4.3 hp and hhp Bisimilarity Coincide for Bounded-degree SW-free Systems

First of all, we will see that SW-free systems have characteristic decomposition properties: a system is SW-free iff its unfolding[4] is decomposable into a set of initially sequential components, and stays so at every reachable state.

Definition 4.4.5 (IS, cis-decomposable systems). Let S be a system.

S is *initially sequential* (short: *IS*) iff $\forall r \in csteps(S).\ |r| \leq 1$.

S is *continuously decomposable into a set of initially sequential systems* (short: *cis-decomposable*) iff at every $s \in Reach(S)$ S has a decomposition $\mathcal{D}$ such that each $c \in \mathcal{D}$ is IS.

Theorem 4.4.2. *A system S is SW-free iff $unf(S)$[4] is cis-decomposable.*

Proof. The proof can be found in Appendix B.2. □

Fact 4.4.1 (facts about cis-decomposable systems).

1. *IS systems are bsc.*

2. *cis-decomposable systems are bsc-decomposable at every reachable state.*

[4]This can probably be strengthened to 'the system itself'.

3. The prime components of cis-decomposable systems are IS.

4. The prime components of cis-decomposable systems are cis-decomposable.

Proof. (1) is a consequence of the definition of IS and bsc. (2) follows from (1). (3) is immediate with (1) and Fact 4.3.3(2): certainly a cis-decomposable system is decomposable into IS components at its initial state.

(4) To the contrary, assume a cis-decomposable system S has a prime component c with a state s_c such that c cannot be decomposed into a set of IS components at s_c. There must be a state s in S such that the behaviour at s corresponds to the behaviour that is possible at s_c in parallel with behaviour of the prime components of S other than c. But then if the behaviour at s_c cannot be decomposed into IS components this is not possible for the behaviour at s either, contradicting that S is cis-decomposable. □

Note that the class of cis-decomposable systems *strictly* contains the class of parallel compositions of sequential systems: the prime components of a cis-decomposable system initially look like sequential systems, but by executing a transition each component may evolve into a set of independent sub-components. These must again be initially sequential, so that the whole system still consists of a set of IS components at the new state. From there, the system can 'fork' once again, and so on. Thus, as a first merit of Theorem 4.4.2 we obtain the following intuition: the SW situations capture all interaction between the sequential parts of a system apart from 'fork'. The typical structure of cis-decomposable systems is illustrated by Figure 4.6.

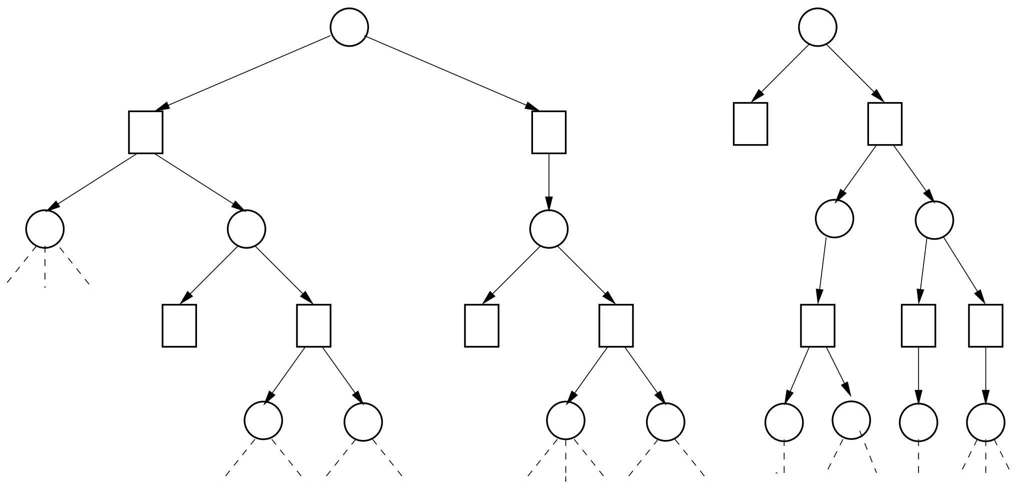


Figure 4.6: A cis-decomposable system

More concretely, Theorem 4.4.2 helps us to obtain our coincidence result: we show that hp and hhp bisimilarity coincide for *bounded-degree* (short for *concurrency-degree bounded*) cis-decomposable systems, and carry this result over to bounded-degree SW-free systems. Bounded-degree is a strengthening of our general restriction 'concurrency-degree finite' (cf. Section 2.1.4): in a bounded-degree system S the *smallest upper bound on the number of transitions that can be executed concurrently* (with respect to any state), say $cbound(S)$, is given by a natural number. This implies that whenever S can be decomposed into more than one prime component then $cbound(S)$ is strictly smaller for the components than for the entire system. Consequently, any bounded-degree cis-decomposable system will, from a certain point onwards, behave like a parallel composition of sequential systems. With Theorem 4.2.1(1) and the composition and decomposition results of Section 4.3 we can then prove our coincidence result by induction on $cbound(S)$; by Fact 4.4.1(2) it is ensured that the decomposition theorem does apply.

Definition 4.4.6 (bounded-degree). Let S be a system.

Given $s \in Reach(S)$ we lift $csteps(S)$ to $csteps(S, s)$ to denote the concurrent steps of S at reachable state s rather than the initial state in the obvious way.

The *smallest upper bound on the number of transitions that can be executed concurrently in* S, $cbound(S)$, is defined by

$$cbound(S) = \min\{\kappa \mid \forall s \in Reach(S).\ \forall r \in csteps(S, s).\ |r| \leq \kappa\}.$$

S is *bounded-degree* (short for *concurrency-degree bounded*) iff $cbound(S) \in \mathbb{N}_0$.

Proposition 4.4.2 (facts about $cbound$**).** *Let S be a system.*

1. $cbound(S) = cbound(unf(S))$.
2. *Let S' be a second system.* $S \sim_{hp} S' \;\Rightarrow\; cbound(S) = cbound(S')$.
3. *Let S be bounded-degree.*
   $|PComps(S)| > 1 \implies \forall c \in PComps(S).\ cbound(c) < cbound(S)$.

Proof. Let S be a system. (1) is obvious. (2) is immediate with Prop. 4.3.2(1). (3) follows by definition of bounded-degree and the fact that the prime components of S are non-empty, and independent of each other. □

Theorem 4.4.3.

1. *hp and hhp bisimilarity coincide for bounded-degree cis-decomposable systems.*

2. hp and hhp bisimilarity coincide for bounded-degree SW-free systems.

Proof. (1.) Let S_1 and S_2 be two bounded-degree cis-decomposable systems such that $S_1 \sim_{hp} S_2$. By definition $cbound(S_1)$ and $cbound(S_2)$ are both in $\mathbb{N}_0$, and by Prop. 4.4.2(2) we have $cbound(S_1) = cbound(S_2)$, say n. We prove $S_1 \sim_{hhp} S_2$ by induction on n. *Base case* $n = 0$: the two systems must be empty, and $S_1 \sim_{hhp} S_2$ is trivially given.

Inductive case $n > 0$: by Fact 4.4.1(2) S_1 and S_2 are bsc-decomposable, and we can apply Theorem 4.3.2(1) to obtain a bijection β between $PComps(S_1)$ and $PComps(S_2)$ such that $c_1 \sim_{hp} \beta(c_1)$ for all $c_1 \in PComps(S_1)$. First of all, this implies we have $PComps(S_1) = PComps(S_2)$, say m. Assume $m > 1$. Each $c_1 \in PComps(S_1)$ is cis-decomposable, and satisfies $cbound(c_1) < cbound(S_1)$; this follows from Fact 4.4.1(4), and Prop. 4.4.2(3) respectively. Together with $c_1 \sim_{hp} \beta(c_1)$ this means we can apply the induction hypothesis, and obtain $c_1 \sim_{hhp} \beta(c_1)$ for each $c_1 \in PComps(S_1)$. But then $S_1 \sim_{hhp} S_2$ follows from Theorem 4.3.1(2).

If $m = 1$ then with Fact 4.4.1(3) it is clear that S_1 and S_2 are IS. We distinguish between the following two cases: (a) S_1 and S_2 are sequential systems; (b) S_1 and S_2 initially behave like sequential systems, but will fork later on. Our case split is complete since in general we have: if a sequential system is hp bisimilar to another system S' then S' will also be sequential. If (a) holds then $S_1 \sim_{hhp} S_2$ follows from Theorem 4.2.1(1). In case (b) we proceed as follows. We match the sequential beginning by simply copying suitable tuples from any prefix-closed hp bisimulation relating S_1 and S_2. As soon as we reach a state s with $|PComps(s)| > 1$ in S_1, or S_2 equivalently, we proceed similarly to above when $m > 1$. It is straightforward to verify that in this manner we can obtain a hhp bisimulation for S_1 and S_2. If $m = 0$ then the base case applies.

(2.) This follows from (1) by Theorem 4.4.2, Prop. 4.4.2(1), and Fact 2.1.1. ☐

By Prop. 4.4.1 we also obtain:

Corollary 4.4.1. *hp and hhp bisimilarity coincide for bounded-degree comm-free net systems.*

4.5 Confusion

4.5.1 Introduction

We now consider a behavioural situation which results from the interplay of concurrency and conflict, the situation of *confusion*. Assume that two transitions, t_1 and t_2, are concurrently enabled at some state. It can happen that the occurrence of one of them, say t_2, brings about a change in the set of transitions that are in conflict with the other one, t_1. This is what makes an instance of confusion. The confusion lies in the fact that the two possible linear realizations of the concurrent step $\{t_1, t_2\}$ have different behavioural properties: t_1 resolves a conflict in one of the linearizations which it does not decide in the other one (because the conflict is either not visible or has already been resolved by t_2). Good accounts of confusion can be found in [RT86] and [RE98]; according to the first reference this behavioural situation was probably first identified by Holt.

Definition 4.5.1 (conflict set, confusion, confused). Let S be a system, and $s \in \mathit{Reach}(S)$.

Let $t \in T_S$ such that $s[t\rangle$. The *conflict set of t at s*, denoted by $\mathit{cfl}(s,t)$, is the set $\{t' \in T_S \mid s[t'\rangle \ \&\ t\ D_S\ t'\}$.

Let $t_1, t_2 \in T_S$ such that $s[t_1\rangle$, $s[t_2\rangle$ & $t_1\ I_S\ t_2$. The triple (s, t_1, t_2) is a *confusion (at s)* iff $\mathit{cfl}(s,t_1) \neq \mathit{cfl}(s',t_1)$, where s' is such that $s \xrightarrow{t_2} s'$.

We say S is *confused at s* iff there is a confusion at s.

It is common to classify instances of confusion as follows.

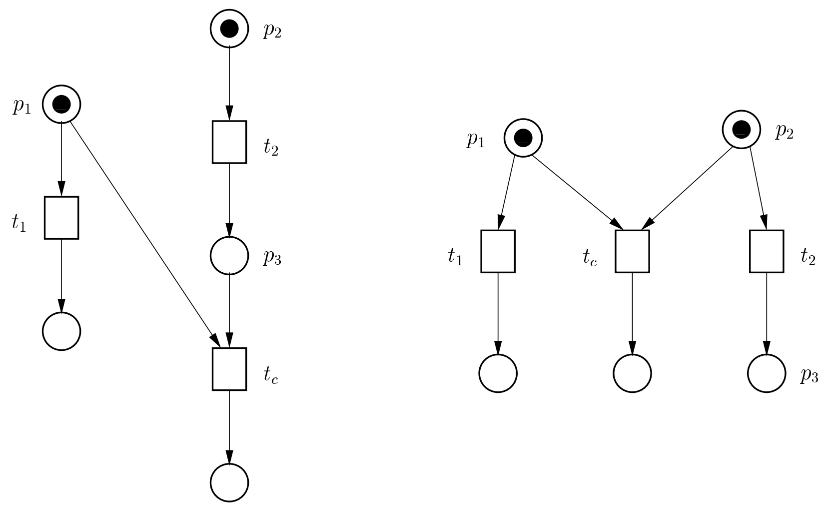
Definition 4.5.2 (types of confusion). Let S be a system, $s \in \mathit{Reach}(S)$, t_1, $t_2 \in T_S$, and let $\gamma = (s, t_1, t_2)$ be a confusion.

Let s' be such that $s \xrightarrow{t_2} s'$. γ is a *conflict-increasing confusion* (short: *ci-confusion*) iff $\mathit{cfl}(s,t_1) \subset \mathit{cfl}(s',t_1)$, and γ is a *conflict-decreasing confusion* (short: *cd-confusion*) iff $\mathit{cfl}(s,t_1) \supset \mathit{cfl}(s',t_1)$.

γ is *symmetric* iff (s, t_2, t_1) is also a confusion, and *asymmetric* otherwise.

The classification of confusions into ci and cd is not exhaustive: as shown in [RT86] and [RE98] there are examples of confusions which are neither ci nor cd. Also note that the two lines of classification are independent, the only nontrivial relation between the two is that cd confusions are always symmetric [RT86, RE98]. To further analyse a situation of confusion we can also consider whether it is a confusion with respect to a particular transition.

Definition 4.5.3 (t_c-confusion). Let S be a system, $s \in \mathit{Reach}(S)$, t_1, $t_2 \in T_S$, such that $\gamma = (s, t_1, t_2)$ is a confusion. Given $t_c \in T_S$, we say γ is a *confusion w.r.t. t_c* (short: *t_c-confusion*) iff we have either (a) $t_c \notin \mathit{cfl}(s,t_1)$ & $t_c \in \mathit{cfl}(s',t_1)$ or (b) $t_c \in \mathit{cfl}(s,t_1)$ & $t_c \notin \mathit{cfl}(s',t_1)$, where s' is such that $s \xrightarrow{t_2} s'$. If (a) holds then γ is a *conflict-increasing* (short: *ci*) t_c-confusion, and in the case of (b) a *conflict-decreasing* (short: *cd*) t_c-confusion.



A **B**

Figure 4.7: The two basic cases of confusion

By definition such relative confusions exhaustively fall into ci or cd. They can also be classified along the line symmetric – asymmetric, but at least for 1-safe net systems, the cd t_c-confusions exactly correspond to the symmetric t_c-confusions, and similarly for ci and asymmetric t_c-confusions. Figure 4.7 depicts the two basic cases of confusion:

Basic confusion A. Set $s = \{p_1, p_2\}$. The triple (s, t_1, t_2) is a ci t_c-confusion. Observe that the potential conflict of t_1 with t_c is not visible at s and how the occurrence of t_2 *introduces* this conflict.

Basic confusion B. Set $s = \{p_1, p_2\}$. The triple (s, t_1, t_2) is a cd t_c-confusion. Note how the conflict of t_1 with t_c is *resolved* by the occurrence of t_2. Similarly, the occurrence of t_1 would resolve the conflict of t_2 with t_c, which makes the confusion symmetric.

As reported in [RT86] and [RE98] confusion seems to be a fundamental phenomenon that occurs wherever both, concurrency and conflict, are present; for example, it appears as the so-called glitch problem in the area of switching circuits. In net theory, confusion is well-known to make the analysis of systems more difficult. And, as stated in [RE98] and [RT86], it has even been suggested that "it is not the combination of choice and concurrency as such that causes difficulties. Only those combinations of concurrency and conflict that result in confusion create problems." Thus, it seems worthwhile to investigate what influence this behavioural situation has on the coincidence problem of hp and hhp bisimilarity.

4.5.2 Confusion and the Coincidence Problem

With the explanation of above it seems immediately clear that confusion is significant for separating hhp from hp bisimilarity. This is confirmed by a look at the two standard counter-examples:

Counter-example 1 (Figure 1.7). Consider system A. The triple $\gamma = (s, b_2, a_2)$ is a confusion, which in general is symmetric but neither ci nor cd. On the one hand, γ is a ci c-confusion, which is caused by one of the MNH-situations employed in the example. On the other hand, γ is a cd a_1-, and also a cd b_1-confusion; this results from the counter-example's frame situation. There are corresponding confusions in the other square of system A, and in system B.

Counter-example 2 (Figure 1.8). Consider system B. The triple $\gamma = (s', a'_1, b'_1)$ is a confusion, which in general is cd and symmetric. On the one hand, γ is a cd a'_2-, and also a cd b'_2-confusion; this is caused by the frame situation employed in the example. On the other hand, γ is a cd c'_1- and also a cd c'_2-confusion; in turn, this originates from the integration of the counter-example's MNH-situation into its frame situation. By themselves the MNH-situations do not give rise to confusion. (cf. Section 4.1.1). Again, there are corresponding confusions in the other squares of system B, and also in system A.

It has long been believed that confusion is not only significant but also *necessary* to distinguish hp and hhp bisimilarity. Specifically, in [Che96] it has been conjectured that the two bisimilarities coincide for *free choice net systems*, which are a structural characterization of *confusion-free systems*.

Definition 4.5.4. A system S is *confusion-free* iff for all $s \in \mathit{Reach}(S)$ S is not confused at s.

The conjecture is, however, incorrect: it is easy to check that our third counter-example (Figure 4.1, 4.2) does not contain any instance of confusion, and, as we will be able to tell later, the two systems are actually free choice.

Theorem 4.5.1.
hp and hhp bisimilarity do not coincide, in general, for confusion-free systems.

A more compact counter-example is presented in Figure 4.8. Again the two systems are confusion-free, and they are also free choice. In both, A and B, we can concurrently compute an a- and a b-transition such that afterwards the system will either have reached a final state, or the a-thread and b-thread can synchronize via a c-transition. The crucial difference is that in A we can compute an a- and

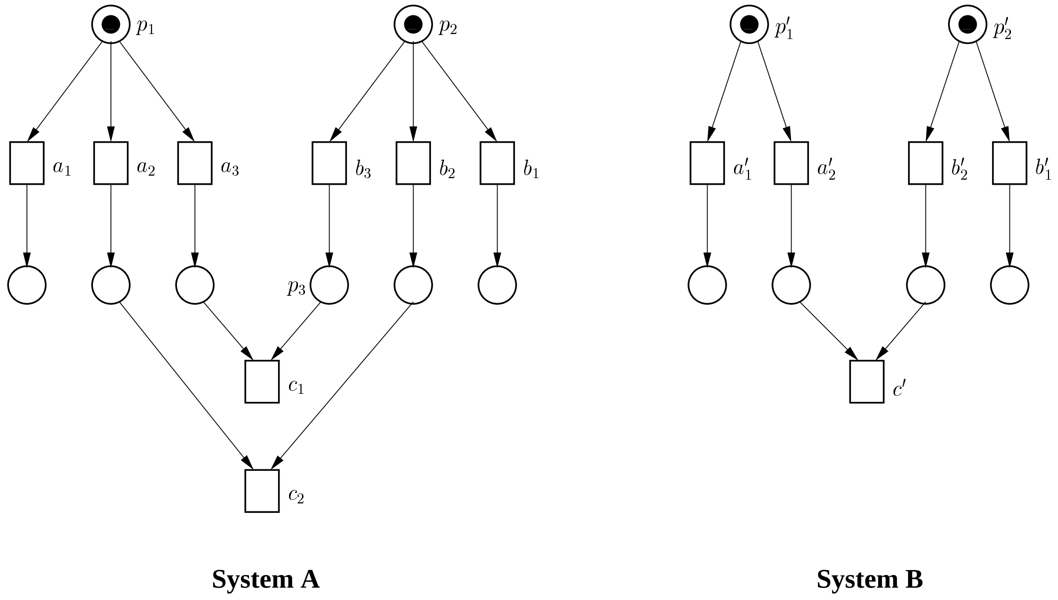


Figure 4.8: Counter-example 4: a more compact counter-example that demonstrates non-coincidence for confusion-free systems

a b-transition such that the resulting threads cannot synchronize with each other but *both of them* could do if they were in parallel with their true synchronization partner. This difference is easily detected by hhp bisimilarity via backtracking but cannot be seen by hp bisimilarity.

4.5.3 A New Kind of Confusion: Syn-confusion

Although we have seen that counter-example 3 and 4 do not employ any instance of confusion in the sense of Def. 4.5.1 it is still intuitive that every counter-example should rely on some kind of confusion (in the non-formal sense). A closer look at counter-example 3 and 4 shows that they both use MNH situations that can be captured by a new type of confusion. Consider counter-example 3:

Counter-example 3 (Figure 4.1, 4.2). Consider system S. Set $s = \{p_1, p_2\}$, and observe: the transitions e_1 and a are concurrently enabled at s. The sequence '$b\,f_2$' is enabled at s such that the threads '$b\,f_2$' and e_1 can occur concurrently, and then synchronize via the g_1-transition. Although a is independent of e_1, a's occurrence has an impact on the 'behavioural environment' of e_1: if a occurs it will take away the g_1-synchronization capability of e_1.

We can capture this scenario in the style of confusion as follows: a situation of *syn-confusion* is present at a state s iff there are two transitions, t_1 and t_2, concurrently enabled at s such that the occurrence of one of them, say t_2, brings about a change in the set of transitions that can occur as a *synchronization of*

t_1 later on; we say *a transition t_s can occur as a synchronization of t_1* later on iff there is a thread r enabled concurrently with t_1 at s such that t_s can occur causally dependent on t_1 and r. The confusion lies in the fact that the two possible linearizations of the concurrent step $\{t_1, t_2\}$ have different behavioural properties: if t_1 is computed before t_2, it occurs as a transition that has a specific synchronization capability, which it does not have if it is computed after t_2. Formally, we define:

Definition 4.5.5 (synchronization, synchronization set, syn-confusion). Let S be a system, $s \in \mathit{Reach}(S)$, and $t \in T_S$ such that $s[t\rangle$.

A transition t_s is a *synchronization of t at s* iff there is $w \in T_S^*$ with $s[w\rangle$ & $w\ I_S\ t$, and $t' \in T_S$ such that (s', t, t', t_s) is a SW-1 situation, where s' is such that $s \xrightarrow{w} s'$.

The *synchronization set of t at s*, denoted by $\mathit{syn}(s,t)$, is defined to be the set $\{t_s \in T_S \mid t_s \text{ is a synchronization of } t \text{ at } s\}$.

Let t_1, $t_2 \in T_S$ so that $s[t_1\rangle$, $s[t_2\rangle$ & $t_1\ I_S\ t_2$. We say the triple (s, t_1, t_2) is a *synchronization-confusion* (short: *syn-confusion*) *(at s)* iff $\mathit{syn}(s, t_1) \neq \mathit{syn}(s', t_1)$ (or equivalently $\mathit{syn}(s, t_1) \supset \mathit{syn}(s', t_1)$), where s' is such that $s \xrightarrow{t_2} s'$.

Let $\gamma = (s, t_1, t_2)$ be a syn-confusion, and $t_s \in T_s$. γ is a *syn-confusion w.r.t. t_s* (short: t_s *syn-confusion*) iff $t_s \in \mathit{syn}(s, t_1)$ & $t_s \notin \mathit{syn}(s', t_1)$, where s' is such that $s \xrightarrow{t_2} s'$.

We say S is *syn-confused at s* iff there is a syn-confusion at s.

Similarly to confusion, instances of syn-confusion can be classified into symmetric and asymmetric; this is even fruitful for relative syn-confusions. On the other hand, as a consequence of the definition syn-confusions are always decreasing. The example shown in Figure 4.9 can be regarded as the most basic case of syn-confusion:

Basic Syn-confusion. Set $s = \{p_1, p_2\}$ and $s' = \{p_1, p_3\}$. The triple (s, t_1, t_2) is a syn-confusion (w.r.t. t_s) since $\mathit{syn}(s, t_1) = \{t_s\}$ but $\mathit{syn}(s', t_1) = \emptyset$. Note how the occurence of t_2 takes away t_1's synchronization capability.

We already know that syn-confusion is significant for separating hhp from hp bisimilarity. Just as with classical confusion, it is *not* a necessary condition. This fact is quickly established by a look at the two standard counter-examples: they do not contain any instance of syn-confusion.

Definition 4.5.6. A system S is *syn-confusion free* iff for all $s \in \mathit{Reach}(S)$ S is not syn-confused at s.

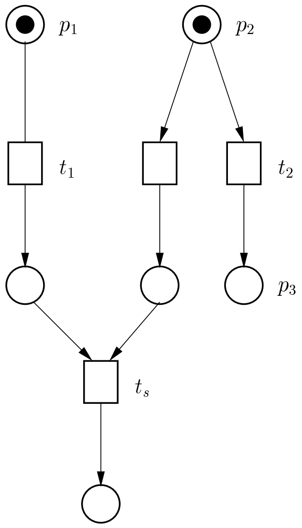


Figure 4.9: The basic case of syn-confusion

Theorem 4.5.2. *hp and hhp bisimilarity do not coincide, in general, for syn-confusion free systems.*

The question is whether excluding both, classical confusion *and* syn-confusion, will make hp and hhp bisimilarity coincide. We will pursue this theme in Chapter 6. There, we will study the coincidence problem for a (half-)structural system class that apart from being confusion-free appears to exclude syn-confusion: the class of *live free choice net systems*.

4.6 Liveness

Finally, we introduce the well-known concept of *liveness*. This behavioural property will be particularly interesting later on, when we study it in combination with the structural condition of *free choice*. A system is *live* iff each of its transitions can always be made to occur again; formally, we define:

Definition 4.6.1 (liveness, well-formedness). Let S be a system.

A transition $t \in T_S$ is *live* iff $\forall s \in \mathit{Reach}(S).\ \exists s' \in \mathit{Reach}(S, s).\ s'[t\rangle$.

S is *live* iff $\forall t \in T_S.\ t$ is live.

Consider S as a system base, that is without a specified initial state. S is *well-formed* iff there exists a state $s \in S_S$ such that S with initial state s is a live system.

Example 4.6.1. The system of Figure 4.10 is live, whereas the systems of Figure 1.7 are not.

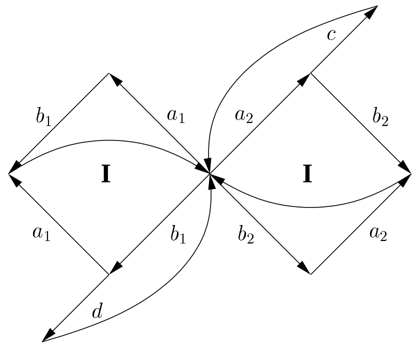


Figure 4.10: A live version of system A of Figure 1.7

Considering liveness by itself is not very interesting for the two hhp bisimilarity problems: the systems of the two standard counter-examples (Figure 1.7 and 1.8) can easily be transformed into live systems such that they still demonstrate the non-coincidence of hp and hhp bisimilarity. The analogue is true for the systems employed in the undecidability proof of hhp bisimilarity (cf. [JN00]). In both cases, one simply adds a 'resetting' transition from every final state to the initial state, just as it has been done in the example above. We record:

Theorem 4.6.1.

1. *hp and hhp bisimilarity do not coincide, in general, for live systems.*

2. *hhp bisimilarity is undecidable for live systems.*

However, it will be worthwhile to study liveness in combination with the structural property of free choice; as mentioned in the previous section, this combination appears to exclude both confusion and syn-confusion.

Chapter 5

Basic Parallel Processes

5.1 Introduction

We now come to investigate hp and hhp bisimilarity on our first group of structural system classes: the process algebra *basic parallel processes* (short: *BPP*) and its specialization to *simple BPP* (short: *SBPP*), equipped with partial order semantics. BPP and SBPP come from the area of infinite-state verification, and constitute the natural classes to start with when exploring partial order paradigms on infinite-state systems (cf. Section 1.3.2). It turns out that they restrict the interplay of causality, concurrency, and conflict in an interesting way: under our semantics, SBPP are interpreted as a class of (infinite-state) comm-free net systems, and BPP as a class of (infinite-state) *proper-comm free* net systems. From Section 4.4 we know that the first are SW-free; proper-comm free is a new concept, and we will see that net systems satisfying this condition are SW-{1,2} free.

We obtain the following results: for SBPP we prove decidability and coincidence of hp and hhp bisimilarity. Since SBPP are not necessarily bounded-degree but always finitely describable, this result is orthogonal to the one of Section 4.4.3. For BPP the two bisimilarities do not coincide; this follows from the second standard counter-example. But we separately achieve decidability for both. Besides, our proofs lead us to two coincidence results: for BPP, hp bisimilarity coincides with the distributed bisimilarity of [Cas88, CH89], and hhp bisimilarity with chhp bisimilarity. As noted earlier, the decidability of hp bisimilarity for BPP is already known via that of causal bisimilarity [KH94], and the coincidence of hp and distributed bisimilarity is also proved in [Kie99]; further, [Chr92, Chr93] prove the decidability of distributed bisimilarity. The results concerning hhp bisimilarity are all new.

There is a general scheme behind our proofs: due to their restricted synchronization comm-free and proper-comm free net systems have good composition and decomposition properties. We will see that this translates into corresponding composition and decomposition results for hp and hhp bisimilarity, first of all on the level of the respective net class, and second on the syntactic level of process terms. Based on the syntactic insights we then construct tableau systems which establish the decidability and coincidence results.

Altogether our proofs confirm the predicted trend that partial order paradigms on standard classes of infinite-state systems can have clear decision procedures due to good composition/decomposition properties. Importantly, the results on hhp bisimilarity show that in the non-interleaving world equivalences can be computationally harder for finite-state systems than for standard classes of infinite-state systems; it is primarily the interplay of causality, concurrency, and conflict that matters.

A word about our partial order semantics: inspired by the SBPP semantics of [EK95] we translate each given BPP or SBPP into an occurrence net that intuitively presents the partial order unfolding of the process. For SBPP we can employ the standard notion of Petri net unfolding: SBPP can be identified with 'full standard form' BPP, and these in turn have a well-known interpretation as (weighted, non-safe) comm-free Petri nets. For BPP there is no suitable Petri net representation, and we shall therefore develop a direct notion of BPP unfolding.[1]

The remainder of the chapter is organized as follows: in Section 5.2 we give the necessary definitions concerning BPP and SBPP, and explain the tableau method in more detail. Afterwards we present the partial order semantics and results, first for SBPP in Section 5.3, then for BPP in Section 5.4. Finally, we draw conclusions and give directions for further research.

5.2 Definitions and Methodology

5.2.1 BPP and SBPP

We first give the definition of BPP; we follow [Chr93] but for technical convenience we use labelled transitions instead of the usual actions. In the following assume a countably infinite set of transitions $T = \{t, t_1, \ldots, u, u_1, \ldots\}$, and a countably infinite set of process variables $Vars = \{X, X_1, \ldots, Y, Y_1, \ldots\}$.

[1] Note that for our purpose it is perfectly natural to employ a semantics of unfoldings; using a truly-concurrent operational semantics would not give us any advantage since our equivalences are based on partial order runs.

Definition 5.2.1 (BPP expressions). *BPP expressions* are given by the following grammar:

$$\begin{array}{rcll} E & ::= & \mathbf{0} & \text{(inaction)} \\ & \mid & X & \text{(process variable, } X \in \mathit{Vars}) \\ & \mid & t.E & \text{(transition prefix, } t \in T) \\ & \mid & E + E & \text{(choice)} \\ & \mid & E \mid\mid E & \text{(parallel composition)} \end{array}$$

Let E be a BPP expression. We denote the set of variables that occur in E by $\mathit{Vars}(E)$, and the set of transitions by $T(E)$.

E is *guarded* iff every variable in E occurs within the scope of transition prefix. E is *transition-genuine* iff every $t \in T(E)$ appears syntactically only once in E.

We usually consider BPP expressions modulo the structural congruence $\equiv$: let $\equiv$ be the least syntactic congruence satisfying associativity, commutativity, and $\mathbf{0}$ as unit for the operators choice and parallel composition.

Informally, the meaning of the operators can be understood as follows: $\mathbf{0}$ represents the inactive process that is not capable of performing any transition. The process $t.E$ can perform transition t and thereby evolve into E. $E + F$ behaves either like E or like F, depending on whether the first transition is chosen from E or F. In the parallel composition $E \mid\mid F$, the components E and F act concurrently and independently of each other.

We allow variables in BPP expressions as a means of defining recursive processes. The meaning of the variables will be determined by a system of equations that associates a defining expression to each variable. By assuming defining expressions to be guarded we ensure that recursive definitions yield unique solutions. In our setting of labelled transitions it is natural to further require that defining expressions are transition-genuine, and that the transitions of distinct defining expressions are disjoint. The labels of the transitions will be specified by a labelling function.

Definition 5.2.2 (BPP defining system). A *BPP defining system* is a triple $\Delta = (T_\Delta, l_\Delta, \Delta)$, where

- $T_\Delta \subset T$ is a finite set of transitions,
- $l_\Delta : T_\Delta \to \mathit{Act}$ is a labelling function, and
- $\Delta = \{X_i \stackrel{def}{=} E_i \mid i = 1, 2, \ldots, n\}$ is a finite family of recursive process equations, where the X_i are distinct variables, and the E_i are guarded and transition-genuine BPP expressions such that $\forall i \in [1, n]$. $\mathit{Vars}(E_i) \subseteq$

$$\frac{}{t.E \xrightarrow{t} E} \qquad \frac{E \xrightarrow{t} E'}{X \xrightarrow{t} E'} (X \stackrel{def}{=} E) \in \Delta$$

$$\frac{E \xrightarrow{t} E'}{E + F \xrightarrow{t} E'} \qquad \frac{F \xrightarrow{t} F'}{E + F \xrightarrow{t} F'}$$

$$\frac{E \xrightarrow{t} E'}{E \mid\mid F \xrightarrow{t} E' \mid\mid F} \qquad \frac{F \xrightarrow{t} F'}{E \mid\mid F \xrightarrow{t} E \mid\mid F'}$$

Figure 5.1: Transition rules relative to a BPP defining system Δ

$Vars(\Delta) \stackrel{def}{=} \{X_1, X_2, \ldots, X_n\}$ & $T(E_i) \subseteq T_\Delta$, and $\forall i, j \in [1, n].\ i \neq j \implies T(E_i) \cap T(E_j) = \emptyset$.

Let Δ be a BPP defining system. We say a *BPP expression E is defined by* Δ iff $Vars(E) \subseteq Vars(\Delta)$ and $T(E) \subseteq T_\Delta$. We denote the set of BPP expressions defined by Δ by $BPP(\Delta)$.

A BPP is a BPP expression that is defined by a defining system:

Definition 5.2.3 (BPP). A *BPP* is a pair $\mathcal{E} = (\Delta_\mathcal{E}, E_0)$, where $\Delta_\mathcal{E}$ is a BPP defining system, and $E_0 \in BPP(\Delta_\mathcal{E})$. If the defining system is clear from the context, we shall denote a BPP simply by its BPP expression.

The standard interleaving semantics for BPP is given via the SOS rules of Figure 5.1: every BPP defining system Δ determines a lts $\mathcal{T}_\Delta = (BPP(\Delta), T_\Delta, \rightarrow_\Delta)$, where $\rightarrow_\Delta$ is the least relation satisfying the transition rules; elements of T_Δ can be substituted by elements of Act via l_Δ. A BPP $\mathcal{E} = (\Delta_\mathcal{E}, E_0)$ can be understood as the ltsis $\mathcal{T}_\mathcal{E} = (\mathcal{T}_\Delta, E_0)$. Note how the semantics implements the informal meaning of the BPP operators.

SBPP. We now define the subclass SBPP following [EK95]. SBPP restrict the nesting of 'choice' and parallel composition by requiring choice to be guarded in that the expressions of a choice must be prefixed terms. As a consequence every SBPP corresponds to a parallel composition of initially sequential processes.

In the following assume two countably infinite subsets of $Vars$, $Vars^{[E]} = \{X, X_1, X_2, \ldots\}$ and $Vars^{[S]} = \{Y, Y_1, Y_2, \ldots\}$.

Definition 5.2.4 (SBPP). *SBPP expressions* are defined by the following grammar:

$$\begin{array}{rcll} E & ::= & S & \text{(an initially sequential process)} \\ & \mid & X & \text{(process variable, } X \in \mathit{Vars}^{[E]}) \\ & \mid & E \,||\, E & \text{(parallel composition),} \end{array}$$

where *initially sequential process* (short: *ISP*) *expressions* are given by:

$$\begin{array}{rcll} S & ::= & \mathbf{0} & \text{(inaction)} \\ & \mid & Y & \text{(process variable, } Y \in \mathit{Vars}^{[S]}) \\ & \mid & t.E & \text{(transition prefix, } t \in T,\ E \text{ a SBPP)} \\ & \mid & S + S & \text{(choice).} \end{array}$$

Clearly, SBPP and ISP expressions are both special kinds of BPP expressions.

A *SBPP defining system* is a BPP defining system Δ such that for each '$X \stackrel{def}{=} E$' $\in \Delta$ with $X \in \mathit{Vars}^{[E]}$, E is a SBPP expression, and for each '$Y \stackrel{def}{=} S$' $\in \Delta$ with $Y \in \mathit{Vars}^{[S]}$, S is an ISP expression. We denote the set of SBPP expressions defined by Δ by $\mathit{SBPP}(\Delta)$.

A SBPP is a pair $\mathcal{E} = (\Delta_{\mathcal{E}}, E_0)$, where $\Delta_{\mathcal{E}}$ is a SBPP defining system, and $E_0 \in \mathit{SBPP}(\Delta_{\mathcal{E}})$.

5.2.2 Normal Forms

When working with process algebras it is often convenient to restrict one's attention to processes that are in a certain normal form. Naturally, this requires that every process can effectively be transformed into a semantically equivalent normal form process. In the interleaving world it is common to consider BPP in so-called *full standard form* (*SNF*) [Chr93]. Every BPP can be represented as a bisimilar SNF process; however, the transformation relies on the expansion law, and therefore SNF is not valid under partial order semantics. We shall introduce a new concept for BPP, the so-called *execution normal form* (*ENF*); for SBPP we can still employ the simpler SNF (with more generous initial expression).

We have not given a partial order semantics yet; in fact, for technical convenience we would like to define it on normal form processes only. How can we then be sure that we capture the entire class of BPP, and SBPP respectively? Our normal forms are very unrestricted in that every BPP and SBPP can effectively be transformed into a normal form representative by using operations that only affect the appearance of the defining expressions, that is 'syntactic' operations like introduction of new variables, substitution of variables for subexpressions, and unfolding of variables. Certainly, any semantic equivalence of interest is preserved under such operations.

BPP in ENF are based on ENF expressions; these are BPP expressions in which every variable occurrence is immediately guarded and every transition prefix is directly followed by a variable. In other words, ENF expressions are based on subexpressions of the form $t.X$ or $\mathbf{0}$, which are arbitrarily nested by choice and parallel composition.

Definition 5.2.5 (BPP in ENF). The class of *BPP expressions in execution normal form* (short: *ENF expressions*) is defined by the following grammar:

$$E ::= \mathbf{0} \mid t.X \mid E + E \mid E \mid\mid E.$$

In the following we assume ENF expressions to be transition-genuine (as defined in Def. 5.2.1).

A BPP defining system Δ is *in ENF* iff for each '$X \stackrel{def}{=} E$' $\in \Delta$, E is in ENF. To abbreviate we also use the term *ENF defining system* for 'BPP defining system in ENF'. We denote the set of ENF expressions defined by an ENF defining system Δ by $\mathit{ENF}(\Delta)$.

A BPP $\mathcal{E} = (\Delta_{\mathcal{E}}, E_0)$ is *in ENF* iff $\Delta_{\mathcal{E}}$ is in ENF, and $E_0 \in \mathit{ENF}(\Delta_{\mathcal{E}}) \cup \mathit{Vars}(\Delta_{\mathcal{E}})$.

Let $\Delta = \{X_i \stackrel{def}{=} E_i \mid i = 1, \ldots, n\}$ be an ENF defining system such that $T_\Delta = \bigcup_{i \in [1,n]} T(E_i)$, and $t \in T_\Delta$. t appears in exactly one defining equation '$X \stackrel{def}{=} E$' $\in \Delta$; we denote X by $\mathit{prevar}(t)$. Further, there will be exactly one subexpression of the form '$t.Y$' in E; we denote Y by $\mathit{postvar}(t)$.

The name "execution normal form" comes from the fact that the transitions occurring in a defined ENF expression are exactly the ones that can be executed as next transition via the standard semantics:

Proposition 5.2.1. *Let Δ be an ENF defining system. For all $E \in \mathit{ENF}(\Delta)$, $t \in T_\Delta$ we have:*

$$E \xrightarrow{t} \iff t \in T(E).$$

Proof. Obvious from the definitions. □

For SBPP we shall employ SNF, or to be precise, a slight generalization of SNF concerning the initial expression.

For a given defining system Δ, let $\mathit{Vars}(\Delta)^{\otimes} = \{\alpha, \beta, \ldots\}$ be the set of finite multisets over $\mathit{Vars}(\Delta)$. We identify a multiset $\alpha = \{X, X, Y\}$ with the parallel composition $X \mid\mid X \mid\mid Y$; the empty multiset will be recognized as the process $\mathbf{0}$. We allow sums to be written via $\sum$, and correspondingly identify the empty sum with $\mathbf{0}$.

Definition 5.2.6 (SBPP in SNF). A SBPP defining system Δ is in *full standard form* (short: *SNF*) iff for every '$X \stackrel{def}{=} E$' $\in \Delta$, E is of the form

$$E = \sum_{i=1}^{n} t_i.\alpha_i,$$

where $t_i \in T_\Delta$, and $\alpha_i \in \mathit{Vars}(\Delta)^{\otimes}$ for $i \in [1, n]$. To abbreviate we also use the term *SNF defining system* for 'SBPP defining system in SNF'.

A SBPP $\mathcal{E}$ is *in SNF* iff $\Delta_\mathcal{E}$ is in SNF, and $E_0 = \alpha$ with $\alpha \in \mathit{Vars}(\mathcal{E})^{\otimes}$, or $E_0 = \sum_{i=1}^{n} t_i.\alpha_i$ with $t_i \in T_\mathcal{E}$, $\alpha_i \in \mathit{Vars}(\mathcal{E})^{\otimes}$.

Let $\Delta = \{X_i \stackrel{def}{=} E_i \mid i = 1, \ldots, n\}$ be a SNF defining system such that $T_\Delta = \bigcup_{i \in [1,n]} T(E_i)$, and $t \in T_\Delta$. t appears in exactly one defining equation '$X \stackrel{def}{=} \sum_{i=1}^{n} t_i.\alpha_i$' $\in \Delta$; we denote X by $\mathit{prevar}(t)$. Further, there is exactly one $i \in [1, n]$ with $t_i = t$; we denote α_i by $\mathit{postvars}(t)$.

It is a routine exercise to check that, as motivated above, our normal forms are indeed semantically nonrestrictive in that every BPP (SBPP) can effectively be transformed into a BPP in ENF (SBPP in SNF) by using operations that only affect the appearance of the defining expressions.

5.2.3 The Tableau Technique

The results of this chapter will be proved by means of tableau systems. In preparation, we now describe the tableau method in technical detail, geared to our purpose.

A *tableau system* is a syntax-driven goal-directed proof scheme. To prove that two processes E and F are equivalent w.r.t. to a given equivalence one starts with the goal '$E = F$', and builds from this root node a *tableau*, or proof tree, as prescribed by the system's *tableau rules* and *terminal conditions*.

The tableau rules specify how goals can be substituted by a set of subgoals. For our purpose they are of the form:

$$\frac{E = F}{E_1 = F_1 \qquad \cdots \qquad E_n = F_n}$$

sometimes with side conditions. The premise '$E = F$' represents the goal, and the consequents '$E_i = F_i$' the subgoals that have to be achieved. The application of the rules is steered by the structure of the processes. But we allow rule instantiations to be nondeterministic.

Terminal conditions are of the form '$E = F$', usually with side conditions. If a node matches a terminal condition the construction will stop at this point. The node is called a *terminal node*; it will constitute a leaf in the proof tree. Terminal conditions are classified as either *successful* or *unsuccessful*. The matching of terminal conditions to nodes will be deterministic, and hence terminal nodes can also be classified as either successful or unsuccesful. A tableau is successful iff it is finite and all its terminals are successful, and unsuccessful otherwise.

The intuition is that a goal '$E = F$' is true iff there is a successful tableau with root node '$E = F$'. Two properties are crucial to ensure that this is guaranteed: the tableau system must be *complete* in that whenever the root is correct it is possible to construct a successful tableau. On the other hand, the tableau system must be *sound* in that whenever the root is false it is *not* possible to construct a successful tableau. If we want to employ a tableau system as a decision procedure then we further need *finiteness*: if for any two given processes there is only a finite number of possible tableaux, and each of them is finite then we can decide the respective equivalence by exhaustive search; simply construct all possible tableaux, and check if there is a successful one among them. We shall also use tableau systems to establish coincidence results: if a finite tableau system is complete and sound for two a priori different equivalences, then the two equivalences are decided by the same decision procedure, and hence they clearly coincide with each other.

Finally, we introduce some standard tableau terminology as it can be found e.g. in [CHM93a]: we denote a tableau with the root labelled '$X = Y$' by $T(X = Y)$. We use the letter π to designate paths through a tableau, and the letter n to denote nodes of a tableau. When we want to indicate the label of a node n we write $n : E = F$.

5.2.4 Further Definitions

In view of our partial order semantics, we introduce some more definitions; most of them are standard, and we define them following [EK95].

A *(labelled) weighted net* is a tuple (S, T, W, l), where S and T are disjoint sets of places and transitions, $W : (S \times T) \cup (T \times S) \to \mathbb{N}_0$ is a *weight function*, and $l : T \to Act$ is a labelling function. The pre-set of an element $x \in S \cup T$, ${}^\bullet x$, is defined by $\{y \in S \cup T \mid W(y, x) > 0\}$, the post-set of x, $x^\bullet$, similarly is $\{y \in S \cup T \mid W(x, y) > 0\}$. A *weighted Petri net* is a pair (N, M_0), where N is a weighted net and M_0 is a marking of N; markings are defined as for unweighted nets.

Let (S, T, F, l) be a net and let $x_1, x_2 \in S \cup T$. We say x_1 and x_2 are in *conflict*, denoted by $x_1 \# x_2$, if there exist distinct transitions $t_1, t_2 \in T$ such that

${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, and there exist paths in the net leading from t_1 to x_1, and from t_2 to x_2. $x \in S \cup T$ is in *self-conflict* iff $x\#x$. Note that this definition of conflict is not intuitive for general nets, but in the context of our partial order semantics we will think of *occurrence nets*.

A *(labelled) occurrence net* is an acyclic net $N = (B_N, E_N, F_N, l_N)$ such that:

- for every $b \in B_N$, $|{}^\bullet b| \leq 1$,
- N is finitely preceded, i.e., for every $x \in B_N \cup E_N$, the set of elements $y \in B_N \cup E_N$ such that there exists a path from y to x is finite,
- no $e \in E_N$ is in self-conflict.

We call the places of occurrence nets *conditions*, and the transitions *events*. For two elements $x, y \in B_N \cup E_N$ we define $x \leq_N y$ iff there exists a path from x to y. Since occurrence nets are acyclic, it is clear that $\leq_N$ is a partial order. We denote the set of minimal elements of $B \cup E$ with respect to $\leq_N$ by $Min(N)$.

Let $\mathcal{N}_1 = (N_1, M_1^0)$ and $\mathcal{N}_2 = (N_2, M_2^0)$ be net systems such that N_1 and N_2 are occurrence nets. We write $\mathcal{N}_1 \overset{occ}{=} \mathcal{N}_2$ iff the reachable part of $\mathcal{N}_1$, $reach(\mathcal{N}_1)$, and that of $\mathcal{N}_2$, $reach(\mathcal{N}_2)$, are isomorphic. $reach(N, M_0)$, where N is an occurrence net, is defined by $((S', T', F', l'), M_0)$, where $S' = S_N \cap (\uparrow_N M_0)$, $T' = T_N \cap (\uparrow_N M_0)$, $F' = F_N \cap ((S' \times T') \cup (T' \times S'))$, and $l' = l_N \restriction_{T'}$; $\uparrow_N$ is interpreted with respect to $\leq_N$.

We shall also employ the following terminology: given a net system $\mathcal{N}$, we use $En(\mathcal{N})$ to denote the set of transitions that are initially enabled in $\mathcal{N}$, that is $En(\mathcal{N}) := \{t \in T_N \mid M_0[t\rangle\}$.

5.3 Simple Basic Parallel Processes

5.3.1 Partial Order Semantics

Analogously to [EK95] we first translate SBPP into communication-free Petri nets, and then use the notion of Petri net unfolding given in [Eng91]. As explained in Section 5.2.2 it is justified to assume SBPP in SNF, and we can therefore employ the standard characterization of SNF processes as communication-free Petri nets [Esp97b]. This is different to [EK95], where a direct transformation for the entire SBPP class is developed. The concept of communication-free for weighted Petri nets is defined as follows:

Definition 5.3.1. A weighted net $N = (S, T, W, l)$ is *communication-free* iff $\forall t \in T.\ |{}^\bullet t| = 1\ \&\ \forall s \in S, t \in T.\ W(s,t) \leq 1$.

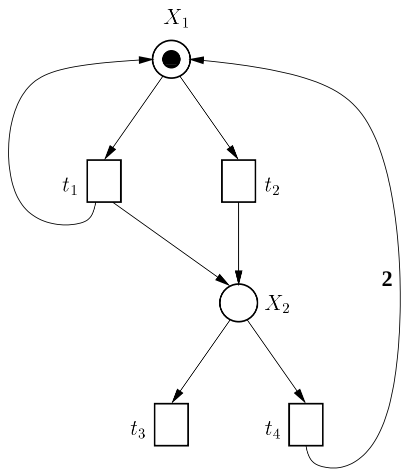


Figure 5.2: The Petri net representation of $\mathcal{E}$

A weighted Petri net $\mathcal{N} = (N, M_0)$ is *communication-free* iff N is communication-free.

Next we present the standard translation from BPP in SNF to communication-free Petri nets:

Definition 5.3.2. Let Δ be a SNF defining system. The *net representation of* Δ, denoted by $net(\Delta)$, is defined by the tuple (S, T, W, l), where

1. $S = Vars(\Delta)$,
2. $T = T_\Delta$,
3. W is such that for all $X \in Vars(\Delta)$, $t \in T_\Delta$ we have:
   (a) $W(X,t) = \begin{cases} 1 & prevar(t) = X, \\ 0 & otherwise, \end{cases}$
   (b) $W(t,X) = |postvars(t)|_X$,
4. $l = l_\Delta$.

Let $\mathcal{E}$ be a SBPP in SNF; w.l.o.g. assume $E_0 = \alpha$ for some $\alpha \in Vars(\mathcal{E})^{\otimes}$. The *Petri net representation of* $\mathcal{E}$, denoted by $PN(\mathcal{E})$, is defined by the pair $(net(\Delta_{\mathcal{E}}), M_0)$, where M_0 is such that $\forall X \in Vars(\mathcal{E}).\ M_0(X) = |\alpha|_X$.

Example 5.3.1. Figure 5.2 gives the Petri net representation of the SBPP $\mathcal{E} = (\Delta, X_1)$, where $\Delta = \{X_1 \stackrel{def}{=} t_1.(X_1 \parallel X_2) + t_2.X_2;\ X_2 \stackrel{def}{=} t_3.\mathbf{0} + t_4.(X_1 \parallel X_1)\}$.

The notion of Petri net unfolding is formalized in [Eng91] for Petri nets without weights; following [EK95] we lift the concept to our setting with weights. One starts off with the definition of partial unfoldings, so-called *branching processes*:

Definition 5.3.3. Let $\mathcal{N} = (N, M_0)$ be a weighted Petri net with $N = (S, T, W, l)$. A *branching process of* $\mathcal{N}$ is a pair $\beta = (N', f)$, where $N' = (B, E, F, l')$ is an occurrence net, and f is a function $f : B \cup E \to S \cup T$ satisfying:

1. $f(B) \subseteq S$, and $f(E) \subseteq T$.
2. For all $t \in T$, $e \in E$ such that $f(e) = t$ we have: $\forall s \in S.\ |{}^\bullet e \cap f^{-1}(s)| = W(s,t)\ \&\ |e^\bullet \cap f^{-1}(s)| = W(t,s)$.
3. For all $e_1, e_2 \in E$ we have: ${}^\bullet e_1 = {}^\bullet e_2\ \&\ f(e_1) = f(e_2) \implies e_1 = e_2$.
4. $\forall s \in S.\ |Min(N') \cap f^{-1}(s)| = M_0(s)$.
5. $l' = l \circ f\restriction_E$.

Let $\beta_1 = (N_1, f_1)$, $\beta_2 = (N_2, f_2)$ be two branching processes of $\mathcal{N}$. An *isomorphism from* β_1 *to* β_2 is an isomorphism h from N_1 to N_2 such that $f_2 \circ h = f_1$ (or equally $f_1 \circ h^{-1} = f_2$). *Branching processes are only considered up to isomorphism.*

The set of branching processes of a Petri net $\mathcal{N}$, denoted by $BP(\mathcal{N})$, is naturally structured by the following partial order: let $\beta_1, \beta_2 \in BP(\mathcal{N})$. $\beta_1 \leq \beta_2$ iff β_1 is an initial part of β_2, and thus unfolds $\mathcal{N}$ to a lesser degree than β_2. For $\mathcal{N}$ without weights it is shown in [Eng91] that $(BP(\mathcal{N}), \leq)$ forms a complete lattice. The result naturally carries over to our setting with weights. Then, each (weighted) Petri net $\mathcal{N}$ has a largest branching process, the *unfolding of* $\mathcal{N}$. In the following, we will denote it by $unf(\mathcal{N})$.

Altogether, we are now ready to define the unfolding of SBPP in SNF:

Definition 5.3.4. Let $\mathcal{E}$ be a SBPP in SNF. The *unfolding of* $\mathcal{E}$, denoted by $unf(\mathcal{E})$, is defined by $unf(PN(\mathcal{E}))$.

Example 5.3.2. Figure 5.3 demonstrates the unfolding of $PN(\mathcal{E})$, and thus $\mathcal{E}$, where $\mathcal{E}$ is as given in Example 5.3.1.

For our purpose it will be convenient to consider SBPP unfoldings via a concrete representative of the respective isomorphism class, and to view the net part as a Petri net:

Convention 5.3.1. Let $\mathcal{E}$ be a SBPP in SNF. We shall view $unf(\mathcal{E})$ as a pair $((N, Min(N)), f)$ such that (N, f) is a concrete representative of $unf(\mathcal{E})$.

(This convention makes sense since isomorphic net systems are always (h)hp bisimilar, and because isomorphism classes of concrete branching processes (N, f) extend to isomorphism classes of Petri nets $(N, Min(N))$).

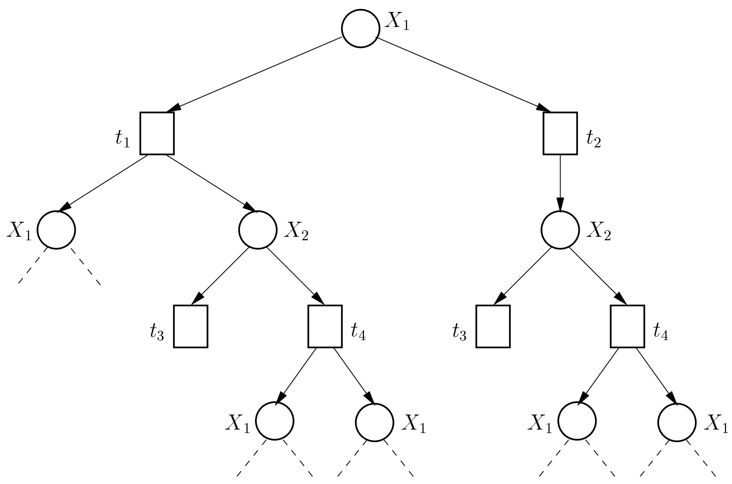


Figure 5.3: The unfolding of $PN(\mathcal{E})$, and thus $\mathcal{E}$

It is easy to see that the Petri net of an unfolding is actually 1-safe. Hence, our partial order notions carry over, and we are finally in a position to define (h)hp bisimilarity for SBPP:

Definition 5.3.5. Two SBPP in SNF $\mathcal{E}$ and $\mathcal{F}$ are (h)hp bisimilar iff $\mathit{proj}_1(\mathit{unf}(\mathcal{E}))$ and $\mathit{proj}_1(\mathit{unf}(\mathcal{F}))$ are (h)hp bisimilar.

In Section 4.4.1 we have defined 'communication-free' for net systems. It is easy to see that:

Proposition 5.3.1. *Let $\mathcal{E}$ be a SBPP in SNF, and $\mathcal{N} = \mathit{proj}_1(\mathit{unf}(\mathcal{E}))$. $\mathcal{N}$ is a comm-free net system, and hence SW-free.*

Proof. Obvious. □

5.3.2 Decidability and Coincidence of hp and hhp Bisimilarity

We now establish the decidability and coincidence of hp and hhp bisimilarity for SBPP. Similarly to our result on bounded-degree SW-free systems (cf. Section 4.4.3) the composition and decomposition theorems of Section 4.3 will deliver the crucial insight behind our proof; this time they will help us to develop a tableau-based procedure that decides hp and hhp bisimilarity at the same time.

First of all, we transfer the composition and decomposition results to SBPP. We proceed via comm-free occurrence net systems: the lats of a comm-free occurrence net system is a SW-free occurrence lats, and with Theorem 4.4.2 and Fact 4.4.1(3) it is easy to see that its prime components must be IS. This is captured at the level of net systems as follows: each 'active' place of a comm-free occurrence net system defines an initially sequential component. Moreover, since we do not allow synchronization these components will never communicate with each other but run completely independently. Formally, we define and formulate:

Definition 5.3.6. Let $\mathcal{N} = (N, M_0)$ be a comm-free occurrence net system. We define the *active places of* $\mathcal{N}$ as $APlaces(\mathcal{N}) = \{p \mid p \in M_0 \ \&\ p^\bullet \neq \emptyset\}$, and the *components of* $\mathcal{N}$ as $Comps(\mathcal{N}) = \{(N, \{p\}) \mid p \in APlaces(\mathcal{N})\}$.

Proposition 5.3.2. *Let $\mathcal{N}$ be a comm-free occurrence net system. We have: $lats(Comps(\mathcal{N})) \cong PComps(lats(\mathcal{N}))$, and every $c \in PComps(lats(\mathcal{N}))$ is IS.*

Proof. This is easy to see with the above explanation and the definition of comm-free. □

Now, we can transfer the composition and decomposition results to comm-free net systems; we require decomposition for hp bisimilarity, and composition for hhp bisimilarity, or to be precise for hhp bisimilarity approximations (this strengthening is naturally also valid). Since IS systems are bsc the decomposition result does apply.

Lemma 5.3.1. *Let $\mathcal{N}_1$, $\mathcal{N}_2$ be two comm-free occurrence net systems.*

1. *Whenever we have $\mathcal{N}_1 \sim_{hp} \mathcal{N}_2$ then there exists a bijection $b : Comps(\mathcal{N}_1) \to Comps(\mathcal{N}_2)$ such that $\forall c_1 \in Comps(\mathcal{N}_1).\ c_1 \sim_{hp} b(c_1)$.*
2. *Whenever there exists a bijection $b : Comps(\mathcal{N}_1) \to Comps(\mathcal{N}_2)$ such that $\forall c_1 \in Comps(\mathcal{N}_1).\ c_1 \sim^n_{hhp} b(c_1)$ then we have $\mathcal{N}_1 \sim^n_{hhp} \mathcal{N}_2$.*

Proof. (1.) is immediate with Prop. 5.3.2 and Theorem 4.3.2(1). (2.) follows similarly from Prop. 5.3.2 and the analogue of Theorem 4.3.1(2) for approximations (which naturally also holds). □

With the help of an observation on SBPP unfoldings, it is then straightforward to further transfer these insights to SBPP.

Definition 5.3.7. Let $\mathcal{E}$ be a SBPP in SNF with $E_0 = \alpha$. We define the *active variables of* $\mathcal{E}$ as $AVars(\mathcal{E}) = \{X \mid X \in \alpha \ \&\ E \neq \mathbf{0}$, where E is such that $(X \stackrel{def}{=} E) \in \Delta_{\mathcal{E}}\}$.

Proposition 5.3.3. *Let $\mathcal{E}$ be a SBPP in SNF with $E_0 = \alpha$, and $unf(\mathcal{E}) = ((N, M_0), f)$. Then there is a bijection $b : APlaces(\mathcal{N}) \to AVars(\mathcal{E})$ such that $\forall (p, X) \in b.\ X = f(p) \ \&\ proj_1(unf(X)) \stackrel{occ}{=} (N, \{p\})$.*

Proof. Easy to read from the definitions. □

Lemma 5.3.2. *Let $\mathcal{E}$, $\mathcal{F}$ be two SBPP with $E_0 = \alpha$, $F_0 = \beta$.*

1. *If we have $\mathcal{E} \sim_{hp} \mathcal{F}$ then there exists a bijection $b : AVars(\mathcal{E}) \to AVars(\mathcal{F})$ such that $\forall X \in AVars(\mathcal{E}).\ X \sim_{hp} b(X)$.*
2. *If there exists a bijection $b : AVars(\mathcal{E}) \to AVars(\mathcal{F})$ such that we have $\forall X \in AVars(\mathcal{E}).\ X \sim^n_{hhp} b(X)$ then $\mathcal{E} \sim^n_{hhp} \mathcal{F}$.*

Proof. Immediate with Lemma 5.3.1 and Prop. 5.3.3. □

This will provide the means of decomposing a goal of two processes to check into subgoals of "smaller" processes to test. Note however, that the lemma will not take us any further if the processes consist of one non-empty IS component each, or, in other words, if the processes correspond to sums. For this case we need a further insight about comm-free net systems.

Lemma 5.3.3. *Let $\mathcal{N}_1$, $\mathcal{N}_2$ be two comm-free net systems such that $|M^0_1| = 1$, $|M^0_2| = 1$.*

1. *If $\mathcal{N}_1 \sim_{hp} \mathcal{N}_2$ then the following two conditions hold:*
   (a) *Whenever $M^0_1 \xrightarrow{t_1} M_1$ for some t_1, M_1, then there exist t_2, M_2 such that $l_1(t_1) = l_2(t_2)$, $M^0_2 \xrightarrow{t_2} M_2$, and $M_1 \sim_{hp} M_2$.*
   (b) *Vice versa.*
2. *If the following two conditions hold then $\mathcal{N}_1 \sim^{n+1}_{hhp} \mathcal{N}_2$:*
   (a) *Whenever $M^0_1 \xrightarrow{t_1} M_1$ for some t_1, M_1, then there exist t_2, M_2 such that $l_1(t_1) = l_2(t_2)$, $M^0_2 \xrightarrow{t_2} M_2$, and $M_1 \sim^n_{hhp} M_2$.*
   (b) *Vice versa.*

Proof. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be given as above. (1.) Clearly holds for any two hp bisimilar net systems.

(2.) Assume we are given label-preserving functions $f : En(\mathcal{N}_1) \to En(\mathcal{N}_2)$, $g : En(\mathcal{N}_2) \to En(\mathcal{N}_1)$, and a family of hhp bisimulation approximations of degree n, $\{\mathcal{H}_{(t_1,t_2)}\}_{(t_1,t_2)\in f\cup g}$, such that for each $(t_1, t_2) \in f \cup g$, $\mathcal{H}_{(t_1,t_2)}$ relates $M^{t_1}_1$ and $M^{t_2}_2$, where $M^0_1 \xrightarrow{t_1} M^{t_1}_1$, and $M^0_2 \xrightarrow{t_2} M^{t_2}_2$. The existence of these entities is guaranteed by the assumption of (2.). We define a relation $\mathcal{H}$ as follows:

$$\mathcal{H} = \{\varepsilon\} \cup \{(t_1, t_2).r \mid (t_1, t_2) \in g \cup f \ \&\ r \in \mathcal{H}_{(t_1,t_2)}\}.$$

It is easy to check that $\mathcal{H}$ is a hhp bisimulation approximation of degree $n + 1$ relating $\mathcal{N}_1$ and $\mathcal{N}_2$. In particular, consider the following two facts: (a) all enabled transitions of $\mathcal{N}_1$, and $\mathcal{N}_2$ respectively, are in conflict with each other; (b) the remaining behaviour of $\mathcal{N}_1$ after the occurrence of a transition t_1 is dependent on the event corresponding to t_1, and similarly for $\mathcal{N}_2$. $\square$

With the help of a further observation on SBPP unfoldings we carry this insight over to SBPP.

Proposition 5.3.4. *Let $\mathcal{E}$ be a SBPP in SNF with $E_0 = \sum_{i=1}^{n} t_i.\alpha_i$, and $unf(\mathcal{E}) = ((N, M_0), f)$.*

1. *For all $e \in E_N$ such that $M_0[e\rangle$ there is $i \in [1, n]$ so that $t_i = f(e)$, and $proj_1(unf(\alpha_i)) \overset{occ}{=} (N, M)$, where $M_0 \overset{e}{\to} M$.*
2. *For all $i \in [1, n]$ there is $e \in E_N$ such that $f(e) = t_i$, $M_0[e\rangle$, and $(N, M) \overset{occ}{=} proj_1(unf(\alpha_i))$, where $M_0 \overset{e}{\to} M$.*

Proof. Easy to read from the definitions. $\square$

Lemma 5.3.4. *Let $\mathcal{E}$, $\mathcal{F}$ be two SBPP with $E_0 = \sum_{i=1}^{n} t_i.\alpha_i$, $F_0 = \sum_{j=1}^{m} u_j.\beta_j$.*

1. *If $\mathcal{E} \sim_{hp} \mathcal{F}$ then the following two conditions hold:*
   (a) *For each $i \in [1, n]$ there is $j \in [1, m]$ such that $l_{\mathcal{E}}(t_i) = l_{\mathcal{F}}(u_j)$, and $\alpha_i \sim_{hp} \beta_j$.*
   (b) *For each $j \in [1, m]$ there is $i \in [1, n]$ such that $l_{\mathcal{E}}(t_i) = l_{\mathcal{F}}(u_j)$, and $\alpha_i \sim_{hp} \beta_j$.*
2. *If the following two conditions hold then $\mathcal{E} \sim_{hhp}^{n+1} \mathcal{F}$:*
   (a) *For each $i \in [1, n]$ there is $j \in [1, m]$ such that $l_{\mathcal{E}}(t_i) = l_{\mathcal{F}}(u_j)$, and $\alpha_i \sim_{hhp}^{n} \beta_j$.*
   (b) *For each $j \in [1, m]$ there is $i \in [1, n]$ such that $l_{\mathcal{E}}(t_i) = l_{\mathcal{F}}(u_j)$, and $\alpha_i \sim_{hhp}^{n} \beta_j$.*

Proof. Immediate with Lemma 5.3.3 and Prop. 5.3.4. $\square$

The Tableau System. We translate these insights directly into a tableau system. The rules can be found in Figure 5.4. Note how rule **Decomp** corresponds to Lemma 5.3.2, and rule **Match** to Lemma 5.3.4. W.l.o.g. we assume that a tableau is started with an expression of the form $\alpha = \beta$. Then the typical order of rule instantiations will be: **Decomp**, **Rec**, **Match**, **Decomp**, $\ldots$; it will always be the case that the subgoals of a rule instantiation match the premise of

Rec $\dfrac{X = Y}{E = F} \quad (X \stackrel{def}{=} E) \in \Delta_{\mathcal{E}}, (Y \stackrel{def}{=} F) \in \Delta_{\mathcal{F}}$

Match $\dfrac{\sum_{i=1}^{n} t_i.\alpha_i = \sum_{j=1}^{m} u_j.\beta_j}{\{\alpha_i = \beta_{f(i)}\}_{i=1}^{n} \quad \{\alpha_{g(j)} = \beta_j\}_{j=1}^{m}}$

where $f : \{1, \ldots, n\} \to \{1, \ldots, m\}, g : \{1, \ldots, m\} \to \{1, \ldots, n\}$ are functions such that $\forall i \in [1, n].\ l_{\mathcal{E}}(t_i) = l_{\mathcal{F}}(u_{f(i)})$, and similarly for g.

Decomp $\dfrac{\alpha = \beta}{\{X = Y\}_{(X,Y) \in b}}$

where $b : AVars(\alpha) \to AVars(\beta)$ is a bijection.

Figure 5.4: Tableau rules relative to two SBPP in SNF $\mathcal{E}$, $\mathcal{F}$

the rule next in this sequence. We proceed in this manner until we hit one of the following terminal nodes.

A node n : *label* is a *successful terminal* iff one of the following conditions holds:

1. *label* = "$\mathbf{0} = \mathbf{0}$".

2. *label* = "$X = Y$", and there is an ancestor node n_a above n in the tableau such that n_a is labelled with "$X = Y$" as well.

A node n : *label* is an *unsuccessful terminal* iff one of the following conditions holds:

3. *label* = "$\alpha = \beta$", and a bijection b as required by rule **Decomp** does not exist.

4. *label* = "$\sum_{i=1}^{n} t_i.\alpha_i = \sum_{j=1}^{m} u_j.\beta_j$", and a pair of functions f and g as required by rule **Match** does not exist.

Clearly, it is easy to check whether a node is a terminal condition. Note how condition (2.) makes sure we 'loop back' whenever we encounter a pair of variables that we have already dealt with before. This will ensure finiteness of the tableau system. Completeness for hp bisimilarity and soundness for hhp bisimilarity will then follow with the first, and respectively second, part of Lemma 5.3.2 and 5.3.4.

Lemma 5.3.5 (finiteness). *Every tableau for two given SBPP in SNF is finite. Furthermore, for two given SBPP in SNF the number of possible tableaux is finite.*

Proof. Let $\mathcal{E}$ with $\Delta_{\mathcal{E}} = \{X_i \overset{def}{=} E_i : i = 1, 2, \ldots, n\}$, $\mathcal{F}$ with $\Delta_{\mathcal{F}} = \{Y_j \overset{def}{=} F_j : j = 1, 2, \ldots, m\}$ be two given SBPP in SNF. Assume to the contrary an infinite tableau $T(E_0 = F_0)$. Since we consider only guarded SBPP any tableau will be finite-branching. Then, König's Lemma applies, and we can assume an infinite path π through the tableau. It is easy to see that any infinite path must contain an infinite number of instantiations of rule **Rec**. But this immediately leads to a contradiction. There are only n variables in $\mathcal{E}$ and m variables in $\mathcal{F}$. Thus, we only have $m \times n$ different nodes of the form $X = Y$ at our disposal, and after at most $m \times n$ instances of **Rec** we will hit a terminal node by the second condition for successful terminals.

This observation also establishes an upper bound on every tableau for given $\mathcal{E}$ and $\mathcal{F}$. So clearly, there can be only finitely many different tableaux for two SBPP. $\square$

Lemma 5.3.6 (completeness for hp bisimilarity). *Let $\mathcal{E}$ and $\mathcal{F}$ be two SBPP in SNF. If $\mathcal{E} \sim_{hp} \mathcal{F}$ then there exists a successful tableau $T(E_0 = F_0)$.*

Proof. Assume we are given two SBPP in SNF $\mathcal{E}$ and $\mathcal{F}$ such that $\mathcal{E} \sim_{hp} \mathcal{F}$. We shall show there exists a successful tableau $T(E_0 = F_0)$.

The tableau rules are forward sound in the following sense: if we apply a rule to a pair of hp bisimilar expressions, we can always find a rule instantiation such that the expressions related by the subgoals of the rule are hp bisimilar as well. This is obvious for rule **Rec**, and follows for **Decomp** from Lemma 5.3.2(1), and for **Match** from Lemma 5.3.4(1).

Thus, starting from the root we can build a tableau such that every node relates two expressions that are hp bisimilar. Since every tableau is finite, this construction will surely terminate. It follows from Lemma 5.3.2(1) and 5.3.4(2) that two expressions that are related by unsuccessful terminal nodes cannot be hp bisimilar, and so each terminal node will be successful. Hence, we have proved that there indeed exists a successful tableau. $\square$

Lemma 5.3.7 (soundness for hhp bisimilarity). *Let $\mathcal{E}$ and $\mathcal{F}$ be two SBPP in SNF. If there is a successful tableau $T(E_0 = F_0)$ then $\mathcal{E} \sim_{hhp} \mathcal{F}$.*

Proof. Let $\mathcal{E}, \mathcal{F}$ be two SBPP in SNF. To the contrary assume there is a successful tableau $T(E_0 = F_0)$, but $\mathcal{E} \not\sim_{hhp} \mathcal{F}$. We shall show that this assumption leads to a contradiction.

If $\mathcal{E} \not\sim_{hhp} \mathcal{F}$ then by Lemma 2.5.2 there is a least k such that $\mathcal{E} \sim^n_{hhp} \mathcal{F}$ for all $n < k$ and $\mathcal{E} \not\sim^n_{hhp} \mathcal{F}$ for all $n \geq k$.

Note that the tableau rules are backwards sound w.r.t. $\sim^n_{hhp}$: if we have a rule instantiation such that the related expressions of each subgoal are hhp bisimilar of approximation n, then the expressions related by the premise must be hhp bisimilar of approximation n, as well. This is obvious for rule **Rec**, and follows for **Decomp** from Lemma 5.3.2(2). For **Match** we actually have a strengthening: the expressions related by the premise must be hhp bisimilar of approximation $n+1$. This is a consequence of Lemma 5.3.4(2).

Thus, in our assumed tableau we can trace a path π such that $E \not\sim_{(k)hhp} F$ for the related expressions E and F of each node. While tracing this path we can mark each node with the least l such that $E \sim^n_{hhp} F$ for all $n < l$, and $E \not\sim^n_{hhp} F$ for all $n \geq l$. Note that the sequence of these measures along π is strictly decreasing due to instantiations of **Match**.

By finiteness (Lemma 5.3.5) π will end in a terminal node n_t. Since the tableau is successful n_t must be a successful terminal, i.e. it is labelled by "$\mathbf{0} = \mathbf{0}$", or by "$X = Y$" and we have an ancestor node n_a labelled by "$X = Y$" as well. The first case cannot be possible since clearly $\mathbf{0} \sim_{hhp} \mathbf{0}$. So let us consider the second case. Let k_{n_t} be the measure of n_t, and k_{n_a} the measure of n_a respectively. Observe that there must be an instantiation of **Match** between n_a and n_t on our path π, and hence we have $k_{n_t} < k_{n_a}$. But this is clearly a contradiction. □

Completeness for hp bisimilarity implies completeness for hhp bisimilarity, and similarly soundness for hhp bisimilarity gives soundness for hp bisimilarity. Then the decidability of hp and hhp bisimilarity is straightforward: we only have to check whether there exists a successful tableau; by finiteness this can easily be done by exhaustive search. Since we decide the two bisimilarities by the same decision procedure it also follows that they coincide.

Theorem 5.3.1.

1. *Two SBPP are hp bisimilar iff they are hhp bisimilar.*

2. *It is decidable whether two SBPP are hp or hhp bisimilar.*

5.4 Basic Parallel Processes

5.4.1 Partial Order Semantics

Similarly to SBPP, we translate each BPP into an occurrence net that gives the unfolding of the BPP. However, since there is no suitable Petri net representation for BPP we cannot employ the mechanism of [Eng91]; instead we shall develop a direct notion of unfolding for BPP. We proceed as follows: first, we show how each variable of a BPP can be represented by a net fragment. Based on this, we develop a concept of branching processes for BPP. The unfolding of a BPP will then be given as the largest branching process with respect to a natural partial order. As justified in Section 5.2.2, we assume BPP to be in ENF.

BPP Net Fragments. It is straightforward to give a net fragment that intuitively represents a given ENF expression E: take a set of net transitions to represent the transitions of E, and equip them with preplaces in a way that reflects how the corresponding base expressions are nested in E by the operators choice and parallel merge. A variable of an ENF defining system is then naturally represented by the net fragment of its defining expression. Moreover, we will embed the name of the variable into the place identifiers of the net fragment.

For technical simplicity, net fragments will be defined as *unlabelled* nets. In the definition we employ the following two operations:

Definition 5.4.1. Let N_1 and N_2 be two unlabelled nets with $T_1 \cap T_2 = \emptyset$, and $F_i \subseteq S_i \times T_i$ for $i \in \{1, 2\}$.

The *parallel composition of* N_1 *and* N_2 is defined by:

$$N_1 \mid\mid N_2 \stackrel{def}{=} (S_1 \uplus S_2, T_1 \cup T_2, \{((p,i),t) \mid i \in \{1,2\} \;\&\; (p,t) \in F_i\}),$$

and the *choice composition of* N_1 *and* N_2 by:

$$\begin{array}{rl} N_1 + N_2 \stackrel{def}{=} & (S_1 \times S_2, T_1 \cup T_2, \\ & \{((p_1,p_2),t) \mid (p_1,p_2) \in S_1 \times S_2 \;\&\; ((p_1,t) \in F_1 \vee (p_2,t) \in F_2)\}). \end{array}$$

Note that the Petri net $(N_1 + N_2, S_{N_1+N_2})$ behaves either like (N_1, S_{N_1}) or like (N_2, S_{N_2}) depending on the choice of the first transition.

Now, we are ready to define:

Definition 5.4.2. Let Δ be an ENF defining system.

Let '$X \stackrel{def}{=} E$' $\in \Delta$. The *net fragment of* X, denoted by $NF(X)$, is given by $netFrag(E)$, where $netFrag$ translates every ENF expression into an unlabelled net; it is inductively defined by:

$$\begin{array}{lcl} netFrag(\mathbf{0}) & = & (\{X\}, \emptyset, \emptyset), \\ netFrag(t.Y) & = & (\{X\}, \{t\}, \{(X,t)\}), \\ netFrag(E_1 \mid\mid E_2) & = & netFrag(E_1) \mid\mid netFrag(E_2), \\ netFrag(E_1 + E_2) & = & netFrag(E_1) + netFrag(E_2). \end{array}$$

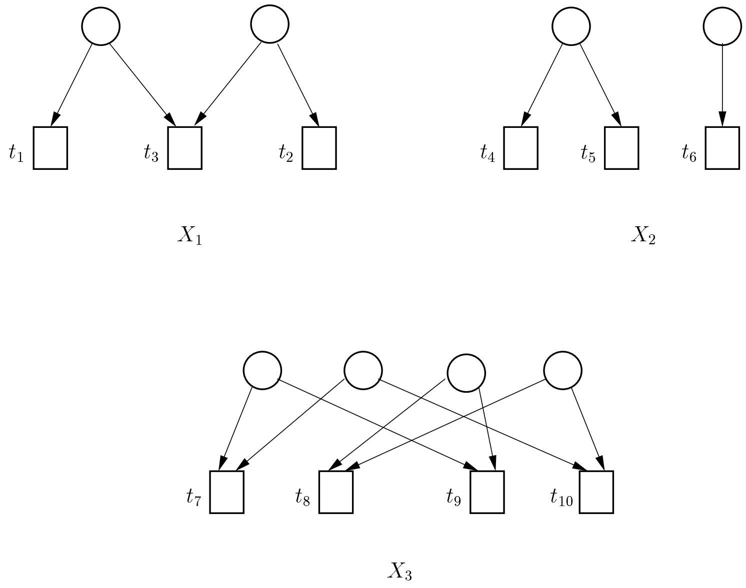


Figure 5.5: The net fragments of Δ

We define the *set of net fragments of* Δ by $\mathit{NFs}(\Delta) = \{\mathit{NF}(X) \mid X \in \mathit{Vars}(\Delta)\}$.

Example 5.4.1. Figure 5.5 shows the net fragments of the ENF defining system $\Delta = \{X_1 \stackrel{def}{=} (t_1.X_2 \mid\mid t_2.X_3) + t_3.\mathbf{0}; X_2 \stackrel{def}{=} (t_4.X_1 + t_5.\mathbf{0}) \mid\mid t_6.X_2; X_3 \stackrel{def}{=} (t_7.X_1 \mid\mid t_8.\mathbf{0}) + (t_9.\mathbf{0} \mid\mid t_{10}.X_3)\}$.

BPP Branching Processes. It seems natural to interconnect the net fragments of a BPP in the following way: simply add arrows that connect each transition t to the places of the net fragment representing the post-variable of t. The resulting net, equipped with an appropriate marking, does, however, not give a satisfying representation of the BPP: tokens generated by parallel occurrences of transitions might jointly enable a transition; such 'cross-synchronization' is clearly contrary to the intuition.

Based on the net fragments we shall instead develop an intuitive notion of partial unfolding for BPP. We first define *canonical* branching processes: their elements are labelled in a special way that keeps track of the history of transition firings; this makes it easy to avoid any 'cross-synchronization'. BPP branching processes are then defined as the isomorphism classes of the canonical representatives.

We start with the definition of the domains from which the elements of canonical branching processes will be picked; after that comes the actual definition.

Definition 5.4.3. Let Δ be an ENF defining system.

We define the *set of places of all net fragments of* Δ by $S_\Delta = \bigcup_{N \in NFs(\Delta)} S_N$. Note that for each $p \in S_\Delta$ there exists exactly one $X \in Vars(\Delta)$ such that $p \in NF(X)$; we denote this variable by $Var(p)$.

Let *init* be a designated 'initial transition' with $init \notin T_\Delta$. We define $ICAN$ to be the smallest set satisfying:

1. $init \in ICAN$, and
2. if $t \in T_\Delta$ and $r \in ICAN$ then $(t, r) \in ICAN$.

Further, we define $CAN_T = T_\Delta \times ICAN$ ($\subset ICAN$), and $CAN_S = S_\Delta \times ICAN$.

Definition 5.4.4. Let $\mathcal{E}$ be a BPP in ENF; w.l.o.g. assume $E_0 = X$ for some $X \in Vars(\mathcal{E})$.

A *canonical branching process of* $\mathcal{E}$ is a pair $\beta = (N, f)$, where $N = (B, E, F, l)$ is an occurrence net satisfying:

1. $B \subseteq CAN_S(\mathcal{E})$, and $E \subseteq CAN_T(\mathcal{E})$,
2. (a) $\{(p, init) \mid p \in S_{NF(X)}\} \subseteq B$,
   (b) $(t, r) \in E \implies \{(p, r) \mid p \in pres(NF(prevar(t)), t)\} \subseteq B$,
   (c) $(p, (t, r)) \in B \implies (t, r) \in E$,
3. F is such that
   (a) $\forall (p, init) \in B.\ {}^\bullet(p, init) = \emptyset$,
   (b) $\forall (p, (t, r)) \in B.\ {}^\bullet(p, (t, r)) = \{(t, r)\}$,
   (c) $\forall (t, r) \in B.\ {}^\bullet(t, r) = \{(p, r) \mid p \in pres(NF(prevar(t)), t)\}$,
4. $\forall (t, r) \in E.\ l(t, r) = l_{\mathcal{E}}(t)$,

and f is a function $f : B \cup E \to S_{\mathcal{E}} \cup T_{\mathcal{E}}$ such that $\forall (p, r) \in B.\ f(p, r) = p$ and $\forall (t, r) \in E.\ f(t, r) = t$.

A *branching process of* $\mathcal{E}$ is a pair $\beta = (N, f)$, where N is an occurrence net and f a function $f : B_N \cup E_N \to S_{\mathcal{E}} \cup T_{\mathcal{E}}$ such that there is a canonical branching process $\beta' = (N', f')$ of $\mathcal{E}$ and an isomorphism h from N to N' satisfying $f = f' \circ h$.

Let $\beta_1 = (N_1, f_1)$, $\beta_2 = (N_2, f_2)$ be two branching processes of $\mathcal{E}$. An *isomorphism from* β_1 *to* β_2 is an isomorphism h from N_1 to N_2 such that $f_2 \circ h = f_1$. *Naturally, we consider branching processes only up to isomorphism.*

We denote the *set of branching processes of* $\mathcal{E}$ by $BP(\mathcal{E})$.

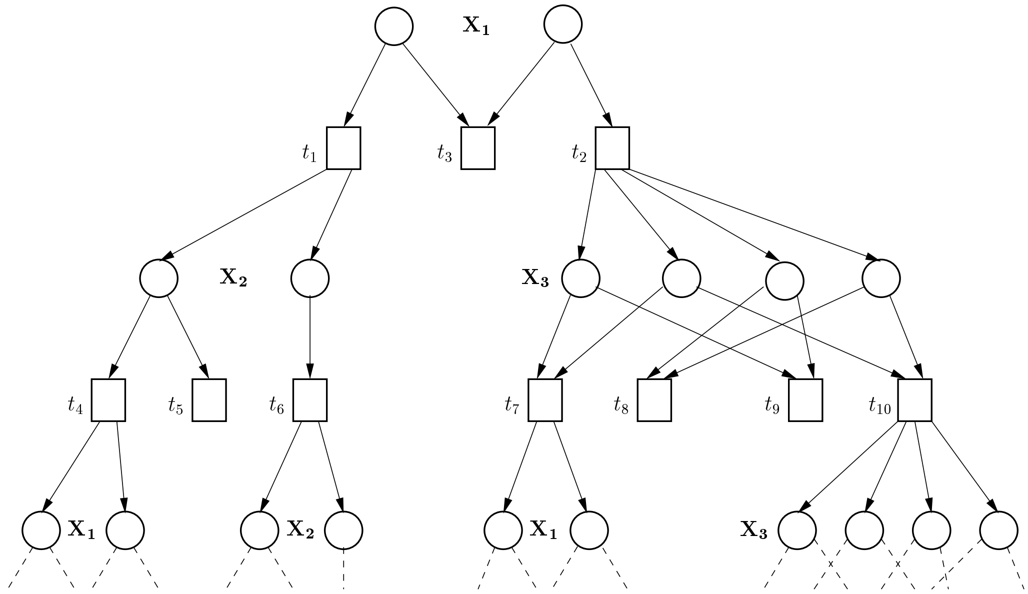


Figure 5.6: The unfolding of $\mathcal{E}$

BPP Unfoldings. Analogously to Petri net branching processes we can structure the branching processes of a BPP $\mathcal{E}$ by a corresponding partial order $\leq$, and then show that $BP(\mathcal{E})$ forms a complete lattice with respect to $\leq$. The latter ensures that each BPP $\mathcal{E}$ has a largest branching process, the *unfolding of* $\mathcal{E}$.

Definition 5.4.5. Let $\mathcal{E}$ be a BPP in ENF. We define the *unfolding of* $\mathcal{E}$, denoted by $\mathit{unf}(\mathcal{E})$, to be the largest element of $(BP(\mathcal{E}), \leq)$.

Example 5.4.2. Figure 5.6 demonstrates the unfolding of $\mathcal{E} = (\Delta, X_1)$, where Δ is as defined in Example 5.4.1.

We adopt a convention analogous to Convention 5.3.1, and view BPP unfoldings as concrete pairs $(\mathcal{N}, f)$, where $\mathcal{N}$ is a Petri net. As can easily be checked Petri nets of BPP unfoldings are in fact 1-safe. Then, (h)hp bisimilarity for BPP is given by:

Definition 5.4.6. Two BPP in ENF $\mathcal{E}$ and $\mathcal{F}$ are (h)hp bisimilar iff $\mathit{proj}_1(\mathit{unf}(\mathcal{E}))$ and $\mathit{proj}_1(\mathit{unf}(\mathcal{F}))$ are (h)hp bisimilar.

Note that for SBPP the semantics given here coincides with the one defined in Section 5.3.1.

BPP Unfoldings are Proper-comm Free Net Systems. The nets of BPP unfoldings are certainly not comm-free, but they can be characterized by a generalization of this net type. Note that in BPP unfoldings communication is only employed to implement conflict between two net components with parallel elements; this makes the occurrence of communication very restricted: only places with identical sets of pre-transitions, might share a post-transition. Dynamically, this means that at every reachable marking for any transition t either none of the pre-places of t hold, or all of them do. Thus, in these net systems there is no synchronization in the sense of 'uniting two independent flows of tokens', since then one would expect that the pre-places of a corresponding synchronization transition can hold separately. This is of course intuitive for the semantics of a BPP. Excluding such synchronization naturally has implications for the behaviour: the corresponding net systems are SW-1 and SW-2 free, but note that they may contain SW-3. Formally, these explanations amount to:

Definition 5.4.7. A net N is *proper-communication free* (short: *proper-comm free*) iff for all $p_1, p_2 \in S_N$ we have: ${p_1}^\bullet \cap {p_2}^\bullet \neq \emptyset \implies {}^\bullet p_1 = {}^\bullet p_2$. A net system $\mathcal{N} = (N, M_0)$ is *proper-comm free* iff N is proper-comm free.

Proposition 5.4.1. *Proper-comm free net systems are SW-$\{1,2\}$ free.*

Proof. This is easy to check with the above description. ☐

Proposition 5.4.2. *Let $\mathcal{E}$ be a BPP in ENF, and $\mathcal{N} = proj_1(unf(\mathcal{E}))$. $\mathcal{N}$ is a proper-comm free net system, and hence SW-$\{1,2\}$ free.*

Proof. This is also obvious. ☐

5.4.2 Non-coincidence of hp and hhp Bisimilarity

It is immediate that hp and hhp bisimilarity do *not* coincide for the entire BPP class:[2] recall that system A of counter-example 2 (Figure 1.8) can be described by the expression $(a.0 + c.0 \mid\mid b.0) + (a.0 \mid\mid b.0) + (a.0 \mid\mid b.0 + c.0)$, and system B by $(a.0 + c.0 \mid\mid b.0) + (a.0 \mid\mid b.0 + c.0)$; these are certainly BPP processes, and it is easy to verify that the lats characterization agrees with our partial order semantics.

Theorem 5.4.1. *hp and hhp bisimilarity do not coincide for BPP in general.*

5.4.3 Decidability of hp Bisimilarity

In this section we show that hp bisimilarity is decidable for BPP.

[2]Many thanks to Rob van Glabbeek for pointing this out to me.

$$\frac{}{t.E \xrightarrow{t} (E, \mathbf{0})} \qquad \frac{E \xrightarrow{t} (E_l, E_c)}{X \xrightarrow{t} (E_l, E_c)} \; (X \stackrel{def}{=} E) \in \Delta$$

$$\frac{E \xrightarrow{t} (E_l, E_c)}{E + F \xrightarrow{t} (E_l, E_c)} \qquad \frac{F \xrightarrow{t} (F_l, F_c)}{E + F \xrightarrow{t} (F_l, F_c)}$$

$$\frac{E \xrightarrow{t} (E_l, E_c)}{E \mid\mid F \xrightarrow{t} (E_l, E_c \mid\mid F)} \qquad \frac{F \xrightarrow{t} (F_l, F_c)}{E \mid\mid F \xrightarrow{t} (F_l, E \mid\mid F_c)}$$

Figure 5.7: Distributed transition rules relative to a BPP defining system Δ

Distributed Transition Semantics. We begin with the introduction of a concept that is crucial for the proof. In [Cas88] Castellani introduces a non-interleaving semantics for a BPP-like process language. The semantics is based on the principle of distribution: to reflect that concurrent processes are situated at different locations, a transition in Castellani's distributed transition systems leads to a compound residual, consisting of a local residual and a concurrent residual. The local residual describes the remaining behaviour of the locality where the action took place, whereas the concurrent residual represents the unaffected behaviour of the localities that have not been involved in the action performance. The parallel composition of the two residuals constitutes the global remaining behaviour. For BPP these distributed transitions are defined by the SOS rules given in Figure 5.7: for a BPP defining system Δ let $\rightarrow \subseteq BPP(\Delta) \times T_\Delta \times (BPP(\Delta) \times BPP(\Delta))$ be the least relation satisfying the transition rules.

The distributed transition relation is consistent with the standard one in the following way:

Proposition 5.4.3. *Let Δ be a BPP defining system, $E \in BPP(\Delta)$, and $t \in T_\Delta$.*

1. *Let $E' \in BPP(\Delta)$. $E \xrightarrow{t} E' \implies \exists E_l, E_c.\ E \xrightarrow{t} (E_l, E_c)\ \&\ E_l \mid\mid E_c \equiv E'$.*

2. *Let E_l, $E_c \in BPP(\Delta)$. $E \xrightarrow{t} (E_l, E_c) \implies \exists E'.\ E \xrightarrow{t} E'\ \&\ E' \equiv E_l \mid\mid E_c$.*

Proof. Easy by rule induction. □

The following ensures that if a BPP is in ENF then its local and concurrent residuals can also be understood as BPP in ENF:

Proposition 5.4.4. *Let Δ be an ENF defining system, $E \in ENF(\Delta) \cup Vars(\Delta)$, $t \in T_\Delta$, and $E_l, E_c \in BPP(\Delta)$ such that $E \xrightarrow{t} (E_l, E_c)$. Then we have $E_l \in Vars(\Delta)$, and $E_c \in ENF(\Delta)$.*

Proof. Easy by rule induction. $\square$

It will be useful to have a characterization of local and concurrent residuals of ENF expressions in terms of functions. This is possible due to our restriction to transition-genuine expressions.

Definition 5.4.8. Let E be an ENF expression, and $t \in T(E)$.

We define the *local residual of E after the occurrence of t* by $locR(E,t) = X$, where $t.X$ is a base expression of E.

The *concurrent residual of E after the occurrence of t* is inductively defined by:

$$\begin{array}{lcl} conR(t.X, t) & = & \mathbf{0}, \\ conR(E + F, t) & = & \textbf{if } t \in T(E) \textbf{ then } conR(E,t) \\ & & \textbf{else } conR(F,t), \\ conR(E \mid\mid F, t) & = & \textbf{if } t \in T(E) \textbf{ then } conR(E,t) \mid\mid F \\ & & \textbf{else } E \mid\mid conR(F,t). \end{array}$$

The following is obvious:

Proposition 5.4.5. *Let Δ be an ENF defining system, $E \in ENF(\Delta)$, and $t \in T(E)$. We have $locR(E,t) \in Vars(\Delta)$, and $conR(E,t) \in ENF(\Delta)$.*

Our functions capture local and concurrent residuals as follows:

Proposition 5.4.6. *Let Δ be an ENF defining system, $E, E_c \in ENF(\Delta)$, $E_l \in Vars(\Delta)$, and $t \in T_\Delta$.*

$$E \xrightarrow{t} (E_l, E_c) \iff t \in T(E), E_l = locR(E,t) \;\&\; E_c = conR(E,t).$$

Proof. Easy with Prop. 5.4.3 and 5.2.1 by induction on the structure of E. $\square$

Partial Order Split. In proper-comm free net systems we can split the system behaviour that remains after the execution of a transition t into two parallel components just as well. Then one component describes the remaining behaviour that is dependent on t, whereas the other one stands for the remaining behaviour independent of t. We call these components the *dependent* and *independent remainder* of the net system. Formally, we define:

Definition 5.4.9. Let N be a proper-comm free net, M a safe marking of N, and $t \in T_N$ with $M[t\rangle$.

We define the *dependent remainder of M after the firing of t* by $depR(M,t) = t^\bullet$, and the *independent remainder of M after the firing of t* by $indR(M,t) = M' - t^\bullet$, where $M \xrightarrow{t} M'$.

In the following we write $M \xrightarrow{t} (M_d, M_i)$ as a short notation for "$M[t\rangle$, $M_d = depR(M,t)$ & $M_i = indR(M,t)$".

It is immediate from the definition that analogously to Prop. 5.4.3 we have:

Observation 5.4.1. *Let N be a proper-comm free net, M a safe marking of N, and $t \in T_N$.*

1. *Let $M' \subseteq S_N$. $M \xrightarrow{t} M' \implies \exists M_d, M_i.\ M \xrightarrow{t} (M_d, M_i)$ & $M_d \cup M_i = M'$.*
2. *Let M_d, $M_i \subseteq S_N$. $M \xrightarrow{t} (M_d, M_i) \implies \exists M'.\ M \xrightarrow{t} M'$ & $M' = M_d \cup M_i$.*

The following shows that our definition agrees with the intuition behind the concepts 'dependent and independent remainder':

Proposition 5.4.7. *Let N be a proper-comm free net, M a safe marking of N, and t, M_d, M_i, M' such that $M \xrightarrow{t} (M_d, M_i)$, and $M \xrightarrow{t} M'$.*

1. $Runs(M_d) = \{r \mid t.r \in Runs(M)\ \&\ \forall i \in [2, |t.r|].\ 1 <_{t.r} i\}$.
2. $Runs(M_i) = \{r \mid t.r \in Runs(M)\ \&\ \forall i \in [2, |t.r|].\ 1\ co_{t.r}\ i\}$.
3. $Runs(M') = \bigcup\{r_d \otimes r_i \mid r_d \in Runs(M_d)\ \&\ r_i \in Runs(M_i)\}$.

Proof. The '$\subseteq$'-direction of (1), both directions of (2), and '$\supseteq$' of (3) are general consequences of the definition of I_N and *Runs*. For the remaining directions of (1) and (3) additionally employ the proper-comm property. $\square$

Coincidence of the Two Views. Via our partial order semantics the 'partial order split' view carries over to BPP, and — importantly for our proof — it coincides with the distributed semantics in the following sense:

Proposition 5.4.8. *Let $\mathcal{E}$ be a BPP in ENF, and $unf(\mathcal{E}) = ((N, M_0), f)$.*

1. *Let $M_d, M_i \subseteq B_N$, $e \in E_N$.*
   $M_0 \xrightarrow{e} (M_d, M_i) \implies \exists E_l, E_c, t.\ E_0 \xrightarrow{t} (E_l, E_c), t = f(e), proj_1(unf(E_l)) \overset{occ}{=} (N, M_d)\ \&\ proj_1(unf(E_c)) \overset{occ}{=} (N, M_i)$.
2. *Let $E_l \in Vars(\mathcal{E})$, $E_c \in ENF(\Delta_{\mathcal{E}})$, $t \in T_{\mathcal{E}}$.*
   $E_0 \xrightarrow{t} (E_l, E_c) \implies \exists M_d, M_i, e.\ M_0 \xrightarrow{e} (M_d, M_i), f(e) = t, (N, M_d) \overset{occ}{=} proj_1(unf(E_l))\ \&\ (N, M_i) \overset{occ}{=} proj_1(unf(E_c))$.

Proof. This can be read from the definitions. $\square$

Crucial Insights. The 'partial order split' view translates in a natural way into hp bisimilarity: assume two proper-comm free net systems $\mathcal{N}_1$ and $\mathcal{N}_2$. Whenever $\mathcal{N}_1$ and $\mathcal{N}_2$ are hp bisimilar then there is a match for each enabled transition of $\mathcal{N}_1$ by one of $\mathcal{N}_2$ and vice versa such that the resulting pairs of dependent remainders and independent remainders are hp bisimilar. Conversely, whenever we can exhibit a match for each enabled transition of $\mathcal{N}_1$ by one of $\mathcal{N}_2$ and vice versa such that the resulting pairs of dependent remainders and independent remainders are hp bisimilar then $\mathcal{N}_1$ and $\mathcal{N}_2$ are hp bisimilar as well. The latter can in fact be strengthened to a statement about hp bisimulation approximations. Formally, we formulate and prove:

Lemma 5.4.1. *Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two proper-comm free net systems.*

1. *If $\mathcal{N}_1 \sim_{hp} \mathcal{N}_2$ then the following two conditions hold:*
   (a) *Whenever $M_1^0 \xrightarrow{t_1} (M_1^d, M_1^i)$ then there exist t_2, M_2^d, M_2^i such that $l_1(t_1) = l_2(t_2)$, $M_2^0 \xrightarrow{t_2} (M_2^d, M_2^i)$, $M_1^d \sim_{hp} M_2^d$, and $M_1^i \sim_{hp} M_2^i$.*
   (b) *Vice versa.*
2. *If the following two conditions hold then $\mathcal{N}_1 \sim_{hp}^{n+1} \mathcal{N}_2$:*
   (a) *Whenever $M_1^0 \xrightarrow{t_1} (M_1^d, M_1^i)$ then there exist t_2, M_2^d, M_2^i such that $l_1(t_1) = l_2(t_2)$, $M_2^0 \xrightarrow{t_2} (M_2^d, M_2^i)$, $M_1^d \sim_{hp}^n M_2^d$, and $M_1^i \sim_{hp}^n M_2^i$.*
   (b) *Vice versa.*

Proof. Let $\mathcal{N}_1$, $\mathcal{N}_2$ be given as above.

(1.) Assume $\mathcal{H}$ to be a hp bisimulation relating $\mathcal{N}_1$ and $\mathcal{N}_2$. We shall show that property (a) holds; (b) can be proved symmetrically.

Let t_1, M_1^d, M_1^i such that $M_1^0 \xrightarrow{t_1} (M_1^d, M_1^i)$. Clearly, this means $M_1^0 \xrightarrow{t_1} M_1$ for $M_1 = M_1^d \cup M_1^i$. Then, by definition of hp bisimulation we obtain t_2, M_2 such that $M_2^0 \xrightarrow{t_2} M_2$, and $(t_1, t_2) \in \mathcal{H}$. This gives us $M_2^0 \xrightarrow{t_2} (M_2^d, M_2^i)$ for M_2^d, M_2^i with $M_2 = M_2^d \cup M_2^i$. $(t_1, t_2) \in \mathcal{H}$ implies that (t_1, t_2) must be a pair of synchronous runs, which in turn implies $l_1(t_1) = l_2(t_2)$. Further, $(t_1, t_2) \in \mathcal{H}$ means $\mathcal{H}$ must 'hp bisimulate' the behaviour of M_1 and M_2 such that the matching reflects the dependencies to t_1 and t_2 correctly. Naturally, these matchings will cover the behaviour of the sub-markings M_1^d, M_1^i, and M_2^d, M_2^i respectively. With Prop. 5.4.7(1) it is easy to see that any behaviour of M_1^d has to be matched by behaviour of M_1^d, and vice versa. Similarly, it follows from Prop. 5.4.7(2) that any behaviour of M_1^i has to be matched by M_2^i, and also the other way around.

But this amounts to the existence of two hp bisimulations, one relating M_1^d and M_d^2, and the other relating M_1^i and M_2^i.

(2.) Imagine we are given label-preserving functions $f : En(\mathcal{N}_1) \to En(\mathcal{N}_2)$, $g : En(\mathcal{N}_2) \to En(\mathcal{N}_1)$, and two families of hp bisimulation approximations of degree n, $\{\mathcal{H}_{(t_1,t_2)dR}\}_{(t_1,t_2)\in f\cup g}$ and $\{\mathcal{H}_{(t_1,t_2)iR}\}_{(t_1,t_2)\in f\cup g}$, such that for each $(t_1,t_2) \in f \cup g$, $\mathcal{H}_{(t_1,t_2)dR}$ relates $depR(M_1^0, t_1)$ with $depR(M_2^0, t_2)$, and $\mathcal{H}_{(t_1,t_2)iR}$ relates $indR(M_1^0, t_1)$ with $indR(M_2^0, t_2)$. The existence of these entities is guaranteed by the assumption of the lemma.

We shall now construct a hp bisimulation approximation of degree $n+1$ for $\mathcal{N}_1$ and $\mathcal{N}_2$ based on these entities. First, we define for each $(t_1,t_2) \in f \cup g$ a set $\mathcal{H}_{(t_1,t_2)}$ as follows:

$$\mathcal{H}_{(t_1,t_2)} = \bigcup \{r_d \otimes r_i \mid r_d \in \mathcal{H}_{(t_1,t_2)dR} \ \& \ r_i \in \mathcal{H}_{(t_1,t_2)iR}\}.$$

With the help of Prop. 5.4.7(3) it is easy to verify that each $\mathcal{H}_{(t_1,t_2)}$ provides a hp bisimulation approximation of degree n for M_{t_1} and M_{t_2}, where $M_1^0 \xrightarrow{t_1} M_{t_1}$, and $M_2^0 \xrightarrow{t_2} M_{t_2}$.

Then, we define:

$$\mathcal{H} = \{\varepsilon\} \cup \{(t_1,t_2).r \mid (t_1,t_2) \in f \cup g \ \& \ r \in \mathcal{H}_{(t_1,t_2)}\}.$$

With the above and Prop. 5.4.7(1,2) it is clear that $\mathcal{H}$ is a hp bisimulation approximation of degree $n+1$ relating $\mathcal{N}_1$ and $\mathcal{N}_2$. □

By the coincidence result of the previous paragraph it follows that the distributed view translates into hp bisimilarity for BPP in analogous fashion; that is:

Lemma 5.4.2. *Let $\mathcal{E}$ and $\mathcal{F}$ be two BPP in ENF.*

1. *If $\mathcal{E} \sim_{hp} \mathcal{F}$ then the following two conditions hold:*
   - *(a) Whenever $E_0 \xrightarrow{t} (E_l, E_c)$ then there exist u, F_l, F_c such that $l_{\mathcal{E}}(t) = l_{\mathcal{F}}(u)$, $F_0 \xrightarrow{u} (F_l, F_c)$, $E_l \sim_{hp} F_l$, and $E_c \sim_{hp} F_c$.*
   - *(b) Vice versa.*
2. *If the following two conditions hold then $\mathcal{E} \sim_{hp}^{n+1} \mathcal{F}$:*
   - *(a) Whenever $E_0 \xrightarrow{t} (E_l, E_c)$ then there exist u, F_l, F_c such that $l_{\mathcal{E}}(t) = l_{\mathcal{F}}(u)$, $F_0 \xrightarrow{u} (F_l, F_c)$, $E_l \sim_{hp}^n F_l$, and $E_c \sim_{hp}^n F_c$.*
   - *(b) Vice versa.*

Proof. Immediate with Lemma 5.4.1 and Prop. 5.4.8. □

$$\textbf{Rec} \qquad \frac{X = Y}{E = F} \quad (X \stackrel{def}{=} E) \in \Delta_{\mathcal{E}},\ (Y \stackrel{def}{=} F) \in \Delta_{\mathcal{F}}$$

$$\textbf{Match} \quad E = F \Leftarrow \begin{cases} \{locR(E,t) = locR(F,f(t))\}_{t \in T(E)} \\ \{conR(E,t) = conR(F,f(t))\}_{t \in T(E)} \\ \{locR(E,g(u)) = locR(F,u)\}_{u \in T(F)} \\ \{conR(E,g(u)) = conR(F,u)\}_{u \in T(F)} \end{cases}$$

where $E \in ENF(\Delta_{\mathcal{E}})$, $F \in ENF(\Delta_{\mathcal{F}})$, and $f : T(E) \to T(F)$, $g : T(F) \to T(E)$ are functions such that $\forall t \in T(E).\ l_{\mathcal{E}}(t) = l_{\mathcal{F}}(f(t))$, and similarly for g.

Figure 5.8: Tableau rules for hp bisimilarity relative to two BPP in ENF $\mathcal{E}$, $\mathcal{F}$

The Tableau System. With this insight it is straightforward to construct a tableau system that decides hp bisimilarity for BPP (assumed in ENF). We simply translate Lemma 5.4.2 into a tableau rule; it will provide matching and decomposition at the same time. Altogether the tableau consists of the two rules depicted in Figure 5.8.

Note that our rules only cover goals of the form "$X = Y$" or "$E = F$", where E and F are ENF expressions. This is sufficient since we start the tableau with an expression of either form, and our rules only generate subgoals that are again of either form. The latter is obvious for **Rec**, and follows for **Match** from Prop. 5.4.5. We develop the tableau until we hit a node that satisfies one of the following terminal conditions.

A node n : *label* is a *successful terminal* iff one of the following conditions holds:

1. *label* = "$\mathbf{0} = \mathbf{0}$".

2. *label* = "$X = Y$", and there is an ancestor node n_a above n in the tableau such that n_a is labelled with "$X = Y$" as well.

A node n : *label* is an *unsuccessful terminal* iff the following condition holds:

3. *label* = "$E = F$", where E and F are ENF expressions, and a pair of functions f and g as required by rule **Match** does not exist.

As in Section 5.3.2 condition (2.) makes sure we 'loop back' whenever we encounter a pair of variables that we have already dealt with before. Finiteness of the tableau can then be established by using the arguments of Lemma 5.3.5

together with the following observation: there is a uniform bound on how often rule **Match** can be applied consecutively; to see this simply consider that each application of $conR$ 'filters out' one base expression, and hence repeatedly applying $conR$ leads to process $\mathbf{0}$.

With Lemma 5.4.2 and Prop. 5.4.6, forward and strengthened backwards soundness of rule **Match** for hp bisimilarity are immediate; completeness and soundness of the tableau can then be proved by following the proof of Lemma 5.3.6 and 5.3.7. Together with finiteness this establishes the decidability of hp bisimilarity.

Theorem 5.4.2. *It is decidable whether two BPP are hp bisimilar.*

5.4.4 Coincidence of hp and Distributed Bisimilarity

Distributed bisimulation [Cas88, CH89] is the natural notion of bisimulation corresponding to Castellani's distributed transition semantics: it refines classical bisimulation by requiring that local residuals and concurrent residuals are related separately. It is defined as follows:

Definition 5.4.10. Let Δ be a BPP defining system. A relation $\mathcal{D} \subseteq BPP(\Delta) \times BPP(\Delta)$ is a *distributed bisimulation* if for any $(E, F) \in \mathcal{D}$ we have

(i) Whenever $E \xrightarrow{t} (E_l, E_c)$ for some t, E_l, E_c, then there exist u, F_l, F_c such that $l_\Delta(t) = l_\Delta(u)$, $F \xrightarrow{u} (F_l, F_c)$, $(E_l, F_l) \in \mathcal{D}$, and $(E_c, F_c) \in \mathcal{D}$.

(ii) Vice versa.

We say two BPP E, $F \in BPP(\Delta)$ are *distributed bisimilar* iff there is a distributed bisimulation $\mathcal{D}$ with $(E, F) \in \mathcal{D}$.

It follows directly from the definition that the tableau rules for hp bisimilarity are forward and (strengthened) backwards sound for distributed bisimulation. Hence, the tableau provides a decision procedure for distributed bisimilarity just as well, which immediately establishes the coincidence of the two notions for BPP.

Theorem 5.4.3. *Two BPP are hp bisimilar iff they are distributed bisimilar.*

As mentioned earlier, the decidability of distributed bisimilarity for BPP has already been established by Christensen in [Chr92]. This proof also employs the tableau technique, and — not surprisingly — a comparison shows that our tableau is similar to the one exhibited there. The major new ingredient in proving the decidability of hp bisimilarity lies in Lemma 5.4.2, which shows that the distributed view translates into hp bisimilarity.

Otherwise, there are technical differences between Christensen's tableau and the one exhibited above. Christensen makes use of his BPP standard normal form, where every defining expression is of the form $\sum_{i=1}^{n} a_i.\alpha_i \lfloor \beta_i$ such that each α_i, β_i is a parallel composition of variables. The left merge operator $\lfloor$ acts like parallel composition under the constraint that the first action must come from the left process. Due to the use of this normal form the local and concurrent residuals are separated out in the process expressions of Christensen's tableau rules. In contrast, we employ labelled transitions and BPP in ENF, and use our functions *locR* and *conR* to determine local and concurrent residuals.

5.4.5 Decidability of hhp Bisimilarity

Now, we will see that for BPP hhp bisimilarity can be decided by means of a tableau system in a straightforward way. The tableau system will also show that for BPP hhp bisimilarity coincides with its strengthening to chhp bisimilarity.

Concurrent Steps. First of all, let us recall the concepts of concurrent and maximal concurrent step from Section 4.3.1.1. Their definition (Def. 4.3.1) is formulated for lats', and carries over to net systems in the natural way. Rather than considering concurrent steps to be *sequences* of transitions it is intuitive to regard them as *sets* of transitions just as well. Since this will be more convenient here, we define:

Definition 5.4.11. Let $\mathcal{N}$ be a net system, and $M \in \mathit{Reach}(\mathcal{N})$. A set $\gamma \subseteq T_N$ is a *concurrent step* at M iff we have: $M \stackrel{w}{\rightarrow}$ for some w such that $\mathit{set}(w) = \gamma$, and $t \; I_N \; t'$ for any distinct $t, t' \in \gamma$. The concept of *maximal concurrent step*, the expressions $\mathit{csteps}(M)$, and $\mathit{mcsteps}(M)$ are defined analogously to Def. 4.3.1.

For BPP we can exhibit corresponding 'syntactic' concepts; specifically for defined ENF expressions we define:

Definition 5.4.12. Let Δ be an ENF defining system, and $E \in \mathit{ENF}(\Delta)$.

A *step of* E is a set $\sigma \subseteq T(E)$ such that there exists $w \in T_\Delta^*$ with $E \stackrel{w}{\rightarrow}$ and $\mathit{set}(w) = \sigma$. We denote the *set of steps of* E by $\mathit{steps}(E)$.

A step σ of E is *maximal* iff $\forall t \in T(E).\; t \notin \sigma \Rightarrow \sigma \cup \{t\} \notin \mathit{steps}(E)$. We denote the *set of maximal steps of* E by $\mathit{msteps}(E)$.

The (maximal) steps of a defined ENF expression correspond to the (maximal) concurrent steps of its unfolding in the following way:

Proposition 5.4.9. *Let Δ be an ENF defining system, $E_0 \in ENF(\Delta)$, and $unf(E_0) = ((N, M_0), f)$.*

1. (a) $\gamma \in csteps(M_0) \implies f(\gamma) \in steps(E_0)$.

   (b) $\sigma \in steps(E_0) \implies f^{-1}(\sigma) \cap En(M_0) \in csteps(M_0)$.

2. (a) $\gamma \in mcsteps(M_0) \implies f(\gamma) \in msteps(E_0)$.

   (b) $\sigma \in msteps(E_0) \implies f^{-1}(\sigma) \cap En(M_0) \in mcsteps(M_0)$.

Proof. This can be read from the definitions. $\square$

Crucial Insights. The first step towards the decidability proof is to realize that the behaviour of a proper-comm free net system can be expressed in terms of concurrent steps and dependent remainders as follows:

Proposition 5.4.10. *Let $\mathcal{N}$ be a proper-comm free net system. We have:*

$$\begin{aligned} Runs(M_0) = \ & \bigcup \{t_1.r_1 \otimes \cdots \otimes t_n.r_n \mid \\ & \{t_1, \ldots, t_n\} \in csteps(M_0) \ \& \ \forall i \in [1, n].\ r_i \in depR(M_0, t_i)\} \end{aligned}$$

Proof. The '$\supseteq$'-direction is a general consequence of the definition of I_N and *Runs*; the other direction additionally relies on the proper-comm property. $\square$

Crucially, this characterization translates into hhp bisimilarity in the following way: assume two proper-comm free net systems $\mathcal{N}_1$ and $\mathcal{N}_2$. If $\mathcal{N}_1$ and $\mathcal{N}_2$ are hhp bisimilar then there is a match between the concurrent steps of $\mathcal{N}_1$ and $\mathcal{N}_2$ such that this match amounts to a hhp bisimulation, and the resulting pairs of dependent remainders are hhp bisimilar. Conversely, if there is a match between the concurrent steps of $\mathcal{N}_1$ and $\mathcal{N}_2$ such that this match amounts to a hhp bisimulation, and the resulting pairs of dependent remainders are hhp bisimilar, then $\mathcal{N}_1$ and $\mathcal{N}_2$ are hhp bisimilar as well. As one would expect, the latter can be strengthened to a statement about hhp bisimulation approximations.

Note that hp bisimilarity can be decomposed and composed according to Prop. 5.4.10 just as well. It is possible to develop a procedure for deciding hp bisimilarity analogously to what will follow for hhp bisimilarity rather than exploit the distributed view as was done in the previous section.

We now proceed to prove the above insight, or, to be precise, a slight strengthening: it suffices to employ matchings of *maximal* concurrent steps when we ensure a certain continuation property is satisfied. Formally, this amounts to:

1. (a) $\{proj_1(\gamma) \mid \gamma \in R\} = mcsteps(M_1^0)$.
   (b) $\{proj_2(\gamma) \mid \gamma \in R\} = mcsteps(M_2^0)$.
2. $\forall (t_1, t_2) \in \bigcup_{\gamma \in R} \gamma.\ l_1(t_1) = l_2(t_2)$.
3. For all $\gamma \in \mathcal{P}(T_1 \times T_2)$ such that $\exists \gamma' \in R.\ \gamma \subseteq \gamma'$ we have:
   (a) If $proj_1(\gamma) \cup t_1 \in csteps(M_1^0)$ for some $t_1 \in T_1$, then there exist $t_2 \in T_2$, $\gamma'' \in R$ so that $\gamma \cup (t_1, t_2) \subseteq \gamma''$.
   (b) Vice versa.

Figure 5.9: Let $\mathcal{N}_1$, $\mathcal{N}_2$ be two net systems. The figure gives conditions for a set $R \subseteq \mathcal{P}(T_1 \times T_2)$

Lemma 5.4.3. *Let $\mathcal{N}_1$, $\mathcal{N}_2$ be two proper-comm free net systems.*

1. *If $\mathcal{N}_1 \sim_{hhp} \mathcal{N}_2$ then there is a set $R \subseteq \mathcal{P}(T_1 \times T_2)$ such that R satisfies the conditions of Figure 5.9, and $\forall (t_1, t_2) \in \bigcup_{\gamma \in R} \gamma.\ depR(M_1^0, t_1) \sim_{hhp} depR(M_2^0, t_2)$.*

2. *If there is a set $R \subseteq \mathcal{P}(T_1 \times T_2)$ such that R satisfies the conditions of Figure 5.9, and $\forall (t_1, t_2) \in \bigcup_{\gamma \in R} \gamma.\ depR(M_1^0, t_1) \sim_{hhp}^{n} depR(M_2^0, t_2)$ then we have $\mathcal{N}_1 \sim_{hhp}^{n+1} \mathcal{N}_2$.*

Proof. Let $\mathcal{N}_1$, $\mathcal{N}_2$ be given as above.

(1.) Presuppose a hhp bisimulation $\mathcal{H}$ for $\mathcal{N}_1$ and $\mathcal{N}_2$. We define a set R as follows:

$$R = \{set(r) \mid r \in \mathcal{H}\ \&\ proj_i(set(r)) \in mcsteps(M_i^0) \text{ for } i = 1, \text{ or } 2\}$$

It is clear that $R \subseteq \mathcal{P}(T_1 \times T_2)$. Next, we show that R satisfies the conditions of Figure 5.9.

To see that condition (1) holds consider: (a) maximal concurrent steps give rise to runs, which are naturally matched by any hp bisimulation; and (b) in any hp bisimulation maximal concurrent steps are matched against maximal concurrent steps (Prop. 4.3.2(2)). Since the elements of R stem from pairs of synchronous runs, it is immediate that condition (2) is also satisfied. To verify condition (3a) assume $\gamma \in \mathcal{P}(T_1 \times T_2)$ such that $\exists \gamma' \in R.\ \gamma \subseteq \gamma'$. Further, assume $t_1 \in T_1$ with $proj_1(\gamma) \cup t_1 \in csteps(M_1^0)$. By definition of R we have $r' \in \mathcal{H}$ with $set(r') = \gamma'$. Note that from r' we can backtrack all pairs of transitions $t \in \gamma' \setminus \gamma$; thereby, we obtain $r \in \mathcal{H}$ with $set(r) = \gamma$. Clearly, we have $proj_1(r) \stackrel{t_1 \cdot w_1}{\rightarrow}$ for some $w_1 \in T_1^*$

so that $set(proj_1(r).t_1.w_1) \in mcsteps(M_1^0)$, and by definition of hp bisimulation there must be $t_2 \in T_2$, $w_2 \in T_2^*$ with $r.(t_1, t_2).(w_1, w_2) \in \mathcal{H}$. Certainly, $\gamma'' \equiv set(r.(t_1, t_2).(w_1, w_2)) \in R$, and $\gamma \cup (t_1, t_2) \subseteq \gamma''$, which means t_2 and γ'' provide entities as required. (4b) follows from the symmetrical argument.

It remains to show that $depR(M_1^0, t_1) \sim_{hhp} depR(M_2^0, t_2)$ for each pair $(t_1, t_2) \in \bigcup_{\gamma \in R} \gamma$. Whenever two transitions t_1 and t_2 are matched against each other in a hhp bisimulation, the remaining behaviour of $\mathcal{N}_1$ that is dependent on t_1 must be matched by remaining behaviour of $\mathcal{N}_2$ that is dependent on t_2, and vice versa. But by Prop. 5.4.7(1) and the way R is defined this amounts to the existence of a hhp bisimulation relating $depR(M_1^0, t_1)$ and $depR(M_2^0, t_2)$ for each $(t_1, t_2) \in \bigcup_{\gamma \in R} \gamma$, as required.

(2.) Imagine we are given a set $R \subseteq \mathcal{P}(T_1 \times T_2)$ satisfying the conditions of Figure 5.9, and a family of hhp bisimulation approximations of degree n, $\{\mathcal{H}_{(t_1,t_2)}\}_{(t_1,t_2) \in \bigcup_{\gamma \in R} \gamma}$, such that for each $(t_1, t_2) \in \bigcup_{\gamma \in R} \gamma$, $\mathcal{H}_{(t_1,t_2)}$ relates $depR(M_1^0, t_1)$ with $depR(M_2^0, t_2)$. We show that a hhp bisimulation approximation of degree $n+1$ can be constructed for $\mathcal{N}_1$ and $\mathcal{N}_2$, based on these entities.

First, we prefix each $\mathcal{H}_{(t_1,t_2)}$ by (t_1, t_2) in the following way:

$$\mathcal{H}'_{(t_1,t_2)} = \{(t_1, t_2).r \mid r \in \mathcal{H}_{(t_1,t_2)}\}.$$

Then we define:

$$\mathcal{H} = \bigcup \{r_{t_1} \otimes \cdots \otimes r_{t_n} \mid \exists \gamma \in R.\ \{t_1, t_2, \ldots, t_n\} \subseteq \gamma \ \&\ \forall i \in [1, n].\ r_{t_i} \in \mathcal{H}'_{t_i}\}.$$

To see that $\mathcal{H}$ is a hp bisimulation approximation of degree $n+1$ consider the following three points: (a) condition (3) of Figure 5.9 ensures that the 'subset-closure' of R gives a complete bisimulation match for the concurrent steps of $\mathcal{N}_1$ and $\mathcal{N}_2$; the match is label-preserving by condition (2). (b) By the '$\supseteq$'-direction of Prop. 5.4.10 we clearly have $\mathcal{H} \subseteq Runs(\mathcal{N}_1) \times Runs(\mathcal{N}_2)$; with the '$\subseteq$'-direction and point (a) it follows that $\mathcal{H}$ provides a complete bisimulation match for the behaviour of $\mathcal{N}_1$ and $\mathcal{N}_2$ up to 'length' $n+1$. (c) Because concurrent steps are matched against concurrent steps, and due to Prop. 5.4.7(1) it follows that dependencies are reflected correctly by $\mathcal{H}$.

Moreover, $\mathcal{H}$ is hereditary: this follows because the concurrent steps of $\mathcal{N}_1$ and $\mathcal{N}_2$ are clearly matched in a hereditary way, and each $\mathcal{H}_{(t_1,t_2)}$ is hereditary by assumption. □

Analogously, we then obtain for BPP:

Lemma 5.4.4. *Let $\mathcal{E}$, $\mathcal{F}$ be two BPP in ENF with $E_0 \in ENF(\Delta_{\mathcal{E}})$, $F_0 \in ENF(\Delta_{\mathcal{F}})$.*

1. (a) $\{proj_1(\sigma) \mid \sigma \in R\} = msteps(E_0)$.

   (b) $\{proj_2(\sigma) \mid \sigma \in R\} = msteps(F_0)$.

2. $\forall (t,u) \in \bigcup_{\sigma \in R} \sigma.\ l_{\mathcal{E}}(t) = l_{\mathcal{F}}(u)$.

3. For all $\sigma \in \mathcal{P}(T(E_0) \times T(F_0))$ such that $\exists \sigma' \in R.\ \sigma \subseteq \sigma'$ we have:

   (a) If $proj_1(\sigma) \cup t \in steps(E_0)$ for some $t \in T(E_0)$, then there exist $u \in T(F_0)$, $\sigma'' \in R$ so that $\sigma \cup (t,u) \subseteq \sigma''$.

   (b) Vice versa.

Figure 5.10: Let $\mathcal{E}, \mathcal{F}$ be two BPP in ENF with $E_0 \in ENF(\Delta_{\mathcal{E}})$, $F_0 \in ENF(\Delta_{\mathcal{F}})$. The figure gives conditions for a set $R \subseteq \mathcal{P}(T(E_0) \times T(F_0))$

Rec $\quad \dfrac{X = Y}{E = F} \quad (X \overset{def}{=} E) \in \Delta_{\mathcal{E}}, (Y \overset{def}{=} F) \in \Delta_{\mathcal{F}}$

Match $\quad \dfrac{E = F}{\{locR(E,t) = locR(F,u)\}_{(t,u) \in \bigcup_{\sigma \in R} \sigma}}$

where $E \in ENF(\Delta_{\mathcal{E}}), F \in ENF(\Delta_{\mathcal{F}})$, and $R \subseteq \mathcal{P}(T(E) \times T(F))$ satisfies the conditions of Figure 5.10.

Figure 5.11: Tableau rules for hhp bisimilarity relative to two BPP in ENF $\mathcal{E}$, $\mathcal{F}$

1. *If $\mathcal{E} \sim_{hhp} \mathcal{F}$ then there is a set $R \subseteq \mathcal{P}(T(E_0) \times T(F_0))$ such that R satisfies the conditions of Figure 5.10, and $\forall (t,u) \in \bigcup_{\sigma \in R} \sigma.\ locR(E_0,t) \sim_{hhp} locR(F_0,u)$.*

2. *If there is a set $R \subseteq \mathcal{P}(T(E_0) \times T(F_0))$ such that R satisfies the conditions of Figure 5.10, and $\forall (t,u) \in \bigcup_{\sigma \in R} \sigma.\ locR(E_0,t) \sim^n_{hhp} locR(F_0,u)$ then we have $\mathcal{E} \sim^{n+1}_{hhp} \mathcal{F}$.*

Proof. Straightforward with Lemma 5.4.3, Prop. 5.4.9, 5.4.8, and 5.4.6. $\square$

The Tableau System. Analogously to the other proofs, we translate this insight into a tableau system that decides hhp bisimilarity. Again, the system is designed for BPP in ENF. The rules can be found in Figure 5.11. **Match** and **Rec** will be applied alternately until one of the terminal conditions is reached.

A node n : *label* is a *successful terminal* iff one of the following conditions holds:

1. *label* = "$\mathbf{0} = \mathbf{0}$".

2. *label* = "$X = Y$", and there is an ancestor node n_a above n in the tableau such that n_a is labelled with "$X = Y$" as well.

A node n : *label* is an *unsuccessful terminal* iff the following condition holds:

3. *label* = "$E = F$", where E and F are ENF expressions, and a set R as required by rule **Match** does not exist.

It is clear that Lemma 5.4.4 provides forward and strengthened backwards soundness of rule **Match** for hhp bisimilarity. Finiteness as well as completeness and soundness for hhp bisimilarity of the tableau system can then be proved by using the same arguments as in the corresponding proofs of Section 5.3.2. As usual, the decidability of hhp bisimilarity follows from these three properties.

Theorem 5.4.4. *It is decidable whether two BPP are hhp bisimilar.*

Coincidence of hhp and chhp Bisimilarity. We now benefit from having proved a slight strengthening of the crucial insight: it is easy to check that in the proof of Lemma 5.4.3(2) $\mathcal{H}$ matches the concurrent steps of $\mathcal{N}_1$ and $\mathcal{N}_2$ not only in hereditary fashion but also coherently. Further, if one assumes that the hhp bisimulation approximations relating the dependent remainders are coherent then $\mathcal{H}$ will be coherent, too. Thus, statements analogous to Lemma 5.4.3(2) and Lemma 5.4.4(2) are true for chhp bisimilarity, which gives us strengthened backwards soundness of **Match** for this stricter notion. Then, the tableau system decides chhp bisimilarity just as well, and we obtain:

Theorem 5.4.5. *Two BPP are hhp bisimilar iff they are chhp bisimilar.*

5.5 Final Remarks

Due to their tree-like behaviour SBPP and BPP enjoy good composition and decomposition properties. We have seen that this translates into hp, hhp, and chhp bisimilarity in natural ways, which allowed us to construct straightforward decision procedures, and exhibit several coincidence results. Our main results are summarized in Figure 5.12. There are some further considerations, and points for future research:

	hp bisimilarity	hhp bisimilarity	chhp bisimilarity
SBPP	coincidence & decidability		
BPP	decidability	coincidence & decidability	

Figure 5.12: Summary of the main results

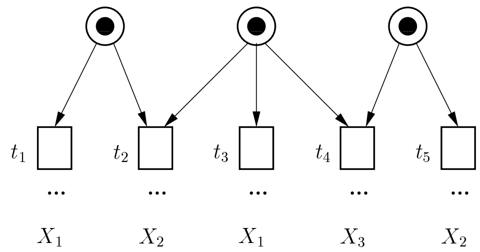


Figure 5.13: A proper-comm free net system that has no BPP representation

Generalized BPP. When we are investigating process languages under true-concurrency semantics the hierarchy of formalisms to consider will be finer than in the interleaving world: a given process language P will typically divide into a spectrum of languages which under interleaving semantics are all equally expressive as P but which form independent classes when moving to true-concurrency semantics. In this chapter, we have already distinguished between SBPP and BPP; now we would like to motivate the addition of a third class to the BPP spectrum.

BPP are not expressive enough to implement all possible mixtures of concurrency and conflict between transitions of one 'level'. This is reflected by the fact that there are (finitely representable) proper-comm free net systems which cannot be represented by any BPP. Figure 5.13 gives an example. One could generalize BPP to overcome this restriction: simply employ lats or Petri net fragments to specify how the transitions of one level are connected (cf. Figure 5.13). To obtain a more process algebra like presentation one could use the left merge operator '$\lfloor$' (cf. Section 5.4.4), together with suitable concurrency axioms. For example, the process corresponding to the net system of Figure 5.13 can also be represented by the following expression: $(t_1 \lfloor ((t_3 \mid\mid t_5) + t_4)) + (t_2 \lfloor t_5) + (t_3 \lfloor (t_1 \mid\mid t_5)) + (t_4 \lfloor t_1) + (t_5 \lfloor ((t_1 \mid\mid t_3) + t_2))$.

It should be straightforward to express these ideas formally, and define the class of *generalized BPP* (short: *GBPP*). Naturally, GBPP will coincide with BPP under interleaving semantics. For causal interleaving semantics the situation is a little more involved. The causal tree induced by the above example cannot be captured by the use of '$\mid\mid$' to express causal independence. To see this note that the component $t_1 \mid\mid ((t_3 \mid\mid t_5) + t_4)$ gives rise to a subtree of behaviour where we

Interleaving semantics	SBPP = BPP = GBPP
Causal interleaving semantics	SBPP $\subset$ BPP$^{\lfloor}$ = GBPP
True-concurrency semantics	SBPP $\subset$ BPP $\subset$ GBPP

Table 1: The BPP spectrum

SBPP	conflict is transitive
BPP	the interplay can be expressed by nesting the transitions via '$\|\|$' and '+'
GBPP	any interplay is allowed

Table 2: Admitted interplay of concurrency and conflict

Figure 5.14: The BPP spectrum

can execute t_5 as first transition, and then t_2 is not possible; such a subtree is not induced by the example. On the other hand, if we allow the '$\lfloor$' operator (naturally without imposing any concurrency axioms as is necessary to define GBPP) then the resulting class, say $BPP^{\lfloor}$, will indeed coincide with GBPP under causal interleaving semantics. Figure 5.14 gives the full BPP spectrum, and shows how the particular classes restrict the interplay of concurrency and conflict.

True-concurrency semantics for GBPP can be defined analogously to our BPP semantics; the resulting net systems will naturally be proper-comm free. Then, it is clear that the BPP results carry over to GBPP: the crucial insights behind our proofs are all formulated for proper-comm free net systems; it should be straightforward to carry the insights over to GBPP and construct tableaux in the same way as we did for BPP.

Results for Net Systems and Lats'. In Section 4.4.3 we showed that hp and hhp bisimilarity coincide for bounded-degree SW-free lats' and bounded-degree comm-free net systems by employing our composition and decomposition insights in an inductive argument. It seems straightforward that coincidence between hhp and chhp bisimilarity can be established for bounded-degree proper-comm free net systems and SW-$\{1, 2\}$ free lats' by applying the decomposition view of Section 5.4.5 in an analogous fashion. Thus, we conjecture:

Conjecture 5.5.1. *1. Two bounded-degree proper-comm free net systems are hhp bisimilar iff they are chhp bisimilar.*

2. Two bounded-degree SW-$\{1, 2\}$ free lats' are hhp bisimilar iff they are chhp bisimilar.

<table>
<tr><th></th><th></th><th>hp bisimilarity</th><th>hhp bisimilarity</th><th>chhp bisimilarity</th></tr>
<tr><td>A</td><td>bounded-degree</td><td colspan="3">coincidence</td></tr>
<tr><td>A</td><td>finite-state</td><td colspan="3">coincidence & decidability</td></tr>
<tr><td>B</td><td>bounded-degree</td><td></td><td colspan="2">coincidence</td></tr>
<tr><td>B</td><td>finite-state</td><td>decidability</td><td colspan="2">coincidence & decidability</td></tr>
</table>

Group A: *comm-free net systems and SW-free lats'*
Group B: *proper-comm free net systems and SW-$\{1, 2\}$ free lats'*

Figure 5.15: Results and conjectures for net sytems and lats'

Since finite-state systems are always bounded-degree we also know that hhp bisimilarity is decidable for finite-state SW-free lats' and finite-state comm-free net systems. On the other hand, we can transfer our conjectures for GBPP to finite-state proper-comm free net systems, and further to finite-state SW-$\{1, 2\}$ free lats': it is easy to see that any finite-state proper-comm free net system $\mathcal{N}$ can be expressed as a GBPP $\mathcal{E}$ such that the unfolding of $\mathcal{N}$ agrees with the unfolding of $\mathcal{E}$. Further, one would expect that there is a translation from SW-$\{1, 2\}$ free lats' to proper-comm free net systems such that two SW-$\{1, 2\}$ free lats' are (c)hhp bisimilar iff their translations to proper-comm free net systems are (c)hhp bisimilar.

Conjecture 5.5.2. *1. It is decidable whether two finite-state proper-comm free net systems are hhp or chhp bisimilar.*

2. It is decidable whether two finite-state SW-$\{1, 2\}$ free lats' are hhp or chhp bisimilar.

From Section 5.4.2 we can carry over that hp and hhp bisimilarity do *not* coincide for proper-comm free net systems; this also follows from our non-coincidence result for SW-$\{1, 2\}$ free lats' of Section 4.4.2. Figure 5.15 gives a summary of our results and conjectures.

In contrast to decidability problems, coincidence investigations can be undertaken for infinite-state classes whose elements are not necessarily finitely describable. This is illustrated by our results on bounded-degree systems. An obvious question to ask is: can we extend our coincidence results to the full classes of comm-free and, respectively, proper-comm free net systems, and similarly for the lats classes? This question has yet to be analysed. It is of theoretical interest to find out whether the results carry over smoothly, and if not what kind of obstacles make this difficult or impossible. If obstacles arise they are bound to be of general interest for the true-concurrency world.

Further Points for Future Research. An interesting point for further research is to extend the BPP algebras by a more sophisticated parallel operator that allows synchronization, and investigate the consequences this has on hp and hhp bisimilarity. For hp bisimilarity it is already known that decidability carries over to BPP_τ, which is BPP plus CCS-style synchronization: in [KH94] it has been proved that causal bisimilarity is decidable for this process algebra. It is important to analyse whether decidability also carries over for hhp bisimilarity: the answer will tell whether the BPP classes are tractable because they have *tree-like* behaviour or whether *syntactically controlled* synchronization is still within the border of decidability.

Instead of extending BPP by synchronization, one could also move up in the *Process Rewrite Systems Hierarchy* [May98]. The next process languages to consider are Petri nets and *PA* (*Process Algebra*). For Petri nets hhp bisimilarity is definitely undecidable, but for PA, which incorporates sequential composition in addition to parallel composition, one might find interesting results.

Chapter 6

Free Choice Systems

6.1 Introduction

In this chapter we focus on our second group of structural system classes: finite-state free choice systems and important subclasses thereof. As described in Section 1.3.1 free choice systems constitute a central class in Petri net theory; they have a structurally controlled interplay of concurrency and conflict, which ensures that they are efficiently analysable while not overly restricted in expressive power. *Live* free choice systems are often considered to be the largest Petri net class that allows a good theory.

Their structural restriction gives free choice systems interesting behavioural properties: it is well-known that they exclude confusion, and consequently they also exclude SW-2 and SW-3; on the other hand, they do admit SW-1. In fact, they can be understood as an alternative generalization of comm-free systems to the one to proper-comm free systems: now we admit synchronization that is separate from conflict rather than allowing an unrestricted interplay of concurrency and conflict at 'one level'.

It has long been believed that confusion is essential to keep hp and hhp bisimilarity distinct, and hence it has been conjectured that the two bisimilarities coincide for free choice systems [Che96]. In Section 4.5, however, we managed to exhibit counter-examples which disprove coincidence for confusion-free systems. Here we will see that the systems employed in these counter-examples are free choice, and consequently we shall carry over that hp and hhp bisimilarity do *not* coincide for this subclass.

There are now two directions to follow: one is to tackle the yet unresolved decidability problem of free choice systems; the second is to consider coincidence and/or decidability for a subclass. We decide to adopt the second approach, and focus our attention on *live* free choice systems. This subclass makes a strong candidate for both, decidability and coincidence. Firstly, it comes with an additional good behavioural property: apart from being confusion-free, live free choice systems appear to be syn-confusion free; they exclude all combinations of frame and MNH situations that have been employed in counter-examples so far. Secondly, classical free choice theory shows that our subclass has good *static* decomposition properties. In particular, we hope to exploit the *S-coverability Theorem* [Hac72], which states that every live free choice system is covered by its *state machine* (short: *SM-*) *components*; consequently, each live free choice system can be understood as a synchronization of a set of live S-systems.

On the one hand, this gives us more reason to believe that hp and hhp bisimilarity coincide for our subclass. It is well-known that in live free choice systems the computations of SM-components are unconstrained by their composite context [TV84]; their computations will at most be interspersed with transitions of other components. As a result, the future behaviour of an SM-component is fully determined by its local state; its computation options cannot be influenced by any parallel action. Considering that all behaviour is made up of component behaviour, one could then speculate that in general there is no reason why the matching should be made dependent on the order of how independent transitions are linearized, and further that hp and hhp bisimilarity coincide. Naturally, this is only a crude intuition; to confirm it we will require deep insights about the matching in hp bisimilarity.

On the other hand, the S-coverability Theorem provides topological information that may help us to prove such insights. One would expect that hp bisimilarity (and more so hhp bisimilarity) respects the compositionality given by an SM-cover to a certain degree; at least one would assume that during the matching of a sequential stretch, the components to which related transitions are assigned stay constant, and that a change can only occur when the respective components synchronize with other components. By fixing a component of one system and tracing how its behaviour is matched in a (h)hp bisimulation we can then gain information about the static structure of the opposite system. This sets up a connection between the matching in (h)hp bisimulations and the topology of the related systems, with whose help we may be able to expose characteristics in the matching, which may in turn be exploited to obtain a coincidence and/or decidability result.

Inspite of this intuition the problem remains inaccessible. What we additionally need is an overall approach that will guide us as to what kind of insights are promising to prove, and how they can be employed to obtain full results. Apart from their static decomposability, live free choice systems also enjoy good *dynamic* decomposition properties: their unfoldings can be understood as interconnections of initially sequential units. This view provides a generalization of our decomposition view for comm-free systems (cf. Section 5.3.2), and the analogy immediately suggests a global way of tackling the coincidence problem: generalizing our proof method for SW-free systems (cf. Section 4.4.3) one could prove that hp and hhp bisimilarity coincide by showing that the dynamic decomposition view translates into the bisimilarities. This idea in turn is hard to implement, but guided by it we have come up with an approach that disentangles the difficulty of the problem by breaking it down into several accessible subgoals. Thereby, we allow for a stepwise advance towards a solution; e.g. decidability can be achieved as an interim result.

Our approach directs us to work with *coherent* hhp (short: *chhp*) bisimilarity instead of hhp bisimilarity. This will not be to our disadvantage. If we achieve coincidence between hp and chhp bisimilarity then in fact we will obtain coincidence between hp, hhp, and chhp bisimilarity. Furthermore, at this stage we are as happy to achieve a decidability result for chhp bisimilarity as we are about one for hhp bisimilarity. The two notions are very close; they bring about the same degree of difficulty due to their 'truly-concurrent' nature. Their subtle difference has yet to be analysed.

Furthermore, as part of our approach we shall adopt two simplifications. Firstly, we restrict our attention for the time being to live *strictly state machine decomposable* (short: *SSMD*) free choice (short: fc) systems. They have the behavioural advantage that each of their SM-components can take its decisions in full freedom. As a consequence, a live SSMD fc system can be viewed as an interconnection of a set of *autonomously* computing live S-systems. This class is very close to the live *strict* fc systems of [ES91]. Secondly, instead of directly working with hp and chhp bisimilarity we shall first tackle the coincidence problem of an auxiliary bisimilarity and its coherent and hereditary version. So-called *compositionality preserving* (short: *cp*) bisimilarity has the benefit of allowing a very direct exploitation of the topological information provided by a *strict SM* (short: *SSM*) cover.

We shall achieve the following results. For live SSMD fc systems we show that cp bisimilarity satisfies a certain decomposition property, called *K-decomposability*. We shall prove this result via the *Crucial Subgoal*^{K}, which will allow us to infer a second result about live SSMD fc systems: the largest cp bisimulation is *sw-(1)coherent* and *sw-(1)hereditary*. This amounts to achieving the first of two conditions that are sufficient for the coincidence of cp, hcp, and chcp bisimilarity. By excluding a special kind of nondeterminism we can additionally overcome the second of the two conditions, and thereby obtain coincidence between cp, hcp, and chcp bisimilarity for the class of live *sy-psd* SSMD fc systems. This is already a good result: it proves that an interleaving concept is as strong as a truly-concurrent one for a substantial system class. Moreover, by further restricting our system class we gain results for hp and (c)hhp bisimilarity: we obtain decidability of chhp bisimilarity for the class of live sy-psd *buffered* SSMD fc systems, and coincidence of hp, hhp, and chhp bisimilarity for the class of live *spsd* buffered SSMD fc systems. The *buffered* restriction introduces a slight structural constraint, whereas the *spsd* condition slightly restricts the nondeterminism in a way that subsumes the *sy-psd* condition. To my knowledge, these are the only positive results for a class that allows a reasonable amount of interplay between causality, concurrency, and conflict while still admitting considerable nondeterminism.

On the way to the Crucial Subgoal^K there are two theorems that deserve mentioning. The *SWFSI Matching Theorem* exhibits a characteristic of the interior of cp bisimulations; it states that in cp bisimilarity on live SSMD fc systems the matching of *switch first synchronization interfaces* (short: *swfsi's*) is deterministic. The SWFSI Matching Theorem builds on the *WNL Theorem*, which in turn exposes a constraint in the topology of live SSMD fc systems; the constraint concerns two new topological entities called *links* and *wedges*. It seems plausible that the two theorems are of further consequence. The first may be employed to improve our results on the coincidence and decidability problem of (hp and) (c)hhp bisimilarity. The second could prove to be useful in a wider context; indeed, there seems to be some connection with a result of [ES91].

The remainder of this introduction is organized as follows. In Section 6.1.1 we give a summary of the approach underlying this work; some intuition can also be found in Appendix C.1. In Section 6.1.2 we then explain what we will achieve, and how we shall arrive there, in view of our approach. Finally, in Section 6.1.3 we provide a synopsis of the chapter.

6.1.1 Approach

We shall now outline the approach that stands behind the work of this chapter. As mentioned earlier, our starting point is the idea that we can tackle the coincidence problem of hp and (c)hhp bisimilarity on live fc systems analogously to our proof method for SW-free systems (cf. Section 4.4.3) by showing that the bisimilarities satisfy certain composition and decomposition aspects. This idea is inspired by the insight that (live) fc systems satisfy a decomposition view (short: *DV-lfcs*) that generalizes the view put forward for comm-free systems (cf. Section 5.3.2): the unfolding of a (live) fc system can be understood as an interconnection of unfoldings of initially sequential units, where the interconnection consists of causality, concurrency, and conflict. DV-lfcs gives rise to a composition and a decomposition property for bisimilarities, called *DV-lfcs composability* and *DV-lfcs decomposability*. As the crucial element of our general plan one has to prove that hp bisimilarity satisfies DV-lfcs decomposability. The degree of difficulty involved in this goal is considerable, and we have developed two intermediary concepts, *bp bisimilarity* and *U-decomposability*, to approach it. Indeed, the latter has led us to a new way of achieving the desired coincidence result, which is more direct than the route via DV-lfcs decomposability; after all, we shall not implement our initial idea but it has acted as the vehicle for leading us onto the right track.

Altogether, we now explain in four steps how, guided by the goal 'hp bisimilarity is DV-lfcs decomposable', the coincidence problem can be broken down into three accessible subgoals. The concepts, insights, and goals we shall come up with are all valid for live fc systems; but according to our second simplification we shall pursue our approach for live SSMD fc systems first.

(1.) In the first step we separate out the *interleaving aspect* of DV-lfcs decomposability, and translate it into a strengthening of hp bisimilarity, called *block preserving* (short: *bp*) *bisimilarity*. Thereby, we reduce the problem 'hp bisimilarity is DV-lfcs decomposable' into two sufficient and necessary subproblems: (1.) hp bisimilarity implies bp bisimilarity, and (2.) bp bisimilarity is DV-lfcs decomposable. The benefit of this reduction is as follows. First of all, it means we have moved everything that can be dealt with on an interleaving level into subproblem (1); this will allow us to focus on the difficult truly-concurrent aspects in 'crystallized form' when tackling subproblem (2). Moreover, if we achieve subproblem (2), we will obtain the decidability of chhp bisimilarity as an interim result: it is straightforward to show that bp bisimilarity is decidable, and with a bit more thought we obtain that chhp bisimilarity implies bp bisimilarity; on the other hand with (2) we will be able to show that bp bisimilarity implies chhp bisimilarity. Naturally, this is the reason why we work with chhp bisimilarity instead of hhp bisimilarity. Subproblem (2) needs to be dissected further, but subproblem (1) we lay down as our first official subgoal:

Subgoal 6.1.1. *hp bisimilarity implies bp bisimilarity.*

(2.) Pursuing the second subproblem, we identify a basic but truly-concurrent decomposition aspect of DV-lfcs decomposability, which is only defined for notions of bisimilarity that already satisfy the interleaving aspect in that they are at least as strong as bp bisimilarity. Our new concept is called *U-decomposability*. Naturally, our aim will be to establish U-decomposability for bp bisimilarity. Not only will this provide a first advance towards subproblem (2), but we speculate that it already gives us the key insight: we hope that DV-lfcs decomposability can be reached by employing its basic aspect U-decomposability in some complicated inductive argument.

(3.) In the end, our quest for U-decomposability leads us to a shortcut that provides a more direct way of achieving 'bp bisimilarity implies chhp bisimilarity' than the route via subproblem (2); this is the point where we can abandon the concept of DV-lfcs decomposability altogether.

Analysing how U-decomposability can be achieved, we come up with a sufficient condition, called the *Crucial Subgoal^x*, where x denotes the respective type of bisimilarity. Apart from establishing U-decomposability for x bisimilarity Crucial Subgoal^x has a second consequence: it implies that the largest x bisimulation satisfies specific padding and backtracking properties, from which we can read that x bisimilarity is *(1)coherent* and *(1)hereditary*. We can then exploit a general insight about bisimilarities and their (1)coherent and (1)hereditary versions: if x bisimilarity coincides with its (1)coherent and (1)hereditary version in the strict sense that *any* x bisimulation can be extended to a (1)coherent and (1)hereditary one, then any x bisimulation can furthermore be extended to one that is fully coherent and hereditary. To sum up, by achieving Crucial Subgoal^{bp} we will obtain coincidence between bp bisimilarity and its coherent and hereditary version, chbp bisimilarity. Since chbp bisimilarity naturally implies chhp bisimilarity, and considering that chhp bisimilarity implies bp bisimilarity, this will entail coincidence between bp and chhp bisimilarity, and furthermore the decidability of chhp bisimilarity.

We will classify the places and transitions of free choice systems into two types, called *switch* and *synch*. U-decomposability, Crucial Subgoal^x, and the properties (1)coherent and (1)hereditary can be split into two analogous parts. This will provide us with an easy way of further structuring our approach: the first part of Crucial Subgoal^x is designed to prove *sw_i-U decomposability*, and as a second consequence it will imply that x bisimilarity is *sw_i-(1)coherent* and *sw_i-(1)hereditary*; symmetrically, the second part of Crucial Subgoal^x will entail that x bisimilarity satisfies *sy_i-U decomposability*, and that it is *sy_i-(1)coherent* and *sy_i-(1)hereditary*; the i can be set to either 1 or 2.

(4.) The last step corresponds to our second simplification: for technical ease and to be better able to exploit the static decomposition information given by an SSM cover, we shall first work with *cp bisimilarity* instead of bp bisimilarity. Accordingly, as our second official subgoal we set:

Subgoal 6.1.2.

1. *Achieve the first part of Crucial Subgoal^{cp}.*

2. *Achieve the second part of Crucial Subgoal^{cp}.*

As we know from above, by achieving Subgoal 6.1.2 we obtain coincidence between cp bisimilarity and its coherent and hereditary version (and corresponding partial results in case we only achieve part 1 or 2). But since we lack the inclusion 'chhp bisimilarity implies cp bisimilarity' we will not be able to conclude to results for chhp bisimilarity. To be able to do so, we will need to transfer our proof of Subgoal 6.1.2 to bp bisimilarity. Potential pitfalls in overcoming this gap are captured by the difference between bp and cp bisimilarity, which can be characterized as consisting of *technical inaccuracies* and more crucially the open *issue of pending synch places*. As an alternative approach, we could overcome the technical inaccuracies by defining a more sophisticated version of cp bisimilarity, say cpb', carry over Subgoal 6.1.2 to cpb', and then deal with the issue of pending synch places while trying to show that chhp bisimilarity implies cpb' (which might be done via bp bisimilarity). Thus, our final subgoal will be:

Subgoal 6.1.3. *Overcome the discrepancy that results from our working with cp bisimilarity instead of bp bisimilarity, which amounts to resolving the technical inaccuracies and the issue of pending synch places.*

6.1.2 Realization

Having summarized our overall plan we now explain what we will realize in this chapter, and how we shall arrive there.

As our main achievement we will establish that the first part of Subgoal 6.1.2 indeed holds. We set out to prove that cp bisimilarity on live SSMD fc systems satisfies *K-decomposability*. The latter captures sw_i-U decomposability for cp bisimilarity (where $i = 1$, or equivalently 2) in a way that incorporates the fact that cp bisimilarity respects the static decompositionality given by SSM covers in a natural manner. Analysing how K-decomposability could be achieved we come up with a subgoal that directly corresponds to the Crucial Subgoal^{cp}. This so-called *Crucial Subgoal*^{K} will only provide an intermediate step in our analysis. Via a chain of further subgoals we are led to the concept of *switch first synchronization interface* (short: *swfsi*). We sketch how the Crucial Subgoal^{K} will follow if we can show that the matching of swfsi's in cp bisimilarity on live SSMD fc systems is deterministic. This final subgoal is far from trivial: it requires us to expose a deep insight about the interior of cp bisimulations. We will establish it as the *SWFSI Matching Theorem*. The essence of its proof is as follows: on the one hand, we show that whenever there exists more than one *swfsi-match* for a given place then we can infer a specific topological scenario, namely a *wedge* and a *link* connected in a characteristic way; this will be possible by exploiting a connection between the matching in cp bisimulations and the topology of the related systems. On the other hand, we show that the topology of live SSMD fc systems is constrained in that the very same scenario cannot exist. This is the statement of the *WNL Theorem*. Altogether, by *reductio ad absurdum* we can then conclude that the matching of swfsi's in cp bisimilarity is indeed deterministic. In the proof of the WNL Theorem we exploit our understanding of live SSMD fc systems as interconnections of autonomously computing SM-components; in particular, we shall employ a setting where we allow the fixing of *courses* for a subset of components.

With the SWFSI Matching Theorem it will then be straightforward to establish the Crucial Subgoal^{K}, and furthermore our two main results for live SSMD fc systems: the Crucial Lemma directly implies that cp bisimilarity is indeed K-decomposable, and as a second consequence we obtain that cp bisimilarity is sw-(1)coherent and sw-(1)hereditary. Figure 6.1 gives an overview of the modules and their interconnections.

From Section 6.1.1 we know that the latter amounts to solving half of the coincidence problem of cp and chcp bisimilarity in the positive direction. Full coincidence would follow if we could further show that cp bisimilarity is sy-(1)coherent and sy-(1)hereditary, which is consequent on Subgoal 6.1.2(2). Subgoal 6.1.3 and Subgoal 6.1.1 represent two more gaps. Subgoal 6.1.3 captures the discrepancy that results from our working with cp bisimilarity instead of bp bisimilarity; by additionally overcoming this gap we would achieve decidability for chhp bisimilarity. Subgoal 6.1.1 depicts the difference between hp and bp bisimilarity, and thereby what is still required, to obtain coincidence for hp, hhp, and chhp bisimilarity.

In general, the three subgoals will remain open for now, but we shall see that

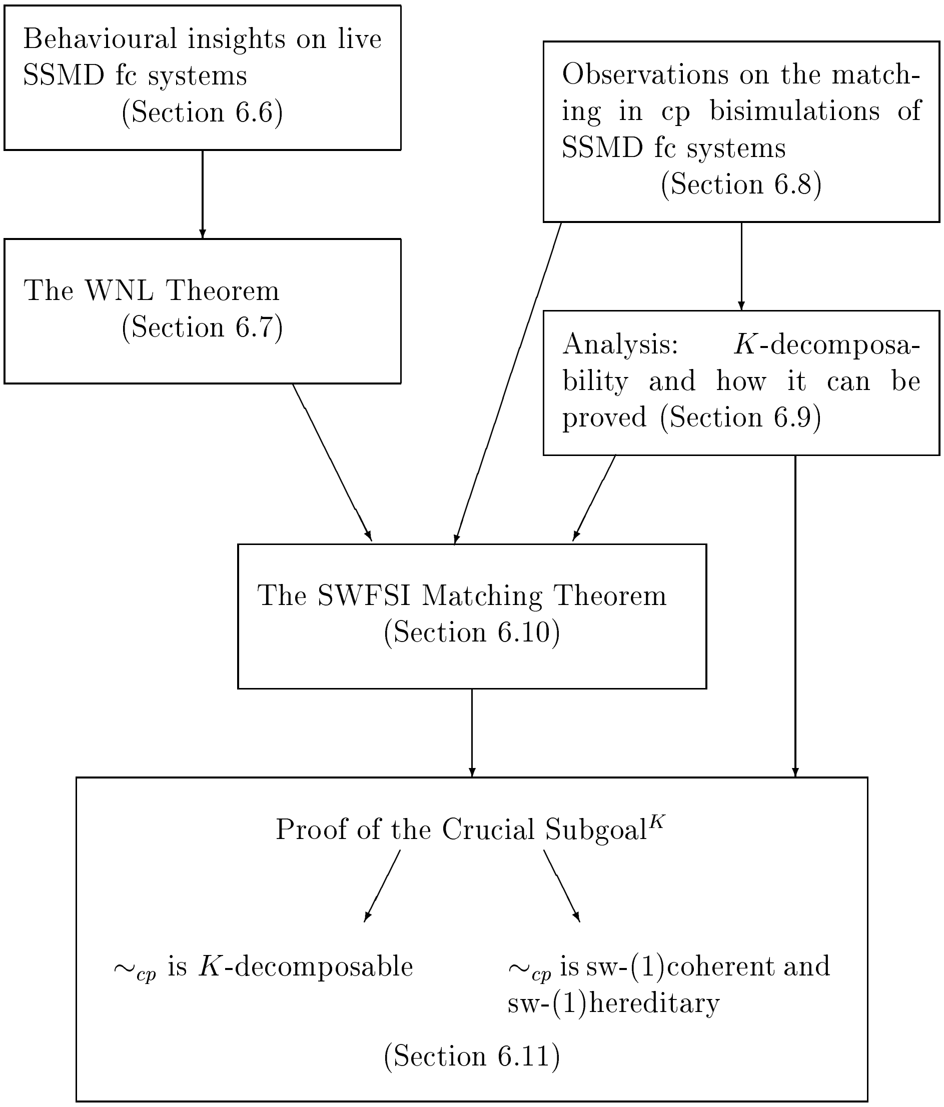


Figure 6.1: Overview of the modules and their logical interdependence ($\leftarrow$ means 'immediately depends on insights of')

the gaps they represent can easily be overcome by imposing slight restrictions on our system class. A constraint on the nondeterminism available at the postset of a synch transition, called *sy-psd*, forces that cp bisimilarity is sy-(1)coherent and sy-(1)hereditary. With the *buffered* condition we achieve that each synch place occurrence is uniquely identified by its generator event. Thereby we will overcome the issue of pending synch places as well as the technical inaccuracies, and induce bp and cp bisimilarity to coincide. By additionally restricting the nondeterminism available at the postset of a transition in a way that subsumes the sy-psd condition, we obtain that cp, bp, and hp bisimilarity coincide; the corresponding condition is called *spsd*. Altogether, this gives us our full results: coincidence of cp, hcp, and chcp bisimilarity for live sy-psd SSMD fc systems,

Gap	Restriction	Achievement
Subgoal 6.1.2(2)	sy-psd	$\sim_{cp}$ is sy-(1)coherent & sy-(1)hereditary
Subgoal 6.1.3	buffered	coincidence between cp and bp bisimilarity
Subgoal 6.1.1	spsd buffered	coincidence between bp and hp bisimilarity

Figure 6.2: Closing the remaining gaps

decidability of chhp bisimilarity for live sy-psd buffered SSMD fc systems, and coincidence of hp, hhp, and chhp bisimilarity for live spsd buffered SSMD fc systems. A summary of the gaps and corresponding restrictions can be found in Figure 6.2.

6.1.3 Synopsis

The remainder of the chapter is organized into three parts, which are separated by two 'interludes'.

Part I comprises the following three sections. In Section 6.2 we provide the background material, including the introduction of our primary system classes and their behavioural properties. In Section 6.3 we formally carry over our non-coincidence result for confusion-free systems to free choice systems. Section 6.4 is about cp bisimilarity: we introduce this notion together with its coherent and hereditary version, provide some preliminary observations, and give our proof methodology.

Part II consists of Section 6.5 to 6.11. Here we present the modules that together prove our two main results on live SSMD fc systems: cp bisimilarity is K-decomposable, and sw-(1)coherent and sw-(1)hereditary. In Section 6.5, which is Interlude I, we describe in detail how this material is organized.

Part III is formed by the final three sections. Section 6.12 provides Interlude II. In the subsequent two sections we bridge over the remaining gaps by further restricting our system class: in Section 6.13 we achieve coincidence of cp, hcp, and chcp bisimilarity for live sy-psd SSMD fc systems; in Section 6.14 we obtain decidability of chhp bisimilarity for live sy-psd buffered SSMD fc systems, and coincidence of hp, hhp, and chhp bisimilarity for live spsd buffered SSMD fc systems. Later on, in Section 7.2, we shall comment on our attempt at the decidability problem of free choice systems.

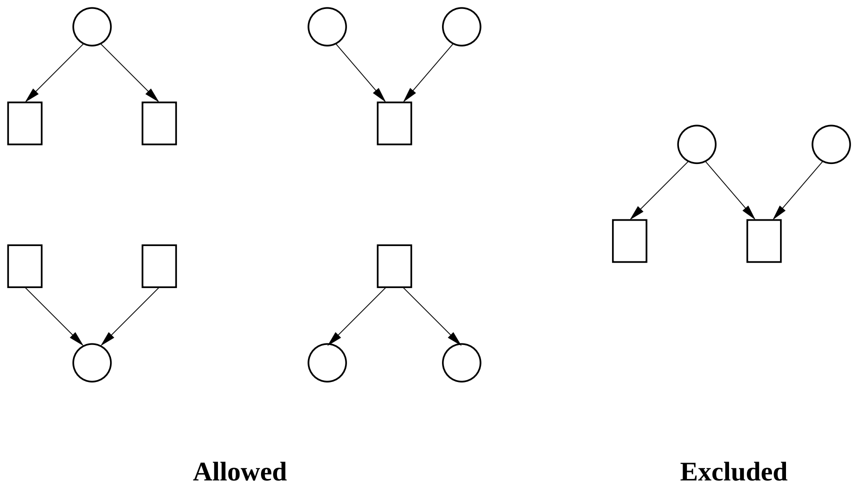


Figure 6.3: Allowed and excluded substructures of free choice nets

6.2 Background

We now present the relevant background material. In the first four sections we give a formal account of the system classes central to this chapter; we present the necessary definitions, and formally introduce the properties that we have identified as crucial to our approach. In Section 6.2.1 we give the definition of free choice systems. In Section 6.2.2 we introduce the notions of SM- and SSM-decomposability. Section 6.2.3 is about live free choice systems: we conjecture that such systems are syn-confusion free, and formally introduce the S-coverability Theorem. Furthermore, in Section 6.2.4 we give the properties that motivate live SSMD fc systems as an interconnection of a set of autonomously computing live S-systems. The fifth and final section presents technical concepts and properties that will be required in the course of the chapter.

6.2.1 Free Choice Systems

Formally, free choice nets and systems are defined as follows ([Rei82]):

Definition 6.2.1 (free choice nets, free choice systems).

A net N is a *free choice net* iff for each arc $(s,t) \in F_N \cap (S_N \times T_N)$ we have: $s^\bullet = \{t\} \vee {}^\bullet t = \{s\}$.

A system $\mathcal{N}$ is a *free choice system* iff its underlying net is free choice.

Free choice nets can also be characterized by either of the following two conditions: (1.) If two transitions share an input place p then neither has any input place apart from p. Or equivalently: (2.) If two places have a common output transition t then neither has any output transition apart from t. This underlines how conflict and synchronization are separated out by the free choice restriction: conflict is only allowed in the 'S-system way', and synchronization only in the 'T-system way' (cf. Figure 6.3). It is a direct consequence of the first condition that if several transitions compete for a token, it is always possible to choose between the transitions freely:

Observation 6.2.1. *Let N be a free choice net. If two transitions t_1 and t_2 of N have a common input place then at any marking of N, either t_1 and t_2 are both enabled, or neither of them is enabled.*

For free choice systems this means that the conflict set of a transition t stays constant during the entire time of its enabledness.[1] Conflict sets can therefore not be influenced by parallel transitions, which immediately implies that free choice systems are confusion-free.

Fact 6.2.1. *Free choice systems are confusion-free.*

The elements of free choice nets can naturally be classified into two different types: we distinguish between *switch* and *synchronization* places, and transitions, respectively. Note that we consider 'switch' to be the default type.

Definition 6.2.2 (types of elements). Let N be a free choice net.

We say $t \in T_N$ is

- a *synchronization* (short: *synch*) *transition* iff $|{}^\bullet t| > 1$, and
- a *switch transition* otherwise.

We say $p \in P_N$ is

- a *synchronization* (short: *synch*) *place* iff $|{}^\bullet(p^\bullet)| > 1$, and
- a *switch place* otherwise.

Clearly, we have:[2]

Observation 6.2.2.

1. *A switch transition has exactly one input place p, and p is of type switch.*
2. *A synch place has exactly one output transition t, and t is of type synch.*

[1] Note that 1-safeness is crucial here.
[2] For (1) consider Restriction 2.1.1.

We assign to a synch place p its set of synchronization partners:

Definition 6.2.3 (synch partners). Let N be a free choice net, and p a synch place of N. We define the *synchronization* (short: *synch*) *partners of* p as $SPartners(p) := \{p' \in P_N \mid p' \neq p \ \&\ p' \in {}^\bullet(p^\bullet)\}$.

We have the following two characteristic properties:

Observation 6.2.3.

1. *A switch transition is enabled iff its unique input place has a token.*
2. *A token on a synch place p can only be taken away by p's unique output transition.*

For technical convenience we introduce the following restriction:

Restriction 6.2.1. From now on, we will only consider nets N that satisfy the following property: $\forall p \in P_N.\ {}^\bullet p \cup p^\bullet \neq \emptyset$.

We carry over the property of *liveness* from lats' to net systems, and *well-formedness* from lats bases to nets in the obvious way (cf. Def. 4.6.1). It is easy to see that for well-formed free choice nets Restriction 6.2.1 implies:

Fact 6.2.2. *Let N be a well-formed free choice net. $\forall p \in P_N.\ p^\bullet \neq \emptyset \ \&\ {}^\bullet p \neq \emptyset$.*

Finally, recall that we are only concerned with *finite-state* free choice systems.

6.2.2 SMD and SSMD Systems

We now introduce the two notions of decomposability which are central in this chapter: *state machine* (short: *SM-*) *decomposability*, and in particular *strict state machine* (short: *SSM-*) *decomposability*. We shall also present several related facts: first about decomposition functions, then about the behaviour of SM-components. We will make use of these facts later on.

6.2.2.1 Definitions

SM- and SSM-decomposability are based on the concept of a *state machine component* (short: *SM-component*). Roughly speaking, SM-components are strongly connected S-systems which fully reflect the forwards and backwards branching at the respective places in the underlying system. For their definition we require the idea of an induced subnet.

Definition 6.2.4 (subnet, induced subnet). Let $N = (S, T; F)$ be a net.

We say a net $N' = (S', T'; F')$ is a *subnet of* N iff

1. $S' \subseteq S$,
2. $T' \subseteq T$, and
3. $F' = F \cap ((S' \times T') \cup (T' \times S'))$.

Let X be a non-empty set of elements of N. We define the *subnet induced by* X to be the subnet $N' = (S', T'; F')$ with

$$\begin{array}{ll} S' & = S \cap ({}^\bullet X \cup X^\bullet \cup X), \\ T' & = T \cap ({}^\bullet X \cup X^\bullet \cup X), \text{ and} \\ F' & = F \cap ((S' \times T') \cup (T' \times S')). \end{array}$$

Definition 6.2.5 (S-components, SM-components). Let $N = (S_N, T_N; F_N)$ be a net, and $K = (S_K, T_K; F_K)$ be a subnet of N. K is an *S-component of* N iff

1. K is a strongly connected S-graph, and
2. K is the subnet of N induced by S_K, i.e. $T_K = {}^\bullet S_K \cup S_K{}^\bullet$ and $F_K = F_N \cap ((S_K \times T_K) \cup (T_K \times S_K))$ (where the dot relation is the one of N).

Let $\mathcal{N} = (N, M_0)$ be a system. A net $K = (S_K, T_K; F_K)$ is a *state machine component* (short: *SM-component*) *of* $\mathcal{N}$ iff K is an S-component of N, and $|M_0(S_K)| = 1$.

The concept needed next is that of an SM-cover:

Definition 6.2.6 (S-covers, SM-covers). Let N be a net. A set *Cover* $= \{K_1, \ldots, K_n\}$ of S-components of N is an *S-cover of* N iff for every $p \in P_N$ there exists $K_i \in$ *Cover* such that $p \in P_{K_i}$. We then say *Cover covers* N.

Let $\mathcal{N} = (N, M_0)$ be a system. A set *Cover* $= \{K_1, \ldots, K_n\}$ of SM-components of $\mathcal{N}$ is an *SM-cover of* $\mathcal{N}$ iff *Cover* is an S-cover of N.

If a system has an SM-cover then it can be understood as a synchronization of strongly connected S-systems. This gives us the notion of SM-decomposability; the corresponding structural concept is that of S-decomposability.

Definition 6.2.7 (S-decomposability, SM-decomposability). A net N is *S-decomposable* (short: *SD*) iff N has an S-cover. A pair (N, *Cover*) is an *S-decomposed* (short: *SD*[3]) *net* iff N is a net, and *Cover* is an S-cover of N.

A system $\mathcal{N}$ is *state machine decomposable* (short: *SMD*) iff $\mathcal{N}$ has an SM-cover. A pair ($\mathcal{N}$, *Cover*) is a *state machine decomposed* (short: *SM-decomposed* or *SMD*[3]) *system* iff $\mathcal{N}$ is a system, and *Cover* is an SM-cover of $\mathcal{N}$.

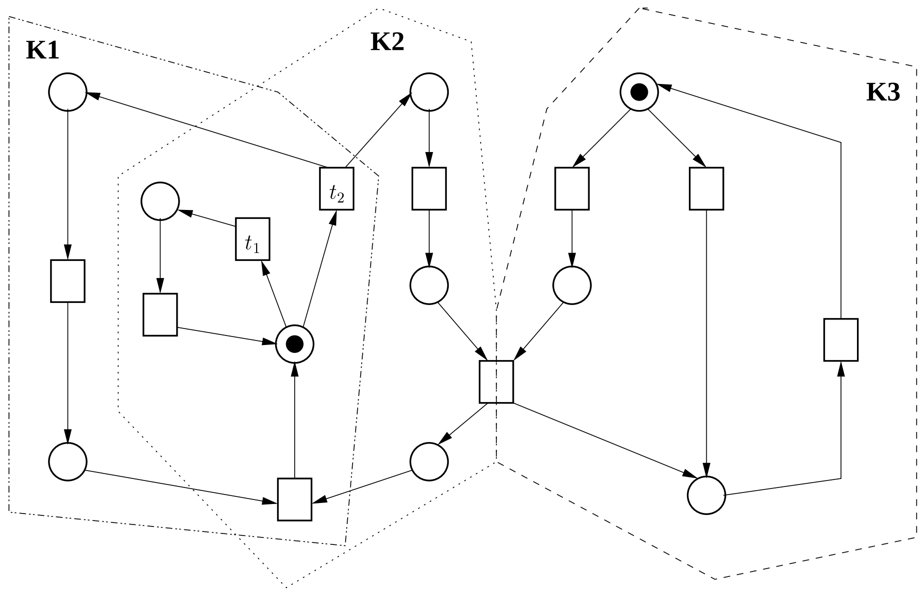


Figure 6.4: A free choice system that is SMD but not SSMD

Example 6.2.1. Figure 6.4 gives an example of a SMD fc system. Its SM components are indicated by the frames.

The example demonstrates that in SM-decomposability we allow components to overlap not only in transitions but also in places. This is excluded in *strict* SM-decomposability: SSMD systems can be viewed as a set of strongly connected S-systems which are connected together only through transitions. Behaviourally this means that each component holds the sole control over its tokens. Note that in SSMD *fc* systems the interconnecting transitions are exclusively of type synch. We will see in Section 6.2.4 that as a result each component can take its decisions in full freedom.

Definition 6.2.8 (strict S-cover, strict SM-cover). Let N be a net. A set $Cover = \{K_1, \ldots, K_n\}$ of S-components of N is a *strict S-cover of* N iff for every $p \in P_N$ there exists *exactly one* $K_i \in Cover$ such that $p \in P_{K_i}$.

Let $\mathcal{N} = (N, M_0)$ be a system. A set $Cover = \{K_1, \ldots, K_n\}$ of SM-components of $\mathcal{N}$ is a *strict SM-cover of* $\mathcal{N}$ iff $Cover$ is a strict S-cover of N.

Definition 6.2.9 (SS-decomposability, SSM-decomposability). A net N is *strictly S-decomposable* (short: *SSD*) iff N has a strict S-cover. A pair $(N, Cover)$ is a *strictly S-decomposed* (short: *SS-decomposed* or *SSD*[3]) *net* iff N is a net, and $Cover$ is a strict S-cover of N.

[3]The ambiguity with 'decomposable' will be resolved by the context.

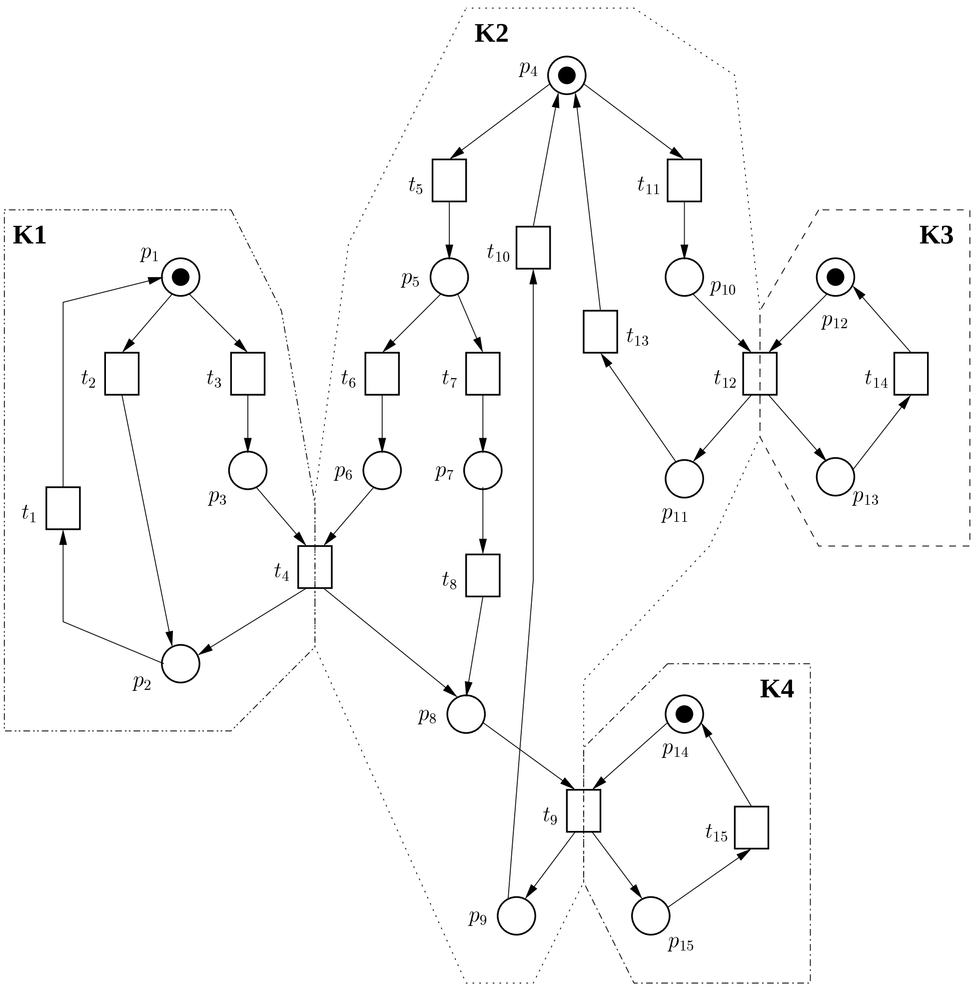


Figure 6.5: A SSMD fc system

A system $\mathcal{N}$ is *strictly state machine decomposable* (short: *SSMD*) iff $\mathcal{N}$ has a strict SM-cover. A pair $(\mathcal{N}, Cover)$ is a *strictly state machine decomposed* (short: *SSM-decomposed* or *SSMD*[3]) *system* iff $\mathcal{N}$ is a system, and *Cover* is a strict SM-cover of $\mathcal{N}$.

Example 6.2.2. Figure 6.5 shows a SSMD fc system. The system of Figure 6.4 is SMD but *not* SSMD.

Convention 6.2.1. In the following, we will transfer definitions and properties of systems and nets to (S)SM-decomposed systems, and (S)S-decomposed nets respectively.

Let $(\mathcal{N} = (N, M_0), Cover)$ be a (S)SM-decomposed system. $(N, Cover)$ is clearly a (S)S-decomposed net, and hence we shall transfer definitions and properties of $(N, Cover)$ to $(\mathcal{N}, Cover)$.

It will be useful to associate a *decomposition function* with each S-decomposed net:

Definition 6.2.10. Let (N, Cover) be a S-decomposed net.

The *decomposition function associated with* N, $\mathit{Ks} : S_N \cup T_N \to \mathcal{P}(\mathit{Cover})$, is defined by:

$$\forall x \in S_N \cup T_N, K \in \mathit{Cover}.\ K \in \mathit{Ks}(x) \iff x \in S_K \cup T_K.$$

If Cover is a strict cover then clearly $\forall p \in P_N.\ |\mathit{Ks}(p)| = 1$; in this case we shall use $K(p)$ to denote the corresponding singleton component.

Further, we generalize the function Ks to sets of places, and sets and (infinite) sequences of transitions in the obvious way.

Finally, we shall employ the following terminology:

Definition 6.2.11. Let (N, Cover) be a S-decomposed net, and $K \in \mathit{Cover}$.

Let M be a marking of N. We denote $M \!\uparrow\! S_K$ by $M(K)$.

Furthermore, assume N to be free choice. We define the *synch transitions of* K by:

$$\mathit{synchT}(K) = \{t \in T_N \mid t \text{ is of type synch } \& \ K \in \mathit{Ks}(t)\}.$$

Let $w \in T_N^*$. We say there is *no synchronization of* K *on* w, denoted by $\mathit{nosynch}(w, K)$, iff $\nexists t \in w.\ t \in \mathit{synchT}(K)$.

Let $p \in P_N$ of type synch. We define the *synchronization (partner) components* of p by: $\mathit{KSPartners}(p) = \mathit{Ks}(\mathit{SPartners}(p))$.

Let M_K be a safe marking of K. We say γ_K is an *infinite* K*-computation at* M_K iff $\gamma_K \in T_K^\omega$ & $\forall w \in \mathit{FinPrefixes}(\gamma_K).\ M_K[w\rangle$. Given an infinite K-computation γ_K at M_K we define the *infinite synchronization partners of* K *on* γ_K by $\mathit{InfSPartners}(K, \gamma_K) = \bigcup\{\mathit{Ks}(t) \backslash \{K\} \mid t \text{ occurs infinitely often on } \gamma_K\}$.

Bibliographic Notes. The concepts concerning SM-decomposability are standard; they go back to the pioneering work on fc systems by Hack in [Hac72] (reference quoted from [TV84]). The particular names and definitions used here are partly from [TV84], and partly from [ES91].

Although SSM-decomposability is clearly very natural, I have not come across this concept in literature. On the other hand, SSMD *fc* systems are very close to the strict fc systems of [ES91]. These were defined to exactly capture the subclass of SMD fc systems in which the SM-components can take their decisions in full freedom. We will require that very property in our proofs, and this is in fact what compelled me to work with something stricter than live fc systems. I was led to defining SSMD systems since for our purpose they are technically very convenient to use. It should, however, be no problem to transfer all our results on SSMD fc systems to the slightly more general strict fc systems.

6.2.2.2 Structural Observations

We now present some elementary properties about S-decomposed nets and their associated decomposition functions.

For this, two immediate facts about S-components will be helpful:

Proposition 6.2.1. *Let N be a net, and K an S-component of N. Note that the dot relation refers to the one of N.*

1. $\forall t \in T_K.\ |{}^\bullet t \cap S_K| = 1 = |t^\bullet \cap S_K|$.
2. $\forall p \in S_K.\ {}^\bullet p \cup p^\bullet \subseteq T_K$.

Proof. Immediate with the definition of S-component. □

Then, the following is straightforward:

Proposition 6.2.2. *Let $(N, Cover)$ be a S-decomposed net. For all $t \in T_N$ we have:*

1. (a) $\forall p, p' \in {}^\bullet t.\ p \neq p' \implies Ks(p) \cap Ks(p') = \emptyset$, *and*
   (b) $\forall p, p' \in t^\bullet.\ p \neq p' \implies Ks(p) \cap Ks(p') = \emptyset$.
2. (a) $\forall p \in {}^\bullet t.\ Ks(p) \subseteq Ks(t)$, *and*
   (b) $\forall p \in t^\bullet.\ Ks(p) \subseteq Ks(t)$, *and even more:*
3. (a) $Ks({}^\bullet t) = Ks(t)$, *and*
   (b) $Ks(t^\bullet) = Ks(t)$.

Proof. (1.) is immediate with Prop. 6.2.1(1), and (2.) with Prop. 6.2.1(2). For (3.) consider: the '$\subseteq$'-directions are given by (2), whereas the '$\supseteq$'-directions easily follow with Prop. 6.2.1(1). □

Next comes a natural fact about independence: whenever two transitions have disjoint sets of components then they are independent of each other.

Proposition 6.2.3. *Let $(N, Cover)$ be a S-decomposed net.*

1. *For all* $t,\ t' \in T_N$ *we have:* $Ks(t) \cap Ks(t') = \emptyset \implies t\ I_N\ t'$, *and so:*

2. *For all* $w,\ w' \in T_N^*$ *we have:* $Ks(w) \cap Ks(w') = \emptyset \implies w\ I_N\ w'$.

Proof. (1.) If $Ks(t) \cap Ks(t') = \emptyset$, then by Prop. 6.2.2(3) $Ks({}^\bullet t \cup t^\bullet) \cap Ks({}^\bullet t' \cup t'^\bullet) = \emptyset$, and so certainly $({}^\bullet t \cup t^\bullet) \cap ({}^\bullet t' \cup t'^\bullet) = \emptyset$. (2.) is immediate with (1.). □

In SS-decomposed *fc* nets the type of transitions is identifed by the decomposition function:

Proposition 6.2.4. *Let* $(N, Cover)$ *be a SS-decomposed fc net. For all* $t \in T_N$ *we have:* t *is of type switch iff* $|Ks(t)| = 1$ *(or equivalently:* t *is of type synch iff* $|Ks(t)| > 1$*).*

Proof. By Prop. 6.2.2(1a) and since *Cover* is strict, we obtain $|Ks({}^\bullet t)| = |{}^\bullet t|$, and further by Prop. 6.2.2(3a) $|Ks(t)| = |{}^\bullet t|$. Then the proposition is immediate: t is of type switch (synch) iff $|{}^\bullet t| = 1$ ($|{}^\bullet t| > 1$). □

This implies an intuitive characterization of $synchT(K)$, which we shall often employ implicitly.

Proposition 6.2.5. *Let* $(N, Cover)$ *be a SS-decomposed fc net,* $K \in Cover$*, and* $t \in T_N$. $t \notin synchT(K) \iff Ks(t) = \{K\} \vee Ks(t) \subseteq Cover \backslash K$.

Proof. This is straightforward with Prop. 6.2.4. □

6.2.2.3 Behavioural Observations

In the following, we give several behavioural properties of SM-components.

By definition every SM-component K of a system $\mathcal{N}$ holds exactly one token at the initial marking of $\mathcal{N}$. This stays so during any computation of $\mathcal{N}$:

Proposition 6.2.6. *Let* $\mathcal{N}$ *be a system, and* K *an SM-component of* $\mathcal{N}$*. For all* $M \in Reach(\mathcal{N})$ *we have:* $|M(K)| = 1$.

Proof. By induction on the number of transitions that need to be fired to reach M, and with the help of Prop. 6.2.1. □

Convention 6.2.2. We shall sometimes employ $M(K)$ to denote the singleton place given by Prop. 6.2.6. It will always be clear from the context whether we regard $M(K)$ as a place, or as a set of places as usual.

Let N be an S-graph, and M a marking of N. It is a well-known result that M is live and safe for N iff N is strongly connected and $|M| = 1$ (see e.g. [BT87]). Together with Prop. 6.2.6 and the definition of S-component this immediately gives us:

Proposition 6.2.7. *Let $\mathcal{N}$ be a system, and K an SM-component of $\mathcal{N}$. For all $M \in Reach(\mathcal{N})$ we have: $M(K)$ is a live and safe marking for K.*

Thus, at any reachable marking M of its underlying system an SM-component K can be understood as the live system $(K, M(K))$. Naturally, a computation in the underlying system implies a computation for K in the following way:[4]

Proposition 6.2.8. *Let $\mathcal{N}$ be a system, K an SM-component of $\mathcal{N}$, and $M \in Reach(\mathcal{N})$. If $M[w\rangle_N M'$ for some $w \in T_N^*$, M', then we have $M(K)[w{\uparrow}T_K\rangle_K M'(K)$.*

Proof. Easy by induction on the length of w with the help of Prop. 6.2.1(2), and the following fact: $\forall t \in T_K.\ (pres(K,t) = pres(N,t) \cap S_K)\ \&\ (posts(K,t) = posts(N,t) \cap S_K)$. □

The property has the following immediate consequences, which will be helpful later on:

Proposition 6.2.9. *Let $\mathcal{N}$ be a system, K an SM-component of $\mathcal{N}$, and $M \in Reach(\mathcal{N})$.*

1. *$M(K) \in Reach(K, M_0(K))$.*
2. *Let $M[w\rangle_N M'$ for some $w \in T_N^*$, M'. $K \notin Ks(w) \implies M(K) = M'(K)$.*
3. *Let $M[w\rangle_N M'$ for some $w \in T_N^*$, M'. For all $p \in S_N$ satisfying $Ks(p) \cap Ks(w) = \emptyset$ we have:*
$$p \in M \iff p \in M'.$$

Proof. (1.) and (2.) follow from Prop. 6.2.8; (3.) is easy with (2.). □

Furthermore, it is now easy to present a counterpart of Prop. 6.2.3: whenever two independent transitions can occur consecutively in a SM-decomposed system then their sets of components must be disjoint.

Proposition 6.2.10. *Let $(\mathcal{N}, Cover)$ be a SM-decomposed system. For all t, $t' \in T_N$ we have: $t\ I\ t'\ \&\ (\exists M \in Reach(\mathcal{N}).\ M[tt'\rangle) \implies Ks(t) \cap Ks(t') = \emptyset$.*

Proof. Let entities be given as above, and to the contrary assume $K \in Ks(t) \cap Ks(t')$. By Prop. 6.2.8 we can infer $M(K)[tt'\rangle_K$, which, since $t\ I\ t'$, implies K is capable of concurrent behaviour. But K is an S-system, and hence this cannot be true. □

[4] This is similar to one direction of Theorem 2.1 in [TV84].

By themselves SM-components are live; but when considered in the context of their underlying system they can clearly become deadlocked. It is natural that from the liveness of a given system we can conclude that its SM-components can never become deadlocked in context:

Proposition 6.2.11. *Let $\mathcal{N}$ be a system, K an SM-component of $\mathcal{N}$, and $M \in Reach(\mathcal{N})$. If $\mathcal{N}$ is live then there exists $r \in Runs_N(M)$ such that $K \in Ks(r)$.*

Proof. Trivial by liveness and the fact that $T_K \neq \emptyset$ for any S-component K (this follows from Restriction 6.2.1). $\square$

Finally, we note some straightforward connections, which will be useful later on.

Proposition 6.2.12. *Let $(\mathcal{N}, Cover)$ be a SSM-decomposed system; assume $M, M' \in Reach(\mathcal{N})$, $K' \in Cover$, $p \in S_N$, and $t \in T_N$.*

1. *Let $p \in M$. $M(K(p)) = p$.*
2. *$M(K') = p \iff K' = K(p)$ & $p \in M$.*
3. *Let $M[t\rangle M'$. $K' \in Ks(t) \implies M(K') \in {}^\bullet t$ & $M'(K') \in t^\bullet$.*
4. *$M[t\rangle$, $p \in M$ & $K(p) \in Ks(t) \implies p \in {}^\bullet t$.*

Proof. (1) and (2) easily follow by definition; (3) is immediate with Prop. 6.2.2(3a) and (2), and (4) with (1) and (3). $\square$

6.2.3 Live Free Choice Systems

Initially, we motivated live fc systems as a promising candidate for the coincidence of hp and (c)hhp bisimilarity on the basis of the following two characteristics.

Firstly, live fc systems come with good behavioural properties: in addition to being confusion-free, they appear to be syn-confusion free; they exclude all the combinations of MNH and frame situations that have been employed in counter-examples so far. Although it is straightforward that live fc systems exclude a net specific version of syn-confusion it seems tricky to prove that they satisfy the behavioural version of Section 4.5.3. For now, we can only conjecture:

Conjecture 6.2.1. *Live fc systems are syn-confusion free.*

Secondly, live fc systems come with interesting decomposition properties: they are SMD. This follows from a major piece of classical free choice theory, the S-coverability Theorem of Hack [Hac72] (reference quoted from [TV84]):

Theorem 6.2.1 (S-coverability Theorem). *Every live fc system* (N, M_0) *is covered by S-components which have exactly one token each at* M_0.

Corollary 6.2.1. *Live fc systems are SMD.*

The S-coverability Theorem provides a structural characterization of safeness in live fc Petri nets. This is the purpose for which the theorem was originally developed. It is complemented by another classical result, known as Commoner's Theorem, which in turn gives a structural criterion for liveness in fc Petri nets (see [Com72, Hac72]). Together, the two theorems make up a major part of the structure theory of Petri nets.

For us the S-coverability Theorem is important in its nature as a decomposition result for live fc systems. We will mainly focus on live SSMD fc systems in this chapter, but Corollary 6.2.1 underlines how close the two classes are. Live fc systems have been the starting point for this work, and there is hope that our results on live SSMD fc systems can be carried over to this slightly more general class.

6.2.4 More on Free Choice Systems

In this subsection we consider the following two issues: (1.) How can the computations of SM-components be affected by a free choice context? (2.) How much freedom do SM-components have for taking their decisions in SM-decomposed fc systems? Concerning these issues, we will derive the following two important properties: (1.) In *live* fc systems the computations of SM-components are unconstrained by their context; their computations will at most be interspersed with transitions of other components. (2.) The SM-components of *strictly* SM-decomposed fc systems can take their decisions in full freedom. Together, the two properties ensure: any live *and* SSMD fc system can be viewed as an interconnection of a set of autonomously computing live S-systems that occasionally synchronize with each other.

A bibliographic note: the first property was proved in [TV84]. As mentioned earlier, the second property is the key characteristic of the strict fc systems of [ES91], and indeed the motivation for their definition. The presentation given here is influenced by these two references, but more elaborate in several aspects and geared towards the needs of this chapter. In particular, note that in [TV84] live fc systems (which are SMD by Corollary 6.2.1) have been characterized as an "interconnection of a set of autonomous live and safe S-graphs that occasionally synchronize with each other". For us the second aspect of autonomy is crucial, and we shall therefore reserve such a characterization for live SSMD fc systems.

6.2.4.1 The Computations of SM-components in Free Choice Systems

Assuming a fc system $\mathcal{N}$ and an SM-component K of $\mathcal{N}$, we would like to know how the computations of K can be affected by the context of $\mathcal{N}$. To find out, we examine how the composite context influences the enabledness of transitions of K. It is easy to see that the enabledness of switch transitions is not affected at all:

Observation 6.2.4. *Let $\mathcal{N}$ be a fc system, K an SM-component of $\mathcal{N}$, $t \in T_K$ of type switch w.r.t. N, and $M \in Reach(\mathcal{N})$. We have:*

$$M(K)[t\rangle_K \implies M[t\rangle_N.$$

This is not the case for synch transitions: let t be a K-transition of type synch; in the context of $\mathcal{N}$, the singleton K-preplace of t will have synchronization partners, and hence t's enabledness will also depend on whether these are ready for the transition. Note that due to the free choice restriction K will be stuck until the respective synchronization can be performed. In particular, K cannot perform a transition other than t as its next transition.

Observation 6.2.5. *Let $\mathcal{N}$ be a fc system, K an SM-component of $\mathcal{N}$, $t \in T_K$ of type synch w.r.t. N, and $M \in Reach(\mathcal{N})$. $M(K)[t\rangle_K$ implies in N:*

1. *t can only be performed after the synchronization partners of K have got ready for t; that is for all $t_1 t_2 \dots t_n \in Runs_N(M)$, $i \in [1, n]$ we have:*

   $$t_i = t \implies \exists j \in [0, i-1].\ SPartners(M(K)) \subseteq M_j,$$

   where for $i \in [0, n]$ M_i is given by $M_0 \equiv M \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots M_{n-1} \xrightarrow{t_n} M_n$.

2. *There is no other K-transition that can be executed before t; that is for all $t_1 t_2 \dots t_n \in Runs_N(M)$, $i \in [1, n]$ we have:*

   $$t_i \in T_K \;\&\; (\forall j \in [1, i-1].\ t_j \notin T_K) \implies t_i = t.$$

Altogether we have: an SM-component K is not affected by its context as long as it performs switch transitions; if K reaches a synchronization interface it can get delayed waiting for its synchronization partners to get ready. Thereby, K can become deadlocked: it may be the case that the synchronization partners can never be reached. Figure 6.6 gives an example of a system that is prone to such deadlocks.

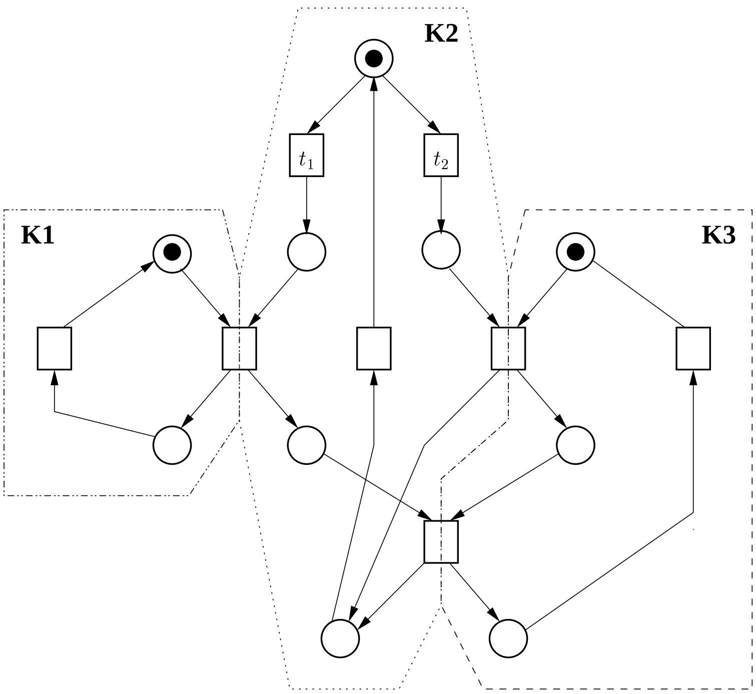


Figure 6.6: A SSMD fc system, which is prone to deadlocks; e.g. if K_2 chooses t_1 as its next transition then K_3 will be deadlocked, and later on K_2 and K_1 will also become deadlocked

Live Free Choice Systems. Now, assume $\mathcal{N}$ to be live. Then, by Prop. 6.2.11 it is not possible that SM-components of $\mathcal{N}$ become deadlocked. Moreover, by definition of liveness any transition of $\mathcal{N}$ can always be made to occur again. Combining either of these facts with Obs. 6.2.5(2) gives us: if a synch transition t is enabled in K then it can also be performed in $\mathcal{N}$, although possibly delayed by transitions that do not belong to K.

Observation 6.2.6. *Let $\mathcal{N}$ be a live fc system, K an SM-component of $\mathcal{N}$, $t \in T_K$ of type synch w.r.t. N, and $M \in Reach(\mathcal{N})$.*

If $M(K)[t\rangle_K$ then there is $r \in Runs_N(M)$ such that $r{\uparrow}T_K = \varepsilon$ & $M[r.t\rangle$.

Together, Obs. 6.2.4 and 6.2.6 imply that in live fc systems the computations of SM-components are not constrained at all by the composite context: any computation of an SM-component K can also be performed in the context of $\mathcal{N}$; the computation will at most be interspersed by transitions that do not belong to K.

Theorem 6.2.2. *Let $\mathcal{N}$ be a live fc system, K an SM-component of $\mathcal{N}$, and $M \in Reach(\mathcal{N})$.*

For all $r_K \in Runs_K(M(K))$ there is $r \in Runs_N(M)$ such that $r{\uparrow}T_K = r_K$.

Proof. By induction on the length of r_K using Obs. 6.2.4 and 6.2.6. $\square$

Finally, we give a related observation, which will be useful in the sequel:

Proposition 6.2.13. *Let $(\mathcal{N}, Cover)$ be a live SM-decomposed fc system, $K \in Cover$, and $M \in Reach(\mathcal{N})$. For all $p \in M$ such that p is of type synch, there are r, M' such that $M[r\rangle_N M'$, $SPartners(p) \subseteq M'$, and $Ks(p) \cap Ks(r) = \emptyset$.*

Proof. This follows similarly to Obs. 6.2.6. $\square$

6.2.4.2 The Decision-Making of SM-components in SM-decomposed Free Choice Systems

We now investigate how much freedom the components of SM-decomposed fc systems have for making their decisions.

Definition 6.2.12 (choice set). Let $\mathcal{N}$ be a system, and $M \in Reach(\mathcal{N})$. A *choice set of $\mathcal{N}$ at M* is a set $C \subseteq T_N$ such that $|C| > 1$ & $\forall t \in C.\ M[t\rangle$.

First of all, assume a fc system $\mathcal{N}$, and an SM-component K of $\mathcal{N}$. We study how choice sets of K can be affected by the context of $\mathcal{N}$. Thereby, we find: decisions cannot be taken away from K by its context; whenever in K there is a decision to take on how to proceed then the corresponding decision is pending in $\mathcal{N}$ and at any successor marking of $\mathcal{N}$ until K resolves the decision by taking its next move.

Observation 6.2.7. *Let $\mathcal{N}$ be a fc system, K an SM-component of $\mathcal{N}$, $M \in Reach(\mathcal{N})$, and C a choice set of K at $M(K)$. For all $w \in T_N^*$ and M' such that $M[w\rangle M'$ we have:*

$$T_K \cap set(w) = \emptyset \implies C \text{ is a choice set of } \mathcal{N} \text{ at } M'.$$

Proof. Note that the elements of C must all be switch transitions with the same unique input place $M(K)$. The observation then follows easily with Prop. 6.2.9(2) and Obs. 6.2.3(1). $\square$

Thus, $\mathcal{N}$ does not constrain at all the decisions that are due in its SM-components. However, given a concrete SM-decomposition *Cover* for $\mathcal{N}$, it is easy to see that the components of *Cover* might influence each other in how to resolve their decisions: since the components can overlap not only in synch transitions but also in switch transitions, they might have to make some decisions in agreement with each other. This is illustrated by Figure 6.4: K_1 and K_2 have to synchronize their pending choice between t_1 and t_2.

SSM-decomposed Free Choice Systems. The components of *strictly* SM-decomposed systems do not overlap in switch transitions, and hence they can take their decisions by themselves, without having to agree with other components:

Observation 6.2.8. *Let $(\mathcal{N}, Cover)$ be an SSM-decomposed fc system, $K \in Cover$, $M \in Reach(\mathcal{N})$, and C a choice set of K at $M(K)$. For all $t \in C$ we have:*

1. $Ks(t) = \{K\}$.
2. *Let M' be such that $M[t\rangle M'$. $\forall K' \in Cover \backslash K.\ M(K') = M'(K')$.*

Proof. Note that the elements of C are all switch transitions. Then (1.) follows with 6.2.4. (2.) follows from (1.) with Prop. 6.2.9(2).

□

Taken together Obs. 6.2.7 and 6.2.8 mean:

> The components of an SSM-decomposed fc system can take their decisions in full freedom.

6.2.5 Further Preliminaries

Finally, we present technical concepts and properties, which will be required in the sequel. Section 6.2.5.1 gives a well-known fact concerning liveness. Section 6.2.5.2 is about paths; in particular we deal with paths in decomposed nets, and cover the close relationship between paths and runs in S-graphs. In Section 6.2.5.3 we present observations about projections of processes and runs onto components. Finally, Section 6.2.5.4 introduces a technical framework, which we shall employ in most sections of this chapter.

6.2.5.1 Place-liveness

As defined in Section 4.6 liveness is concerned with the occurrence of transitions. It is possible to consider a notion of liveness with respect to places just as well: a place is live if from any reachable marking it can always be filled with a token again. Correspondingly, we call a system place-live, if all of its places are live. This notion has been defined e.g. in [Bes87].

Definition 6.2.13 (place-liveness). Let $\mathcal{N}$ be a system.

$p \in P_N$ is *live* iff $\forall M \in Reach(\mathcal{N}).\ \exists M' \in [M\rangle.\ p \in M'$.

$\mathcal{N}$ is *place-live* iff $\forall p \in P_N.\ p$ is live.

It is not difficult to see that under Restriction 6.2.1 liveness implies place-liveness. For an explicit proof see [DE95] or [Bes87].

Proposition 6.2.14. *If a system $\mathcal{N}$ is live then it is also place-live.*

For a superclass of free choice systems, called asymmetric choice systems, the converse also holds, that is place-liveness and liveness are equivalent. This has been proved in [Bes87]. But for our purpose the above direction is sufficient.

6.2.5.2 Paths

Since nets can be regarded as bipartite graphs, we can employ standard graph terminology. We will use the concept of *paths* together with the following terminology. Note that we will classify paths according to the type of their first and last elements e.g. we speak of *PP-paths* or *PT-paths*. (similar to [ES91]).

Definition 6.2.14 (paths). Let N be a net.

A (directed finite) *path of* N is a nonempty sequence $\pi = x_1 x_2 \dots x_n$ such that $x_i \in S_N \cup T_N$, and for $i \in [1, n-1]$, $(x_i, x_{i+1}) \in F_N$. We say π leads from x_1 to x_n.

The *first element of* π is defined as $\mathit{first}(\pi) := x_1$, and the *last element of* π as $\mathit{last}(\pi) := x_n$.

Let X, Y range over $\{P, T\}$. If x_1 is an X-element and x_n is a Y-element, then we say π is an *XY-path*.

We shall also employ *circuits* and *infinite paths*.

Definition 6.2.15 (circuits, infinite paths). Let N be a net.

A *circuit of* N is a path π of N such that $(\mathit{last}(\pi), \mathit{first}(\pi)) \in F_N$.

An *infinite path of* N is an infinite sequence $\pi = x_1 x_2 \dots x_i \dots$ such that $x_i \in S_N \cup T_N$, and for $i \in \mathbb{N}$ $(x_i, x_{i+1}) \in F_N$. We say π starts at x_1, and define the *first element of* π as $\mathit{first}(\pi) := x_1$. If $\mathit{first}(\pi) \in S_N$ then we say π is an *infinite P-path.*

Paths and infinite paths are always alternating sequences of places and transitions; we associate the corresponding (infinite) transition sequence to each (infinite) path:

Definition 6.2.16 ($ts(\pi)$). Let N be a net, and π either a finite or infinite path of N. We define the *(infinite) transition sequence of* π as $ts(\pi) := \pi{\uparrow}T_N$.

Paths in S-decomposed Nets. In the context of an S-cover, we attach to a path π the set of components participating in π. Moreover, given some component K, we distinguish the paths which are also paths in K and call them *K-paths*.

Definition 6.2.17 ($Ks(\pi)$, K-paths). Let (N, Cover) be an S-decomposed net, and π either a finite or infinite path of N.

We define the *components of* π as $Ks(\pi) := \bigcup_{x\in\pi} Ks(x)$.

Let $K \in \mathit{Cover}$. We say π is an *(infinite) K-path* iff π is an (infinite) path of K. With a corresponding meaning we employ the term *K-circuit*.

Proposition 6.2.15. *Let (N, Cover) be an S-decomposed net, and π a path of N that contains at least one transition. We have: $Ks(\pi) = Ks(ts(\pi))$.*

Proof. This is immediate with Prop. 6.2.2(3). □

To K-paths of SS-decomposed nets we associate their set of *synchronization partners* as follows.

Definition 6.2.18 ($\mathit{SPartners}(K, \pi_K)$). Let (N, Cover) be an SS-decomposed net, $K \in \mathit{Cover}$, and π_K a K-path of N. We define the *synchronization partners of K on π_K* as $\mathit{SPartners}(K, \pi_K) := Ks(ts(\pi))\backslash\{K\}$.

The Interrelation between Paths and Runs in S-graphs. In S-graphs there is a close relation between paths and runs: on the one hand, PP-paths give rise to runs; on the other hand, certain runs induce PP-paths.

Proposition 6.2.16. *Let N be an S-graph.*

1. *For all PP-paths π of N we have: $\{\mathit{first}(\pi)\}[ts(\pi)\rangle_N\{\mathit{last}(\pi)\}$.*

2. *Let $p \in P_N$. For all runs $t_1 \dots t_n \in \mathit{Runs}_N(\{p\})$ we have: $p_0t_1p_1 \dots t_np_n$ is a PP-path of N, where $p_0 = p$ and for $i \in [1, n]$ p_i is given by $\{p_{i-1}\}[t_i\rangle_N\{p_i\}$.*

Proof. This is easy to read from the definition of S-graph and the Petri net firing rule. □

The first part has the following two immediate consequences:

Proposition 6.2.17. *Let N be an S-graph.*

1. *Let π^ω be an infinite P-path of N. $ts(\pi^\omega)$ is an infinite run at $\{\mathit{first}(\pi^\omega)\}$.*

2. *Assume N to be strongly connected, and let $p, p' \in P_N$. There exists a run r such that $\{p\}[r\rangle_N\{p'\}$.*

Proof. (1) and (2) are immediate with Prop. 6.2.16(1). For (2) consider that by strong connectedness there exists a PP-path from p to p'. □

6.2.5.3 Subprocesses

We now present several observations, which will be required to introduce and investigate our decomposition property K-decomposability (cf. Section 6.9). We work in the context of a SSM-decomposed fc system $(\mathcal{N}, \mathit{Cover})$, and intend to view its set of reachable markings as a domain of processes. Accordingly, we work with the following technical setting:

Definition 6.2.19. We define the set of *processes of $\mathcal{N}$* by $\mathit{Proc} := \mathit{Reach}(\mathcal{N})$, and the set of *runs of $\mathcal{N}$* by $\mathit{Runs} := \bigcup_{M \in \mathit{Proc}} \{(M, r) \mid r \in \mathit{Runs}(N, M)\}$. We carry over $\rightarrow$ to Proc and Runs in the obvious way.

In the previous sections we have often projected processes and runs of $\mathcal{N}$ onto the elements of a component $K \in \mathit{Cover}$ in order to extract the share of K. Usually, we have interpreted the outcome with respect to K, and employed the projection functions to infer local behaviour from global. Now, given a set $\mathcal{K} \subseteq \mathit{Cover}$, we shall interpret such projections with respect to the entire net N. Thereby, we will obtain subprocesses and sub-behaviour of $\mathcal{N}$.

Convention 6.2.3. Given $\mathcal{K} \subseteq \mathit{Cover}$ and an entity x of $\mathcal{N}$, we use $x \uparrow \mathcal{K}$ as short for $x \uparrow (\bigcup_{K \in \mathcal{K}} S_K \cup T_K)$.

From the behaviour of a process M we can infer behaviour for the subprocess $M \uparrow \mathcal{K}$ in the following way:

Proposition 6.2.18. *Let $M \xrightarrow{t} M'$, and $\mathcal{K} \subseteq \mathit{Cover}$.*

1. $\mathit{Ks}(t) \subseteq \mathcal{K} \implies M \uparrow \mathcal{K} \xrightarrow{t} M' \uparrow \mathcal{K}$.
2. $\mathit{Ks}(t) \cap \mathcal{K} = \emptyset \implies M \uparrow \mathcal{K} = M' \uparrow \mathcal{K}$.

Proof. This follows easily from the Petri net firing rule and Prop. 6.2.2(3). ☐

Conversely, from the behaviour of $M \uparrow \mathcal{K}$ we can infer behaviour for M:

Proposition 6.2.19. *Let $M \in \mathit{Proc}$, and $\mathcal{K} \subseteq \mathit{Cover}$.*
$M \uparrow \mathcal{K} \xrightarrow{t} M'_{\mathcal{K}} \implies \mathit{Ks}(t) \subseteq \mathcal{K} \ \& \ M \xrightarrow{t} M'_{\mathcal{K}} \cup M \uparrow (\mathit{Cover} \backslash \mathcal{K})$.

Proof. Again, this is straightforward by the Petri net firing rule and Prop. 6.2.2(3). ☐

With these insights it is easy to show: a subprocess $M \uparrow \mathcal{K}$ exactly captures the share of $\mathcal{K}$ in the behaviour of M up to synchronization between components of $\mathcal{K}$ and components not contained in $\mathcal{K}$.

Proposition 6.2.20. *Let $M \in Proc$, and $\mathcal{K} \subseteq Cover$.*

$$
\begin{aligned}
&Runs_N(M{\uparrow}\mathcal{K}) = \\
&\qquad \{r{\uparrow}\mathcal{K} \mid r \in Runs_N(M) \ s.t.\ \forall t \in r.\ (Ks(t) \subseteq \mathcal{K} \ \vee\ Ks(t) \subseteq (Cover \backslash \mathcal{K}))\}.
\end{aligned}
$$

Proof. For the '$\subseteq$'-direction we prove:

$$M{\uparrow}\mathcal{K} \xrightarrow{r} M'_{\mathcal{K}} \implies Ks(r) \subseteq \mathcal{K} \ \&\ M \xrightarrow{r} M'_{\mathcal{K}} \cup M{\uparrow}(Cover\backslash\mathcal{K}),$$

which is straightforward by induction on the length of r and Prop. 6.2.19. On the other hand, the '$\supseteq$'-direction follows from:

$$M \xrightarrow{r} M' \ \&\ \forall t \in r.\ (Ks(t) \subseteq \mathcal{K} \vee Ks(t) \subseteq (Cover\backslash\mathcal{K})) \implies M{\uparrow}\mathcal{K} \xrightarrow{r{\uparrow}\mathcal{K}} M'{\uparrow}\mathcal{K}.$$

Analogously, this is easy by induction on the length of r and Prop. 6.2.18. ☐

Later, given $M \in Proc$ and $K \in Cover$, we will split M into its subprocesses $M{\uparrow}K$ and $M{\uparrow}(Cover\backslash K)$. The following two propositions will be helpful; they are immediate with the previous characterization.

Proposition 6.2.21. *Let $M \in Proc$.*

1. *Let $K \in Cover$, and set $\mathcal{R} = Cover\backslash K$.*
   (a) *For all $r \in Runs_N(M{\uparrow}K)$ we have $nosynch(K, r)$.*
   (b) *For all $r \in Runs_N(M{\uparrow}\mathcal{R})$ we have $nosynch(K, r)$.*
2. *Let $\mathcal{K} \subseteq Cover$. For all $r \in Runs_N(M{\uparrow}\mathcal{K})$ we have $Ks(r) \subseteq \mathcal{K}$.*

Proof. This is immediate by Prop. 6.2.20. For (1) also consider Prop. 6.2.5. ☐

Conversely, we have:

Proposition 6.2.22. *Let $M \in Proc$, and $r \in Runs_N(M)$.*

1. *Let $K \in Cover$, and set $\mathcal{R} = Cover\backslash K$. If $nosynch(K, r)$ then we have:*
   (a) *$r{\uparrow}K \in Runs_N(M{\uparrow}K)$, and*
   (b) *$r{\uparrow}\mathcal{R} \in Runs_N(M{\uparrow}\mathcal{R})$.*
2. *Let $\mathcal{K} \subseteq Cover$. If $Ks(r) \subseteq \mathcal{K}$ then we have: $r{\uparrow}\mathcal{K} \in Runs_N(M{\uparrow}\mathcal{K})$.*

Proof. Again, this is immediate by Prop. 6.2.20. For (1) also consider Prop. 6.2.5. ☐

Finally, the following shows how from shuffling behaviour of subprocesses of M we can infer behaviour for M.

Proposition 6.2.23. *Let $M \in Proc$, and $\mathcal{K}$, $\mathcal{K}' \subseteq Cover$ such that $\mathcal{K} \cap \mathcal{K}' = \emptyset$ and $\mathcal{K} \cup \mathcal{K}' = Cover$. Then for any $L \subseteq Runs_N(M \uparrow \mathcal{K})$, $L' \subseteq Runs_N(M \uparrow \mathcal{K}')$ we have: $L \otimes L' \subseteq Runs_N(M)$.*

Proof. It is easy to see that for any $M \in Proc$, $\mathcal{K}, \mathcal{K}' \subseteq Cover$ the following holds: (1) $\mathcal{K} \cap \mathcal{K}' = \emptyset \implies M \uparrow \mathcal{K} \cap M \uparrow \mathcal{K}' = \emptyset$. (2) $\mathcal{K} \cup \mathcal{K}' = Cover \implies M \uparrow \mathcal{K} \cup M \uparrow \mathcal{K}' = M$. But then the proposition immediately follows from Prop. 2.5.2. □

6.2.5.4 A Framework

So far, we have employed two alternative ways of defining hp and (c)hhp bisimilarity: in the standard definition we introduced each of the notions as a relation on separate systems with specified initial state. On the other hand, in Section 3.2 we presented (h)hp bisimilarity as a relation on the runs of a fixed system which is thought to cover all the behaviour one is interested in. Here, we shall require yet another technical setting.

Assume two net systems $\mathcal{N}_1$ and $\mathcal{N}_2$, and consider that their sets of reachable markings can be viewed as two domains of processes (cf. Section 6.2.5.3). We are interested in comparing behaviour of the first process domain with behaviour of the second, and vice versa. Accordingly, we define ((c)h)hp bisimilarity as a relation of pairs of runs (r_1, r_2), where r_1 is a run of $\mathcal{N}_1$, and r_2 a run of $\mathcal{N}_2$. Since we primarily regard $\mathcal{N}_1$ and $\mathcal{N}_2$ as generators of process domains we naturally allow runs to start at any reachable marking, or process as we will say in this context.

Formally, we fix net systems $\mathcal{N}_i = (N_i \equiv (S_i, T_i; F_i), M_0^i)$ for $i = 1, 2$, and associate the following entities with them:

Definition 6.2.20. For $i = 1, 2$ we define

- the set of *processes of* $\mathcal{N}_i$ by $Proc_i := Reach(\mathcal{N}_i)$, and
- the set of *runs of* $\mathcal{N}_i$ by $Runs_i := \bigcup_{M_i \in Proc_i} \{(M_i, r) \mid r \in Runs(N_i, M_i)\}$.

We carry over $\to$ to $Proc_i$ and $Runs_i$ in the obvious way. Furthermore, we define the set of *synchronous runs (of* $\mathcal{N}_1$ *and* $\mathcal{N}_2$*)* by:

$SRuns := \{(r_1, r_2) \in Runs_1 \times Runs_2 \mid proj_2(r_1)$ and $proj_2(r_2)$ are synchronous$\}$.

We assume BEn and δ to be adapted correspondingly; for subsets of $SRuns$ we carry over the property of prefix-closed in the obvious way.

Now, we are ready to redefine ((c)h)hp bisimilarity. As usual we restrict ourselves to *prefix-closed* hp bisimulations.

Definition 6.2.21 (((c)h)hp bisimilarity). A *hp bisimulation* is a *prefix-closed* relation $\mathcal{H} \subseteq SRuns$ that satisfies

1. If $(r_1, r_2) \in \mathcal{H}$ and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, then there is some t_2 so that $r_2 \xrightarrow{t_2} r_2.t_2$ and $(r_1.t_1, r_2.t_2) \in \mathcal{H}$.

2. Vice versa.

A hp bisimulation $\mathcal{H}$ is *hereditary* (short: h) when it further satisfies

3. If $r \in \mathcal{H}$ and $t \in BEn(r)$, then $\delta(r, t) \in \mathcal{H}$.

A hhp bisimulation $\mathcal{H}$ is *coherent* (short: c) when it further satisfies

4. If $rw, rt \in \mathcal{H}$ such that $t\ I\ w$ then $rtw \in \mathcal{H}$.

We say two runs $r_1 \in Runs_1$, $r_2 \in Runs_2$ are *((c)h)hp bisimilar*, written $r_1 \sim_{((c)h)hp} r_2$, iff $(r_1, r_2) \in \mathcal{H}$ for some ((c)h)hp bisimulation $\mathcal{H}$. That is, we define:

$$\sim_{((c)h)hp} = \bigcup \{\mathcal{H} : \mathcal{H} \text{ is a ((c)h)hp bisimulation}\}.$$

We can then define ((c)h)hp bisimilarity between systems as a derived notion:

Definition 6.2.22. $\mathcal{N}_1$ and $\mathcal{N}_2$ are *((c)h)hp bisimilar* iff $((M_0^1, \varepsilon), (M_0^2, \varepsilon)) \in \sim_{((c)h)hp}$.

It is easy to check that this relation on systems coincides with ((c)h)hp bisimilarity as given by the standard definitions (Def. 2.2.2, 2.2.7).

6.3 hp and hhp Bisimilarity do not Coincide for Free Choice Systems

Having defined free choice systems we can now easily see that the systems of Counter-example 3 (Figure 4.1, 4.2) and Counter-example 4 (Figure 4.8) are free choice. Thus, we have:

Theorem 6.3.1. *hp and hhp bisimilarity do not coincide in general for free choice systems.*

6.4 cp and (c)hcp Bisimilarity

This section is about our auxiliary notions of bisimilarity: *compositionality preserving* (short: *cp*) bisimilarity together with its (coherent and) hereditary version. In Section 6.4.1 we introduce and define our new concepts. In Section 6.4.2 we briefly discuss how the cp bisimilarities are related to the hp bisimilarities. Section 6.4.3 gives some first observations, and introduces a shuffle operation for the domain cp bisimilarity is based on. Finally, in Section 6.4.4 we introduce two insights, which provide proof methodology analogously to two points of our general approach (cf. Section 6.1.1). Throughout the section we fix two systems $\mathcal{N}_1$ and $\mathcal{N}_2$, and adopt the setting of Section 6.2.5.4.

6.4.1 Introducing cp and (c)hcp Bisimilarity

The behaviour of a net system $\mathcal{N}$ is compositional in the following sense: each process of $\mathcal{N}$ can be viewed as a parallel composition of atomic subprocesses, or places. Accordingly, if a process M evolves into a new process via a transition t, M is affected only locally: t replaces a subset of subprocesses of M by a new set of subprocesses as it is specified by the input and output places of t; all the other subprocesses of M remain unaffected.

cp bisimilarity is designed to preserve these aspects of compositionality: firstly, it ensures that two related processes M_1 and M_2 are decomposed in the same fashion: it requires that the subprocesses of M_1 and M_2 are matched to each other by a bijection. Secondly, cp bisimilarity makes sure that whenever two transitions t_1, t_2 are matched against each other at a pair of processes M_1, M_2 then t_1 has the same local effect on M_1 as t_2 has on M_2: (a) in the bijection that relates M_1 and M_2 the input places of t_1 must be matched to the ones of t_2, and vice versa; (b) in the bijection that relates the resulting processes M_1' and M_2' the output places of t_1 must be matched to the ones of t_2, and vice versa, and the remaining places must be related as before.

We shall see that preserving the compositional structure of two systems in this manner entails preserving their causal structure: cp bisimilarity respects the immediate cause relationship between transition occurrences in runs. Importantly for us, this immediately implies that cp bisimilarity is partial order preserving.

On the other hand, cp bisimilarity is not a truly-concurrent notion: it fully respects the compositional dependencies between processes and transitions within a linearized run, but it is not capable of coordinating the matching of linearizations which represent the same partial order execution. To ensure that the matching is done in a truly-concurrent fashion we need to impose a backtracking (and padding) condition on cp bisimilarity: we will define a hereditary (and coherent) version of cp bisimilarity in the same way as we obtained (c)hhp bisimilarity.

The coincidence and decidability problems associated with cp and (c)hcp bisimilarity have similar characteristics to the ones associated with hp and (c)hhp bisimilarity. In particular, decidability of cp bisimilarity is easily obtained whereas the decidability problem of (c)hcp bisimilarity is difficult. On the other hand, cp and (c)hcp bisimilarity provide a better target because we can make use of their place matchings. As explained in Section 6.1, by first analysing the problems of cp and (c)hcp bisimilarity, we ultimately hope to obtain results for hp and (c)hhp bisimilarity.

The remainder of this section falls into five parts. In the first part we formally introduce cp bisimilarity. In the second part we give an alternative characterization of cp bisimilarity which keeps the 'matching history'. This is a prerequisite for the next two parts: in the third part we show that cp bisimilarity is causality preserving, and in the fourth part we define our hereditary (and coherent) version of cp bisimilarity. In the last part we comment on the coincidence and decidability problems of cp and (c)hcp bisimilarity.

cp Bisimilarity. First of all, we translate the above matching requirements into a *compositionality preserving transition system*. It will be based on the domains of *joint processes*, and *joint transitions*:

Definition 6.4.1. The domain of *joint processes* is defined as

$$\mathit{JProc} := \bigcup\{\beta : M_1 \to M_2 \text{ is a bijection} \mid M_1 \in \mathit{Proc}_1 \ \&\ M_2 \in \mathit{Proc}_2\},$$

and the domain of *joint transitions* as

$$\mathit{JT} := T_1 \times T_2.$$

Usually, we let β range over JProc, and t over JT. (It will be clear from the context whether t denotes a joint transition or a standard Petri net transition.)

For $i = 1, 2$ we assume *projection functions* $\mathit{proj}_i : \mathit{JProc} \to \mathit{Proc}_i$, and $\mathit{proj}_i : \mathit{JT} \to T_i$ to recover the ith process, and the ith transition respectively. For JT, proj_i is given via the standard projection functions associated with product domains. For JProc, we transfer the standard projection functions of $P_1 \times P_2$ to the domain $\mathcal{P}(P_1 \times P_2)$: for $\gamma \subseteq P_1 \times P_2$ we define:

$$\mathit{proj}_i(\gamma) := \bigcup\{\mathit{proj}_i(p) \mid p \in \gamma\}.$$

This certainly induces a function $\mathit{proj}_i : \mathit{JProc} \to \mathit{JProc}$.

Convention 6.4.1. Let $\beta \in \mathit{JProc}$. If the ambiguity is resolved by the argument we shall refer to β^{-1} by β.

We associate the following transition relation with *JProc*:

Definition 6.4.2. The *compositionality preserving* (short: *cp*) *transition relation*, $\rightarrow_{cp} \subseteq \mathit{JProc} \times \mathit{JT} \times \mathit{JProc}$, is defined as follows:

$$\beta \xrightarrow{(t_1,t_2)}_{cp} \beta' \text{ iff}$$

1. $\mathit{proj}_i(\beta) \xrightarrow{t_i} \mathit{proj}_i(\beta')$ for $i = 1, 2$,
2. $l(t_1) = l(t_2)$,
3. $\beta({}^\bullet t_1) = {}^\bullet t_2$, and
4. $\beta\lceil_{\mathit{proj}_1(\beta)\setminus {}^\bullet t_1} = \beta'\lceil_{\mathit{proj}_1(\beta)\setminus {}^\bullet t_1}$.

The cp transition relation can also be characterized in the spirit of the Petri net firing rule: executing a joint transition amounts to replacing a joint set of input places with a joint set of output places.

Proposition 6.4.1. *Let $\beta, \beta' \in \mathit{JProc}$, and $t \in \mathit{JT}$. $\beta \xrightarrow{t}_{cp} \beta'$ iff there exists β_{pre}, $\beta_{post} \subseteq P_1 \times P_2$ such that*

1. $\mathit{proj}_i(\beta_{pre}) = {}^\bullet \mathit{proj}_i(t)$ & $\mathit{proj}_i(\beta_{post}) = \mathit{proj}_i(t)^\bullet$ *for* $i = 1, 2$,
2. $\beta_{pre} \subseteq \beta$,
3. $\beta' = \beta \setminus \beta_{pre} \cup \beta_{post}$, *and*
4. $l(\mathit{proj}_1(t)) = l(\mathit{proj}_2(t))$.

Furthermore, β_{pre} is uniquely given by t and β, and β_{post} is uniquely given by t and β'.

Proof. This follows easily from the Petri net firing rule, and Def. 6.4.2. □

We refer to $(\mathit{JProc}, \mathit{JT}, \rightarrow_{cp})$ as the *compositionality preserving* (short: *cp*) *transition system*, and denote it by $\mathcal{T}_{cp}$. Based on $\mathcal{T}_{cp}$ we now define cp bisimilarity:

Definition 6.4.3. A *cp bisimulation* is a relation $\mathcal{C} \subseteq \mathit{JProc}$ that satisfies

1. If $\beta \in \mathcal{C}$ and $\mathit{proj}_1(\beta) \xrightarrow{t_1}$ for some $t_1 \in T_1$, then there are $t_2 \in T_2$, $\beta' \in \mathit{JProc}$ such that $\beta \xrightarrow{(t_1,t_2)}_{cp} \beta'$ and $\beta' \in \mathcal{C}$.

2. Vice versa.

Two processes $M_1 \in Proc_1$, $M_2 \in Proc_2$ are *cp bisimilar w.r.t.* $\beta \in JProc$, written $\beta \in \sim_{cp}$, iff $proj_i(\beta) = M_i$ for $i = 1, 2$, and $\beta \in \mathcal{C}$ for some cp bisimulation $\mathcal{C}$. That is, we define: $\sim_{cp} = \bigcup\{\mathcal{C} \mid \mathcal{C}$ is a cp bisimulation$\}$.

$\mathcal{N}_1$ and $\mathcal{N}_2$ are cp bisimilar iff M_0^1 and M_0^2 are cp bisimilar w.r.t. some $\beta \in JProc$.

Example 6.4.1. It is easy to see that the two systems of Figure 4.8 are cp bisimilar: to construct a cp bisimulation start with the joint process $\{(p_1, p_1'), (p_2, p_2')\}$ and match the transitions according to the hp bisimulation that relates the two systems. The matching of the resulting processes will be dictated by the joint transitions.

cp Bisimilarity with History. We now present an alternative characterization of cp bisimilarity which does not abstract away from the 'matching history'. The new definition will be based on *joint firing sequences* (short: *jfs'*), which constitute the natural notion of run for the cp transition system: in addition to recording the history of joint transitions jfs' also show how past processes were related to each other. For technical reasons, jfs' will be based on *'joint place set'/transition sequences* (short: *jpts'*).

Definition 6.4.4 (jpts'). A *'joint place set'/transition sequence* (short: *jpts*) *(of $\mathcal{N}_1$ and $\mathcal{N}_2$)* is a non-empty sequence $\gamma_0 t_1 \gamma_1 \dots t_n \gamma_n$, where $n \geq 0$, such that $\forall i \in [0, n].\ \gamma_i \subseteq P_1 \times P_2$, and $\forall i \in [1, n].\ t_i \in JT$. We denote the set of jpts' by $JPTS$, and let ϕ range over $JPTS$.

Let $\phi \equiv \gamma_0 t_1 \dots \gamma_n \in JPTS$. We define the *first element of* ϕ as *first*$(\phi) := \gamma_0$, and the *last element of* ϕ as *last*$(\phi) := \gamma_n$.

For $i = 1, 2$, we inductively define a *projection function* $proj_i : JPTS \to \mathcal{P}(P_i) \times T_i^*$ to recover the ith initial set of places, and the ith underlying transition sequence:

$$\begin{aligned} proj_i(\gamma) \quad &= (proj_i(\gamma), \varepsilon), \\ proj_i(\phi\, t\gamma) &= \text{let } (M, w) = proj_i(\phi) \text{ in } (M, w.proj_i(t)). \end{aligned}$$

We use a function $ts : JPTS \to JT^*$ to extract the joint transitions of a jpts in the obvious way.

Let $\mathcal{B} \subseteq JPTS$. We say $\mathcal{B}$ is *prefix-closed* iff $\phi\, t\gamma \in \mathcal{B} \implies \phi \in \mathcal{B}$.

Definition 6.4.5 (jfs'). A jpts $\beta_0 t_1 \beta_1 \dots t_n \beta_n$ is a *joint firing sequence* (short: *jfs*) *(of $\mathcal{N}_1$ and $\mathcal{N}_2$)* iff $\forall i \in [0, n].\ \beta_i \in JProc$, and $\forall i \in [1, n].\ \beta_{i-1} \xrightarrow{t_i}_{cp} \beta_i$. We denote the set of jfs' by JFS, and the set of jfs' starting at β by $JFS(\beta)$. We let σ range over JFS.

We transfer the cp transition relation to jfs' in the obvious way, and define $\rightarrow_{cp} \subseteq JFS \times JT \times JFS$ by:

$$\sigma \xrightarrow{t}_{cp} \sigma' \iff \exists \beta.\ (\sigma' = \sigma\, t \beta \ \&\ last(\sigma) \xrightarrow{t}_{cp} \beta).$$

The following ensures that $proj_i$ on jpts' induces a projection function $proj_i : JFS \rightarrow Runs_i$:

Proposition 6.4.2. *Let $\sigma \in JFS$. For $i = 1, 2$ we have: $proj_i(\sigma) \in Runs_i$.*

Proof. Easy by induction on the length of σ and the definition of $\rightarrow_{cp}$. ☐

Now, we are ready to reformulate cp bisimilarity as a relation of jfs'. To make sure that the jfs' indeed represent 'matching history' we consider only prefix-closed bisimulations.

Definition 6.4.6 (cp bisimilarity on jfs'). A *cp bisimulation* is a *prefix-closed* relation $\mathcal{B} \subseteq JFS$ that satisfies

1. If $\sigma \in \mathcal{B}$ and $proj_1(\sigma) \xrightarrow{t_1}$ for some $t_1 \in T_1$, then there are $t_2 \in T_2$, $\sigma' \in JFS$ such that $\sigma \xrightarrow{(t_1,t_2)}_{cp} \sigma'$ and $\sigma' \in \mathcal{B}$.

2. Vice versa.

Two runs $r_1 \in Runs_1$, $r_2 \in Runs_2$ are *cp bisimilar w.r.t. $\sigma \in JFS$*, written $\sigma \in \sim_{cp}$, iff $proj_i(\sigma) = r_i$ for $i = 1, 2$, and $\sigma \in \mathcal{B}$ for some cp bisimulation $\mathcal{B}$. That is, we define: $\sim_{cp} = \bigcup\{\mathcal{B} \mid \mathcal{B} \text{ is a cp bisimulation}\}$.

$\mathcal{N}_1$ and $\mathcal{N}_2$ are cp bisimilar iff (M_0^1, ε) and (M_0^2, ε) are cp bisimilar w.r.t. some $\sigma \in JFS$.

In the sequel, it will always be clear from the context whether we consider $\sim_{cp}$ and cp bisimulations as sets of joint processes or as sets of jfs'. It is clear that our new definition is equivalent to the original one in the following sense:

Proposition 6.4.3. *Let $\sigma \equiv \beta_0 t_1 \ldots \beta_n \in JFS$. We have:*

$$\sigma \in \sim_{cp} \iff \forall i \in [0, n].\ \beta_i \in \sim_{cp}.$$

Proof. Obvious. ☐

Later, it will be convenient to investigate cp bisimilarity relative to a fixed joint process. For this, we introduce some terminology.

Definition 6.4.7. Let $\beta \in JProc$.

We say a set $\mathcal{B} \subseteq JFS$ is a *cp bisimulation for* β iff $\mathcal{B} \subseteq JFS(\beta)$, $\beta \in \mathcal{B}$, and $\mathcal{B}$ is a cp bisimulation.

We define *cp bisimilarity relative to* β by:

$$\sim_{cp\text{-}\beta} = \bigcup \{\mathcal{B} \mid \mathcal{B} \text{ is a cp bisimulation for } \beta\}.$$

The following connection is immediate:

Proposition 6.4.4. *Let $\beta \in JProc$, and $\sigma, \sigma' \in JFS$ such that $last(\sigma) = \beta = first(\sigma')$.*

1. $\sigma.\sigma' \in \sim_{cp} \implies \sigma' \in \sim_{cp\text{-}\beta}$.

2. $\sigma \in \sim_{cp}$ & $\sigma' \in \sim_{cp\text{-}\beta} \implies \sigma.\sigma' \in \sim_{cp}$.

Proof. Obvious. $\square$

cp Bisimilarity is Causality Preserving. Having defined a notion of run for the cp transition system, we can now see that cp transitions are causality preserving in a very strict sense: each jfs respects the immediate cause dependencies that exist between its transitions (= Petri net boxes). Of course, this immediately implies that cp bisimilarity is partial order preserving.

Proposition 6.4.5. *Let $\sigma \in JFS$, and $(t_1, t_2) \in JT$ such that $\sigma \overset{(t_1,t_2)}{\rightarrow}_{cp}$.*

1. $icauses(proj_1(\sigma), t_1) = icauses(proj_2(\sigma), t_2)$*, and hence*

2. $mcauses(proj_1(\sigma), t_1) = mcauses(proj_2(\sigma), t_2)$.

Proof. (1.) Let σ and (t_1, t_2) be given as above; set $r_i = proj_i(\sigma)$ for $i = 1, 2$. We shall show: $\forall e \in \{init\} \cup [|r_1|].\ e \in icauses(r_1, t_1) \iff e \in icauses(r_2, t_2)$. Clearly, this will prove the proposition.

Assume $e \in icauses(r_1, t_1)$, which means there is $p_1 \in proj_1(last(\sigma))$ such that $p_1 \in {}^\bullet t_1$ and $e = gen(r_1, p_1)$. The latter gives us $p_1 \in (t_1^e)^\bullet$ and $\forall e' \in [e+1, |r_1|]$. $p_1 \notin {}^\bullet(t_1^{e'})$. With the definition of $\rightarrow_{cp}$ it is easy to see that $last(\sigma)(p_1)$ satisfies corresponding properties, so that altogether we can conclude $e \in icauses(r_2, t_2)$ as required. The other direction follows from the symmetric argument.

(2.) is a consequence of (1). $\square$

Naturally, we have:

Proposition 6.4.6. *For all $\sigma \in JFS$, $ts(\sigma) \in SRuns$.*

Proof. This follows since cp transitions are label preserving by definition, and maximal cause preserving by Prop. 6.4.5(2). $\square$

(c)hcp Bisimilarity. In the beginning of this subsection we anticipated that cp bisimilarity is not a truly-concurrent notion. This is illustrated by Example 6.4.1 (or Example 6.4.2 ultimately): like other transitions, b_3 has to be matched dependent on the way in which it is linearized: if b_3 is performed as first transition then it has to be matched against b_2'; on the other hand, if b_3 is computed after the a_2-transition then it has to be matched against b_1'. To ensure that the matching is done in a truly-concurrent fashion, we can impose a (padding and) backtracking condition on cp bisimilarity, and thereby strengthen our notion to (*coherent* and) *hereditary cp* (short: *(c)hcp*) *bisimilarity.* We first define backtracking and hcp bisimulation, and then furthermore padding and chcp bisimulation.

In our standard definition of hhp bisimilarity, we implemented the backtracking requirement compactly as the 'joint' backtracking of pairs of transitions in synchronous runs. The following ensures that we can proceed analogously in the framework of jfs':

Proposition 6.4.7. *Let $\beta_0 t_1 \beta_1 \ldots t_n \beta_n \in JFS$. For all $i \in [1, n]$ we have:*

$$\forall j \in [i+1, n].\ proj_1(t_j)\ I_1\ proj_1(t_i) \iff \forall j \in [i+1, n].\ proj_2(t_j)\ I_2\ proj_2(t_i).$$

Proof. This follows from Prop. 6.4.6. □

Thus, we define:

Definition 6.4.8 (I, BEn). We associate an *independence relation $I \subseteq JT \times JT$* with $\mathcal{N}_1$ and $\mathcal{N}_2$ in the following way:

$$t\ I\ t' \iff (proj_1(t)\ I_1\ proj_1(t'))\ \&\ (proj_2(t)\ I_2\ proj_2(t')).$$

Let $\sigma \equiv \beta_0 t_1 \beta_1 \ldots t_n \beta_n \in JFS$, and $t \in JT$. t is *backtrack enabled in* σ, written $t \in BEn(\sigma)$, iff there is $i \in [1, n]$ such that $t_i = t$, and $\forall j \in [i+1, n].\ t_j\ I\ t_i$.

Having defined BEn we still need to define the result of backtracking; in other words we need to transfer the backtrack function δ to our framework of jfs'. δ on jfs' is more involved: in addition to removing the backtracked transition from the respective jfs we also have to make sure that the joint processes over which it is backtracked are adjusted correspondingly. Here Prop. 6.4.1 comes in useful: executing a joint transition in the cp transition system amounts to replacing a set of joint input places with a set of joint output places.

Definition 6.4.9. Let $\beta \in JProc$, and $t \in JT$.

Let $\beta \xrightarrow{t}$. We call the uniquely given β_{pre} of Prop. 6.4.1 the *set of joint input places of t w.r.t. β*, and denote it by $preSet(\beta, t)$.

Let $\beta' \xrightarrow{t} \beta$ for some β'. Analogously, we call the uniquely given β_{post} of Prop. 6.4.1 the *set of joint output places of* t *w.r.t.* β, and denote it by $postSet(\beta, t)$.

Thus, when backtracking a joint transition t in a jfs we naturally adapt the intermediate joint processes by replacing t's joint output places with its joint input places. To formalize this the following operations will be helpful:

Definition 6.4.10. Let $\phi, \phi' \in JPTS$. If $last(\phi) = first(\phi')$, we define the *concatenation of* ϕ *and* ϕ' as

$$\phi.\phi' := \gamma_0 t_1 \ldots \gamma_n t'_1 \gamma'_1 \ldots \gamma'_m,$$

where $\phi = \gamma_0 t_1 \ldots \gamma_n$, and $\phi' = \gamma_n t'_1 \gamma'_1 \ldots \gamma'_m$.

Let γ, $\gamma_{old}, \gamma_{new} \subseteq P_1 \times P_2$. If $\gamma_{old} \subseteq \gamma$ and $\gamma_{new} \cap (\gamma \backslash \gamma_{old}) = \emptyset$, we define the *substitution of* γ_{new} *for* γ_{old} *in* γ as

$$\gamma[\gamma_{new} \backslash \gamma_{old}] := \gamma \backslash \gamma_{old} \cup \gamma_{new}.$$

We generalize this operation to jpts', and inductively define:

$$\begin{array}{ll} \gamma[\gamma_{new} \backslash \gamma_{old}] & := \gamma[\gamma_{new} \backslash \gamma_{old}] \text{ (substitution for } P_1 \times P_2 \text{ as above),} \\ (\phi\, t \gamma)[\gamma_{new} \backslash \gamma_{old}] & := (\phi[\gamma_{new} \backslash \gamma_{old}])\, t (\gamma[\gamma_{new} \backslash \gamma_{old}]). \end{array}$$

The following proposition ensures: (1.) concatenation on jpts' induces concatenation on jfs'; (2.) if applied in the way we intend, the substitution function will always be defined, and return a jfs.

Proposition 6.4.8.

1. *Let* $\sigma, \sigma' \in JFS$ *with* $last(\sigma) = first(\sigma')$. *Then* $\sigma.\sigma' \in JFS$.

2. *Let* $\beta t.\sigma \in JFS$ *with* $t\ I\ ts(\sigma)$. *Then* $\sigma_s := \sigma[preSet(\beta, t) \backslash postSet(first(\sigma), t)]$ *is defined, and we have:* $\sigma_s \in JFS$ & $first(\sigma_s) = \beta$.

Proof. This follows easily by applying the definitions. □

It is now straightforward how to define δ:

Definition 6.4.11 (δ). Let $\sigma \equiv \beta_0 t_1 \beta_1 \ldots t_n \beta_n \in JFS$, and $t \in BEn(\sigma)$.

In the following, we use $last(\sigma, t)$ to denote the position of the last occurrence of t in σ. That is, $last(\sigma, t) = i$ iff $t_i = t$ and $t_j \neq t$ for all $j \in [i+1, n]$.

We define $\delta(\sigma, t)$ to be the result of backtracking t in σ, that is $\delta(\sigma, t) := \sigma'.\sigma''$, where $\sigma' = \beta_0 t_1 \ldots \beta_{i-1}$, and $\sigma'' = (\beta_i t_{i+1} \ldots \beta_n)[preSet(\beta_{i-1}, t) \backslash postSet(\beta_i, t)]$ for $i = last(\sigma, t)$. By Prop. 6.4.8 it is clear that the operations are defined, and we have $\delta(\sigma, t) \in JFS$.

Finally, we are ready for the definition of hcp bisimulation:

Definition 6.4.12 (hcp bisimulation). A cp bisimulation $\mathcal{B} \subseteq \mathit{JFS}$ is *hereditary* (short: h) when it further satisfies

3. If $\sigma \in \mathcal{B}$ and $t \in \mathit{BEn}(\sigma)$ for some $t \in \mathit{JT}$, then $\delta(\sigma, t) \in \mathcal{B}$.

Having already defined backtracking and hcp bisimulation we can now define padding and chcp bisimulation more succinctly. Analogously to chhp bisimilarity, we impose padding whenever there is a fork between a new joint transition $t\beta$ and an alternative continuation σ' such that $t\beta$ and σ' are independent of each other. The notion of independence will be the one of Def. 6.4.8, which implies we only demand padding if both, $\mathit{proj}_1(t)$ and $\mathit{proj}_2(t)$ are independent of the transitions in $\mathit{proj}_1(\sigma')$, and $\mathit{proj}_2(\sigma')$ respectively. The following makes sure our definition of padding will be sound:

Proposition 6.4.9. *Let σ', $\beta' t\beta \in \mathit{JFS}$ such that $\mathit{first}(\sigma') = \beta'$ and $t \ I \ \mathit{ts}(\sigma')$. Then $\sigma'_s := \sigma'[\mathit{postSet}(\beta, t) \backslash \mathit{preSet}(\beta', t)]$ is defined, and we have: $\sigma'_s \in \mathit{JFS}$ & $\mathit{first}(\sigma'_s) = \beta$.*

Proof. As above this follows easily by applying the definitions. □

Now, we define:

Definition 6.4.13 (padding). Let $\sigma.\sigma'$, $\sigma\, t\beta \in \mathit{JFS}$ such that $t \ I \ \mathit{ts}(\sigma')$. Then we can *pad $t\beta$ between σ and σ'*, and define the result as

$$\zeta(\sigma, t\beta, \sigma') := \sigma\, t\beta.(\sigma'[\mathit{postSet}(\beta, t) \backslash \mathit{preSet}(\mathit{last}(\sigma), t)]).$$

By Prop. 6.4.9 and 6.4.8(1) it is clear that the operations are defined, and we have $\zeta(\sigma, t\beta, \sigma') \in \mathit{JFS}$.

And hence, chcp bisimulation is given as follows:

Definition 6.4.14 (chcp bisimulation). A hcp bisimulation $\mathcal{B} \subseteq \mathit{JFS}$ is *coherent* (short: c) when it further satisfies

4. If $\sigma.\sigma'$, $\sigma\, t\beta \in \mathcal{B}$ such that $t \ I \ \mathit{ts}(\sigma')$ then $\zeta(\sigma, t\beta, \sigma') \in \mathcal{B}$.

Finally, we define (c)hcp bisimilarity:

Definition 6.4.15 ((c)hcp bisimilarity). Two runs $r_1 \in \mathit{Runs}_1$, $r_2 \in \mathit{Runs}_2$ are *(c)hcp bisimilar w.r.t. $\sigma \in \mathit{JFS}$*, written $\sigma \in \sim_{(c)hcp}$, iff $\mathit{proj}_i(\sigma) = r_i$ for $i = 1, 2$, and $\sigma \in \mathcal{B}$ for some (c)hcp bisimulation $\mathcal{B}$. That is, we define:

$$\sim_{(c)hcp} = \bigcup\{\mathcal{B} \mid \mathcal{B} \text{ is a (c)hcp bisimulation}\}.$$

$\mathcal{N}_1$ and $\mathcal{N}_2$ are (c)hcp bisimilar iff (M_0^1, ε) and (M_0^2, ε) are (c)hcp bisimilar w.r.t. some $\sigma \in JFS$.

Example 6.4.2. The systems of Example 6.4.1 are not hcp bisimilar just as they are not hhp bisimilar.

The Coincidence and Decidability Problem(s) of cp and (c)hcp Bisimilarity. Together, Example 6.4.1 and 6.4.2 clearly show:

Fact 6.4.1. *cp and (c)hcp bisimilarity do not coincide in general; they do not even coincide for free choice systems.*

The decidability problem of cp bisimilarity is also readily resolved: two finite-state systems have only finitely many joint processes, and hence cp bisimilarity can be decided by exhaustive search.

Fact 6.4.2. *It is decidable whether two finite-state systems are cp bisimilar.*

On the other hand, the decidability problem of (c)hcp bisimilarity remains open: we face similar difficulties as for (c)hhp bisimilarity.

6.4.2 Relation to hp and (c)hhp Bisimilarity

Since cp bisimilarity is partial order preserving (Prop. 6.4.5) it is straightforward to transform any cp bisimulation $\mathcal{B} \subseteq JFS$ into a hp bisimulation $\mathcal{H}$: simply set $\mathcal{H} := ts(\mathcal{B})$. The transformation preserves the hereditary property, and for cp bisimulations that satisfy a natural minimality criterion also the coherent property. The minimality criterion can always be achieved, and thus altogether we have:

Fact 6.4.3.

1. *cp bisimilarity implies hp bisimilarity.*
2. *(c)hcp bisimilarity implies (c)hhp bisimilarity.*

It is immediately clear that the opposite implications do not hold: hp, hhp, and chhp bisimilarity are all behavioural notions; they abstract away from places, and hence they are not capable of reflecting compositionality in the sense of cp bisimilarity. This is demonstrated by the example of Figure 6.7: the three systems implement exactly the same behaviour, and are clearly chhp bisimilar; on the other hand, they are not cp bisimilar. The systems are already free choice; even more, by means of a loop-back transition they could easily be transferred into a counter-example of *live* fc systems.

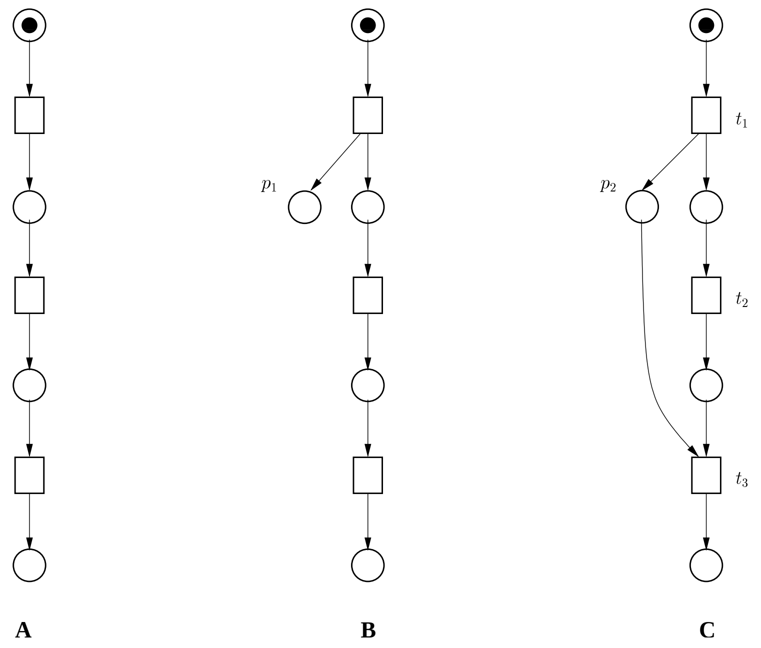


Figure 6.7: A trivial counter-example

Fact 6.4.4. *chhp bisimilarity does not imply cp bisimilarity, not even for live fc systems.*

On the other hand, we could argue that counter-examples that rely on behaviourally irrelevant places merely highlight *technical inaccuracies*, and that these might be overcome by employing a more sophisticated version of cp bisimilarity. For example, we could allow that places can be matched to a designated 'nil' place, and thereby create the opportunity to 'submerge' behaviourally irrelevant place occurrences within a bisimulation. We would hope that a decidability or coincidence result for the original cp bisimilarity could be carried over to such a new version. It is clear that handling the technical inaccuracies will not automatically give us coincidence between the hp and the cp bisimilarities. Indeed, there remains a more essential issue to be resolved, the open problem of *pending synch places*. We will explain this issue in Section 6.14.1, where we discuss it in the context of bp bisimilarity.

6.4.3 Further Concepts and Observations

We shall now attend to two technical issues. Firstly, we investigate whether the free choice types switch and synch are preserved by the matching in cp bisimulations. Secondly, we introduce a shuffle operation for jfs'.

cp Bisimilarity on Free Choice Systems. For this paragraph, assume $\mathcal{N}_1$ and $\mathcal{N}_2$ to be free choice. It is not difficult to see that the cp transition system, and hence cp bisimilarity, respects the types of transitions:

Proposition 6.4.10.

1. *Let $\beta \in JProc$, and $t \in JT$ such that $\beta \xrightarrow{t}_{cp}$.*

   $proj_1(t)$ *is of type switch (synch)* $\iff$ $proj_2(t)$ *is of type switch (synch).*

2. *Let $\beta_0 t_1 \beta_1 \ldots t_n \beta_n \in JFS$. For all $i \in [1, n]$ we have:*

   $proj_1(t_i)$ *is of type switch (synch)* $\iff$ $proj_2(t_i)$ *is of type switch (synch).*

Proof. (1.) Let β and $t \equiv (t_1, t_2)$ be given as above. For $i \in \{1, 2\}$ t_i is of type switch iff $|{}^\bullet t_i| = 1$ (t_i is of type synch iff $|{}^\bullet t_i| > 1$). Then (1) is immediate since by definition of $\rightarrow_{cp}$ there must be a bijection between ${}^\bullet t_1$ and ${}^\bullet t_2$.

(2.) is a consequence of (1). ☐

This is not always given for places (e.g. consider the jfs' in the cp bisimulation of Example 6.4.1). But it will hold if one of two related places (and hence both) contributes to enabling a joint transition:

Proposition 6.4.11. *Let $\beta \in JProc$, $(p_1, p_2) \in \beta$, and $(t_1, t_2) \in JT$ such that $\beta \xrightarrow{(t_1,t_2)}_{cp}$ and $p_i \in {}^\bullet t_i$ for $i = 1$, or 2.*

p_1 *is of type switch (synch)* $\iff$ p_2 *is of type switch (synch).*

Proof. Let entities be given as above, and w.l.o.g. assume $i = 1$. Clearly, for $j \in \{1, 2\}$ the places of ${}^\bullet t_j$ are of type switch (synch) iff t_j is of type switch (synch). But then the proposition easily follows from Prop. 6.4.10(1) and since by definition of $\rightarrow_{cp}$ we must have $p_2 \in {}^\bullet t_2$. ☐

If a system is live then every place occurrence will contribute to enabling a transition at some future point. Whenever a place is matched in a cp bisimulation this case will be anticipated, and with the previous proposition it is clear that in cp bisimilarity on live fc systems the types of places are indeed respected.

Proposition 6.4.12. *Assume $\mathcal{N}_1$ and $\mathcal{N}_2$ to be live.*

1. *Let $\beta \in \sim_{cp}$. For all $(p_1, p_2) \in \beta$ we have:*

   p_1 *is of type switch (synch)* $\iff$ p_2 *is of type switch (synch).*

2. *Let $\beta_0 t_1 \beta_1 \ldots t_n \beta_n \in \sim_{cp}$. For all $i \in [0, n]$, and $(p_1, p_2) \in \beta_i$ we have:*

   p_1 *is of type switch (synch)* $\iff$ p_2 *is of type switch (synch).*

Proof. (1.) Let β and (p_1, p_2) be given as above. Since $\mathcal{N}_1$ is live we can choose $t_1 \in p_1{}^\bullet$ and $r_1 \in Runs_1(proj_1(\beta))$ such that (a) $r_1[t_1\rangle$, and (b) $\nexists t_1' \in r_1.\ p_1 \in {}^\bullet t_1'$ (recall Fact 6.2.2).

Certainly, $\sim_{cp\text{-}\beta}$ contains a match for $r_1.t_1$; this means we can assume $\sigma\, t\beta' \in \sim_{cp\text{-}\beta}$ such that $proj_1(\sigma) = r_1$ and $proj_1(t) = t_1$. Considering the cp transition function and property (b) of r_1 it is easy to see that we must have $(p_1, p_2) \in last(\sigma)$. But then (1) immediately follows from Prop. 6.4.11.

(2.) Considering Prop. 6.4.3 this is easily seen to be a consequence of (1). □

Justified by Prop. 6.4.10 and 6.4.12 we adopt the following convention:

Convention 6.4.2. Let $t \equiv (t_1, t_2)$ be a joint transition that occurs in some jfs. We classify t as type switch (synch) iff t_1, or equivalently t_2, is of type switch (synch). Let $p \equiv (p_1, p_2) \in \beta$, where β is a joint process that occurs in $\sim_{cp}$ on live fc systems. We classify p as type switch (synch) iff p_1, or equivalently p_2, is of type switch (synch).

Finally, the following related observation will be useful for later:

Proposition 6.4.13. *Let $\beta \in JProc$, $(p_1, p_2) \in \beta$ such that p_i is of type synch, and $(t_1, t_2) \in JT$ such that $p_i{}^\bullet = \{t_i\}$ and $\beta \overset{(t_1,t_2)}{\rightarrow}$, where $i = 1$, or 2.*

1. *$p_{\bar{i}}$ is of type synch, and*
2. *$SPartners(p_{\bar{i}}) = \beta(SPartners(p_i))$.*

Proof. Let entities be given as above. (1) is immediate by Prop. 6.4.11. Then (2) easily follows when considering that by definition of $\rightarrow_{cp}$ we have: $p_{\bar{i}} \in {}^\bullet t_{\bar{i}}$ and ${}^\bullet t_{\bar{i}} = \beta({}^\bullet t_i)$. □

Shuffle for jfs'. We now define a shuffle operation for jpts', and hence jfs'. Since we have to take care of the 'joint place sets' the definition will be more involved than the one of shuffle for words (cf. Def. 2.5.3).

Definition 6.4.16. The *shuffle* of two jpts' $\phi^\star \equiv \gamma_0^\star t_1^\star \gamma_1^\star \ldots t_n^\star \gamma_n^\star$, and $\phi^\circ \equiv \gamma_0^\circ t_1^\circ \gamma_1^\circ \ldots t_m^\circ \gamma_m^\circ$ is the set $\phi^\star \otimes \phi^\circ$ of all words of the form

$$\phi_1^\star.\phi_1^\circ.\phi_2^\star.\phi_2^\circ \ldots \phi_k^\star.\phi_k^\circ,$$

where $k \geq 1$, and there exist functions $e^\star : [0,k] \rightarrow [0,n]$, $e^\circ : [0,k] \rightarrow [0,m]$ such that $e^\star(0) = 0 = e^\circ(0)$, both $e^\star$ and e° are monotonic ($\forall a, b \in [0,k].\ a \leq b \Rightarrow$

$e^\star(a) \leq e^\star(b)$, and similarly for e°), and for $j \in [1,k]$ we have:

$$\begin{aligned}
\phi_j^\star &= (\gamma^\circ_{e^\circ(j-1)} \cup \gamma^\star_{e^\star(j-1)})\, t^\star_{e^\star(j-1)+1}\, (\gamma^\circ_{e^\circ(j-1)} \cup \gamma^\star_{e^\star(j-1)+1})\, t^\star_{e^\star(j-1)+2} \cdots \\
&\qquad \cdots t^\star_{e^\star(j)}\, (\gamma^\circ_{e^\circ(j-1)} \cup \gamma^\star_{e^\star(j)}), \\
\phi_j^\circ &= (\gamma^\star_{e^\star(j)} \cup \gamma^\circ_{e^\circ(j-1)})\, t^\circ_{e^\circ(j-1)+1}\, (\gamma^\star_{e^\star(j)} \cup \gamma^\circ_{e^\circ(j-1)+1})\, t^\circ_{e^\circ(j-1)+2} \cdots \\
&\qquad \cdots t^\circ_{e^\circ(j)}\, (\gamma^\star_{e^\star(j)} \cup \gamma^\circ_{e^\circ(j)}).
\end{aligned}$$

The *shuffle* of two sets $L_1,\, L_2 \subseteq JPTS$ is the set

$$L_1 \otimes L_2 = \bigcup_{\phi_1 \in L_1,\, \phi_2 \in L_2} \phi_1 \otimes \phi_2.$$

Naturally, we have:

Proposition 6.4.14. *Let $L_1,\, L_2 \subseteq JPTS$. If L_1 and L_2 are prefix-closed then $L_1 \otimes L_2$ is also prefix-closed.*

Proof. This is easy to see from the definition. □

And analogously to shuffle on (joint) transition sequences (cf. Prop 2.5.2, 2.5.3) we obtain:

Proposition 6.4.15. *Let $\beta \in JProc$, and $\beta_s,\, \beta_s' \subseteq \beta$ such that $\beta_s \cap \beta_s' = \emptyset$. Then for any $L \subseteq JFS(\beta_s)$, $L' \subseteq JFS(\beta_s')$ we have: $L \otimes L' \subseteq JFS(\beta_s \cup \beta_s')$.*

Proof. Straightforward. □

6.4.4 Proof Methodology

In preparation, we now present two insights, which will provide important proof methodology in our quest for the coincidence between cp and chcp bisimilarity on live fc systems. The insights directly correspond to two points of our general approach (cf. Section 6.1.1(3)).

A General Insight. Let x bisimilarity be any notion of bisimilarity that has a hereditary and coherent version. Inspired by Section 3.3 we obtain a *(1)hereditary* and *(1)coherent* version of x bisimilarity by imposing backtracking over only one transition, and an analogously restricted version of padding. We will make use of the following insight: if x bisimilarity coincides with its (1)hereditary and (1)coherent version, in the strict sense that *any* x bisimulation can be extended to a (1)hereditary and (1)coherent one, then any x bisimulation can furthermore be extended to one that is fully hereditary and coherent. This is so because the largest x bisimulation satisfies a simple closure property; e.g. for cp bisimilarity we have: let $\sigma.\sigma_c$, $\sigma' \in \sim_{cp}$; if σ and σ' are the same up to reshuffling of independent transitions then $\sigma'.\sigma_c \in \sim_{cp}$. For us this means: we can establish the coincidence between cp and chcp bisimilarity by proving that $\sim_{cp}$ is (1)hereditary and (1)coherent. Formally, we define and formulate:

Definition 6.4.17. A set $\mathcal{B} \subseteq JFS$ can satisfy the following properties:

- **(1)hereditary** (short: **(1)her**): $\sigma\, t\beta\, t'\beta' \in \mathcal{B}$ & $t\ I\ t' \implies \sigma\, t'\beta'_s \in \mathcal{B}$, where $\beta'_s = \beta'[preSet(last(\sigma), t) \backslash postSet(\beta, t)]$.

- **(1)coherent** (short: **(1)coh**): $\sigma\, t'\beta'$, $\sigma\, t\beta \in \mathcal{B}$ & $t\ I\ t' \implies \sigma\, t\beta\, t'\beta'_s \in \mathcal{B}$, where $\beta'_s = \beta'[postSet(\beta, t) \backslash preSet(last(\sigma), t)]$.

Lemma 6.4.1 (proof method I).

$$\sim_{cp} \models \textbf{(1)her} \ \& \ \textbf{(1)coh} \quad \implies \quad \sim_{cp} = \sim_{chcp}.$$

Lemma 6.4.1 is formally shown for hp bisimilarity in Appendix C.2; the proof carries over in a straightforward manner to cp bisimilarity. Appendix C.2 also shows that it is possible to separately achieve '$\sim_{cp} \models$ **(1)her** $\Rightarrow \sim_{cp} \models$ **her**', and '$\sim_{cp} \models$ **(1)coh** $\Rightarrow \sim_{cp} \models$ **coh**'.

Divide and Conquer. The second insight is trivial but it provides us with an important method of further structuring the coincidence problem of cp and chcp bisimilarity on live fc systems. Assume $\mathcal{N}_1$ and $\mathcal{N}_2$ to be free choice. From Section 6.4.3 we know that the joint transitions of jfs' can be classified into type switch or synch. Accordingly, we can split the properties (1)hereditary and (1)coherent into two parts, and establish '$\sim_{cp} \models$ **(1)her** & **(1)coh**' by separately proving that on the one hand cp bisimilarity is *sw-(1)hereditary* and *sw-(1)coherent*, and on the other hand *sy-(1)hereditary* and *sy-(1)coherent*.

Definition 6.4.18. A set $\mathcal{B} \subseteq JFS$ can satisfy the following properties:

- **sw-(1)hereditary**(short: **sw-(1)her**):
  $\sigma\, t\beta\, t'\beta' \in \mathcal{B},\ t\ I\ t'$ & t' is of type switch $\implies \sigma\, t'\beta'_s \in \mathcal{B}$,
  where $\beta'_s = \beta'[preSet(last(\sigma), t) \backslash postSet(\beta, t)]$.

- **sy-(1)hereditary** (short: **sy-(1)her**):
  $\sigma\, t\beta\, t'\beta' \in \mathcal{B},\ t\ I\ t'$ & t' is of type synch $\implies \sigma\, t'\beta'_s \in \mathcal{B}$,
  where $\beta'_s = \beta'[preSet(last(\sigma), t) \backslash postSet(\beta, t)]$.

- **sw-(1)coherent** (short: **sw-(1)coh**):
  $\sigma\, t'\beta',\ \sigma\, t\beta \in \mathcal{B},\ t\ I\ t'$ & t' is of type switch $\implies \sigma\, t\beta\, t'\beta'_s \in \mathcal{B}$,
  where $\beta'_s = \beta'[\mathit{postSet}(\beta, t)\backslash \mathit{preSet}(\mathit{last}(\sigma), t)]$.

- **sy-(1)coherent** (short: **sy-(1)coh**):
  $\sigma\, t'\beta',\ \sigma\, t\beta \in \mathcal{B},\ t\ I\ t'$ & t' is of type synch $\implies \sigma\, t\beta\, t'\beta'_s \in \mathcal{B}$,
  where $\beta'_s = \beta'[\mathit{postSet}(\beta, t)\backslash \mathit{preSet}(\mathit{last}(\sigma), t)]$.

Lemma 6.4.2 (proof method II). *Let $\mathcal{B} \subseteq \mathit{JFS}$.*

$$\mathcal{B} \models \textbf{sw-(1)her},\ \textbf{sw-(1)coh},\ \textbf{sy-(1)her}\ \&\ \textbf{sy-(1)coh} \\ \implies \mathcal{B} \models \textbf{(1)her}\ \&\ \textbf{(1)coh}.$$

It is important to note that we split the properties according to the type of t' rather than t. Our motivation for this will become clear later on (cf. Section 6.11).

6.5 Interlude I

We have now arrived at Part II, which, apart from this interlude, comprises the following six sections, up to Interlude II. As explained in Section 6.1.2, the six sections provide the modules that together prove our two main results on live SSM-decomposable systems: cp bisimilarity is K-decomposable, and sw-(1)coherent and sw-(1)hereditary. In detail, the material is organized as follows:

1. (Section 6.6) Building on Section 6.2.4 we refine our insight into the behaviour of live SSMD fc systems. In particular, we introduce a setting in which we assume fixed *courses* for a subset of components.

2. (Section 6.7) We introduce the two new topological entities for free choice nets, *links* and *wedges*. With the help of the behavioural concepts of (1.) we will prove important properties about them. This will culminate in the *WNL Theorem*, which reveals a structural constraint of links and wedges in live SSMD fc systems.

3. (Section 6.8) We investigate the cp transition system in the context of SSMD fc systems. This will provide us with insights about the matching in cp bisimilarity, which will be required in the following three parts.

4. (Section 6.9) We introduce the property of *K-decomposability*, and provide a first analysis into whether cp bisimilarity indeed satisfies this decomposition property. In particular, we are led to the *Crucial Subgoal^K*, and sketch how it will follow if we can prove that *swfsi-matching* is deterministic. Throughout the analysis we rely on observations of (3.).

5. (Section 6.10) Motivated by (4.) we set out to prove that for live SSMD fc systems swfsi-matching in cp bisimilarity is deterministic. We indeed achieve this result, and call it the *SWFSI Matching Theorem*. The WNL Theorem delivers the crucial ingredient to the proof while its application is made possible by an observation of (3.).

6. (Section 6.11) With the result of (5.) we can now implement our plan of (4.): we prove the Crucial Subgoal^K, and conclude that cp bisimilarity on live SSMD fc systems is indeed K-decomposable. The Crucial Subgoal^K will have a second consequence: with its help we additionally achieve that cp bisimilarity on live SSMD fc systems is sw-(1)coherent and sw-(1)hereditary.

An overview of this structure can be found in Figure 6.1.

6.6 On the Behaviour of Live SSMD FC Systems

In this section we will utilize and refine our knowledge about the behaviour of live SSM-decomposed fc systems. The newly developed concepts and results will be employed in the next section to prove the WNL Theorem.

Recall that in Section 6.2.4 we derived the following two important facts: (1.) In live fc systems the computations of SM-components are unconstrained by their context (Theorem 6.2.2). (2.) The SM-components of SSM-decomposed fc systems can take their decisions in full freedom. We shall translate (2.) into action, and grant components a will of their own: we will define a setting where we assume fixed *courses* for a subset of components. As we will see, in this setting (1.) is no longer valid: components can become *frozen*, and may never be able to complete their computation. We will, however, obtain a generalized version of Theorem 6.2.2.

We proceed as follows: first of all, we formalize the concept of waiting. Secondly, we give the definitions for the setting with courses. In a third part, we will see how components can become frozen by a course, and finally we present the generalization of Theorem 6.2.2. Throughout this section we assume an SSM-decomposed fc system $(\mathcal{N}, Cover)$. If we require $\mathcal{N}$ to be live we will denote it by $\mathcal{N}_L$. For the examples consider the system of Figure 6.5.

Waiting Orders. As we saw in Section 6.2.4.1 the SM-components of a fc system may have to wait at synch places for their synchronization partners to get ready. In the context of a SSM-decomposition there will be a characteristic scenario of 'waiting dependencies' between the components at each reachable marking. We formally capture this by defining the following relation:

Definition 6.6.1. Let $M \in Reach(\mathcal{N})$, and $K \in Cover$.

Let $p \in P_N$. We say K *is directly waiting for* p *at* M, denoted by $p \lhd_M K$, iff

1. $M(K)$ is a synch place with $p \in SPartners(M(K))$, and
2. $p \notin M$.

Let $K' \in Cover$. We say K *is directly waiting for* K' *at* M, denoted by $K' \prec_M K$, iff there is $p \in P_{K'}$ such that $p \lhd_M K$. If we want to specify p (which is uniquely given by Prop. 6.2.2(1a)) we will write $K'(p) \prec_M K$.

We denote the transitive closure of $\prec_M$ by $\prec\!\prec_M$, and call it the *waiting order at* M. If $K' \prec\!\prec_M K$ then we say K *is waiting for* K' *at* M. We will write $K'(p) \prec\!\prec_M K$ if we want to specify a synch place of K' for which K is waiting (note that there could be several).

We say K *is waiting at* M iff $K' \prec\!\prec_M K$ for some K'.

Example 6.6.1. Let $M = \{p_1, p_6, p_{12}, p_{14}\}$. We have $K_1(p_3) \prec_M K_2 \begin{array}{l} (p_{10}) \prec_M K_3 \\ (p_8) \prec_M K_4. \end{array}$

Indeed, we have:

Proposition 6.6.1. *Let $M \in Reach(\mathcal{N})$, $K, K' \in Cover$, and $p \in P_{K'}$ such that $K'(p) \prec\!\prec_M K$. Further, let $r \equiv t_1 t_2 \dots t_n \in Runs_N(M)$, and M' such that $M[r\rangle M'$.*

1. (a) *K is inactive at least until K' performs some transition:*
   $$\forall i \in [1,n].\ (t_i \in T_K \implies \exists j \in [1, i-1].\ t_j \in T_{K'}),$$
   and more exactly:

   (b) *K is inactive at least until K' performs a p-enabling transition:*
   $$\forall i \in [1,n].\ (t_i \in T_K \implies \exists j \in [1, i-1].\ t_j \in T_{K'}\ \&\ p \in t_j{}^\bullet).$$

2. (a) *K stays waiting for K' at least until K' performs some transition:*
   $$(\forall i \in [1,n].\ t_i \notin T_{K'}) \implies K' \prec\!\prec_{M'} K,$$
   and more exactly:

   (b) *K stays waiting for K' at least until K' performs a p-enabling transition:*
   $$(\forall i \in [1,n].\ \neg(t_i \in T_{K'}\ \&\ p \in t_i{}^\bullet)) \implies K'(p) \prec\!\prec_{M'} K.$$

Proof. (1.) We only need to prove the stronger clause (b). If $K'(p) \prec\!\prec_M K$ then there is a chain $K'(p) \prec_M K_1(p_1) \prec_M \dots K_n(p_n) \prec_M K$. The property can easily be proved by induction on the length of this chain. To prove the inductive step one employs the definition of $\prec_M$, Obs. 6.2.5(1,2), and the fact that if $p \in P_{K'}$ then all p-enabling transitions ($t \in T_N$ with $p \in t^\bullet$) are contained in $T_{K'}$ (this follows from Prop. 6.2.2(2b)).

(2.) Again it is sufficient to prove (b). Assume $K'(p) \prec\!\prec_M K$, and with it a chain $K'(p) \prec_M K_1(p_1) \prec_M \dots K_{n-1}(p_{n-1}) \prec_M K_n \equiv K$. Further, let $M[r\rangle M'$ such that r does not contain any p-enabling K'-transition. By Prop. 6.2.9(2) we obtain $M(K') = M'(K')$, and by additionally employing (1.) of the current proposition $M(K_i) = M'(K_i)$ for all $i \in [1,n]$. It is then clear that the same waiting chain is also present at M', and hence we have $K'(p) \prec\!\prec_{M'} K$. ☐

For SSM-decomposed fc systems in general, $\prec\!\prec_M$ can be reflexive, which expresses that components may be deadlocked. For *live* SSM-decomposed fc systems, $\prec\!\prec_M$ is always a strict order:

Proposition 6.6.2. *For all $M \in Reach(\mathcal{N}_L)$, $\prec\!\prec_M$ is a strict order.*

Proof. Let $M \in Reach(\mathcal{N}_L)$. We only have to prove that $\prec\!\prec_M$ is irreflexive. To the contrary assume there is $K \in Cover$ with $K \prec\!\prec_M K$. Then by Prop. 6.6.1(1) K is deadlocked; it can never do any action again. But this is a clear contradiction to liveness (Prop. 6.2.11). □

In our context of finite-state systems this implies:

Proposition 6.6.3. *Let $M \in Reach(\mathcal{N}_L)$. There is at least one component $K \in Cover$ which is not waiting at M.*

Proof. Follows from Prop. 6.6.2 and the fact that strict orders on finite sets have at least one minimal element. □

Components on Courses. In Section 6.2.4.2 we explained that the components of SSM-decomposed fc systems can take their decisions in full freedom. As promised, we now translate this fact into action, and grant components a will of their own: we allow local computation paths for a subset of components to be fixed, and require that global computations conform to these component *courses.*

Definition 6.6.2. Let $M \in Reach(\mathcal{N})$.

Let $K \in Cover$. A *course for K* (short: *K-course*) *at M* is an infinite K-computation[5] γ_K at $M(K)$. Let $\mathcal{K} \subseteq Cover$. A *course for $\mathcal{K}$* (short: *$\mathcal{K}$-course*) *at M* is a family $\mathcal{C} = \{\gamma_K^{\mathcal{C}}\}_{K\in\mathcal{K}}$, where each $\gamma_K^{\mathcal{C}}$ is a K-course at M.

For the next two items let $\mathcal{K} \subseteq Cover$ and $\mathcal{C}$ be a $\mathcal{K}$-course at M.

Let $r \in Runs_N(M)$. We say *r conforms to $\mathcal{C}$* iff for all $K \in \mathcal{K}$ we have: $r\!\uparrow\! T_K \in FinPrefixes(\gamma_K^{\mathcal{C}})$. We denote the set of runs of M conforming to $\mathcal{C}$ by $CRuns_N(M, \mathcal{C})$.

Let $r \in CRuns_N(M, \mathcal{C})$. We define the *continuation course of $\mathcal{C}$ after r* to be:

$$cont(\mathcal{C}, r) = \{\gamma_K^{cont}\}_{K\in\mathcal{K}}, \text{ where } \gamma_K^{cont} = \gamma_K^{\mathcal{C}} - r\!\uparrow\! T_K.$$

The following proposition complements the definition; in particular the first part shall be used implicitly later on.

[5] Recall Def. 6.2.11

Proposition 6.6.4. *Let $M \in Reach(\mathcal{N})$, $\mathcal{K} \subseteq Cover$, and $\mathcal{C}$ a $\mathcal{K}$-course at M. Further let $r \in CRuns_N(M, \mathcal{C})$, $\mathcal{C}' = cont(\mathcal{C}, r)$, and M' such that $M[r\rangle M'$.*

1. *$\mathcal{C}'$ is a $\mathcal{K}$-course at M'.*
2. *Let $r' \in CRuns_N(M', \mathcal{C}')$, $\mathcal{C}'' = cont(\mathcal{C}', r')$, and M'' such that $M'[r'\rangle M''$. Then we have: $r.r' \in CRuns_N(M, \mathcal{C})$, $cont(\mathcal{C}, r.r') = \mathcal{C}''$, and $M[r.r'\rangle M''$.*

Proof. (1.) follows with Prop. 6.2.8. (2.) is immediate with the definitions. □

Example 6.6.2. Let M_0 be as indicated in Figure 6.5. $\gamma^{\mathcal{C}}_{K_1} := (t_2 t_1)^\omega$ is a K_1-course at M_0, $\gamma^{\mathcal{C}}_{K_2} := (t_5 t_6 t_4 t_9 t_{10})^\omega$ a K_2-course at M_0, and hence $\mathcal{C} := \{\gamma^{\mathcal{C}}_{K_1}, \gamma^{\mathcal{C}}_{K_2}\}$ is a $\{K_1, K_2\}$-course at M_0.

Set $r := t_2 t_5 t_1 t_2 t_6 t_1$, $\gamma^{\mathcal{C}'}_{K_1} := (t_2 t_1)^\omega$, $\gamma^{\mathcal{C}'}_{K_2} := (t_4 t_9 t_{10} t_5 t_6)^\omega$, and $\mathcal{C}' := \{\gamma^{\mathcal{C}'}_{K_1}, \gamma^{\mathcal{C}'}_{K_2}\}$. We have $r \in CRuns_N(M_0, \mathcal{C})$, and $cont(\mathcal{C}, r) = \mathcal{C}'$. Note that $\mathcal{C}'$ is a $\{K_1, K_2\}$-course at M, where M is as defined in Example 6.6.1, or equivalently given by $M_0[r\rangle M$.

Freezing Components. From Obs. 6.2.6 we know that whenever in a live fc system an SM-component K has arrived at a synch place, K never has to wait in vain: there is always a computation of the remaining system that makes K's synch partners ready for synchronization. However, the observation presupposes that the remaining system can be steered according to the requirements of K. If we grant other components a will of their own, and compute the system in conformity with a course $\mathcal{C}$, it can happen that K is made to wait indefinitely; it will be inactive as long as $\mathcal{C}$ is kept. Formally, a component can be *frozen* by a course in the following way:

Definition 6.6.3. Let $M \in Reach(\mathcal{N}_L)$, $\mathcal{K} \subseteq Cover$, and $\mathcal{C}$ a $\mathcal{K}$-course at M. We say a component $K \in Cover$ is *frozen by $\mathcal{C}$ at M* iff there are $K' \in \mathcal{K}$ and $p \in P_{K'}$ such that $K'(p) \prec\!\!\prec_M K$ & $\nexists t \in \gamma^{\mathcal{C}}_{K'}.\ p \in t^\bullet$. We use $frozen(M, \mathcal{C})$ to denote the set of components frozen by $\mathcal{C}$ at M.

Example 6.6.3. Let M_0, $M, \mathcal{C}, \mathcal{C}'$ be as in Example 6.6.2. We have $frozen(M_0, \mathcal{C}) = \{K_3\}$, and $frozen(M, \mathcal{C}') = \{K_2, K_3, K_4\}$.

Naturally, we have:

Proposition 6.6.5. *Let $M \in Reach(\mathcal{N}_L)$, $\mathcal{K} \subseteq Cover$, and $\mathcal{C}$ be a $\mathcal{K}$-course at M. If $K \in frozen(M, \mathcal{C})$ then for all $r \in CRuns_N(M, \mathcal{C})$ we have:*

1. *$K \notin Ks(r)$.*

2. $K \in \mathit{frozen}(M', \mathit{cont}(\mathcal{C}, r))$*, where* M' *is such that* $M[r\rangle M'$*.*

Proof. (1.) is immediate with Prop. 6.6.1(1b); (2.) follows with Prop. 6.6.1(2b). $\square$

Clause (2) implies that while computing in conformity with a course $\mathcal{C}$ the number of frozen components can only increase, never decrease. On the other hand, liveness makes sure that at least one component remains active: it is not possible to freeze all components; this is a consequence of the definition of frozen and Prop. 6.6.3. Moreover, if $\mathcal{C}$ is non-empty at least one of the components participating in $\mathcal{C}$ will be active:

Proposition 6.6.6. *Let* $M \in \mathit{Reach}(\mathcal{N}_L)$*,* $\mathcal{K} \subseteq \mathit{Cover}$ *with* $\mathcal{K} \neq \emptyset$*, and* $\mathcal{C}$ *be a* $\mathcal{K}$*-course at* M*. There is at least one component* $K \in \mathcal{K}$ *such that* $K \notin \mathit{frozen}(M, \mathcal{C})$*.*

Proof. Let M, $\mathcal{K}$, and $\mathcal{C}$ be given as above. If $\mathcal{K} \subseteq \mathit{frozen}(M, \mathcal{C})$ then for each $K \in \mathcal{K}$ there would be $K' \in \mathcal{K}$ such that $K' \prec\!\prec_M K$. But this is clearly not possible since $\mathcal{K}$ is finite and $\prec\!\prec_M$ is a strict order (Prop. 6.6.2). $\square$

The following property will be helpful later on; it can be derived from the definitions by case analysis.

Proposition 6.6.7. *Let* $M \in \mathit{Reach}(\mathcal{N}_L)$*,* $\mathcal{K} \subseteq \mathit{Cover}$*,* $\mathcal{C}$ *be a* $\mathcal{K}$*-course at* M*, and* K*,* $K' \in \mathit{Cover}$*. If* $K' \in \mathit{frozen}(M, \mathcal{C})$ *and* $M(K)$ *is a synch place such that* $\exists p' \in \mathit{SPartners}(M(K))$*.* $p' \in P_{K'}$ *then we also have* $K \in \mathit{frozen}(M, \mathcal{C})$*.*

Proof. See Appendix C.3. $\square$

The Computations of Components in the Context of a Course. As expressed in Theorem 6.2.2, in live fc systems the computations of an SM-component K are unconstrained in that any computation of K can also be achieved in the composite context. It is clear that this is not valid in the setting with courses: components can become frozen and Obs. 6.2.6 is not true. However, the following generalization of Theorem 6.2.2 does hold:

> Any computation of a component K that conforms to the present course can either also be performed in the context of the composite net, or it can be performed partially and K will be frozen at the resulting marking.

Theorem 6.6.1. *Let* $M \in \mathit{Reach}(\mathcal{N}_L)$*,* $\mathcal{K} \subseteq \mathit{Cover}$*,* $\mathcal{C}$ *be a* $\mathcal{K}$*-course at* M*, and* $K \in \mathit{Cover}$*. For all* $r_K \in \mathit{Runs}_K(M(K))$*, where* $r_K \in \mathit{FinPrefixes}(\gamma_K^{\mathcal{C}})$ *if* $K \in \mathcal{K}$*, there is* $r \in \mathit{CRuns}_N(M, \mathcal{C})$ *such that we have:*

1. $r \uparrow K = r_K$, *or*

2. $r \uparrow K \in \mathit{Prefixes}(r_K)$ & $K \in \mathit{frozen}(M', \mathit{cont}(\mathcal{C}, r))$, *where* M' *is such that* $M[r\rangle M'$.

The proof of Theorem 6.6.1 is lengthy: one has to prove a generalized version of it for a setting where not only infinite but also finite computations are admitted as courses of components. The generalized theorem can then be proved by double induction on the number of components not frozen by $\mathcal{C}$ at M, and the length of r_K. The detailed proof can be found in [Frö00a].

We have already seen that when computing $\mathcal{N}_L$ in conformity with a non-empty $\mathcal{K}$-course $\mathcal{C}$, there is at least one $K \in \mathcal{K}$ which is not frozen by $\mathcal{C}$ (Prop. 6.6.6). It is intuitive that there should be at least one $K \in \mathcal{K}$ which is always able to follow its course. With Theorem 6.6.1 this is now easy to prove.

Proposition 6.6.8. *Let* $M \in \mathit{Reach}(\mathcal{N}_L)$, $\mathcal{K} \subseteq \mathit{Cover}$ *with* $\mathcal{K} \neq \emptyset$, *and* $\mathcal{C}$ *a* $\mathcal{K}$*-course at* M. *There is at least one* $K \in \mathcal{K}$ *which can follow its course* $\gamma_K^{\mathcal{C}}$ *while* $\mathcal{C}$ *is kept; that is for all* $r_K \in \mathit{FinPrefixes}(\gamma_K^{\mathcal{C}})$ *there is* $r \in \mathit{CRuns}_N(M, \mathcal{C})$ *such that* $r \uparrow K = r_K$.

Proof. To the contrary assume there is no such K. This means for all $K \in \mathcal{K}$ there exists $r_K \in \mathit{FinPrefixes}(\gamma_K^{\mathcal{C}})$ such that there is no $r \in \mathit{CRuns}_N(M, \mathcal{C})$ with $r \uparrow K = r_K$. With the help of Theorem 6.6.1 one can then bring one component after another into a frozen state. But this is certainly a contradiction to Prop. 6.6.6. □

6.7 On Links and Wedges

In this section we will introduce the new topological entities *links* and *wedges*, and present important results about them. We will first attend to links in Section 6.7.1, then to wedges in Section 6.7.2. After that the section will culminate in a structural result concerning both of these topological entities in live SSMD fc systems, namely the promised WNL Theorem; it will be formulated and proved in Section 6.7.3.

6.7.1 Links

Links are topological entities of free choice nets, which — when interpreted w.r.t. a strict decomposition — exhibit that two components are interconnected or linked to each other in a certain way. We are interested in two forms of interconnection, and define two corresponding types of links, so-called *simple links* and *indirect links*. Both of these types play an important role when observing how components are matched against each other in a cp bisimulation, and how this can change in the course of the matching.

The section is organized as follows. In the next paragraph we introduce two types of paths, which we need for the definition of links. The definitions of simple links, indirect links, and links in general follow together with their attributes. We then show that in live SSMD fc systems, starting from either the initial or the final component we can freeze all of a link's components. This property will be essential for proving an important result about links, the so-called Link Theorem, which will follow in the last paragraph together with a corollary. The Link Corollary will make up half of the WNL Theorem.

WS-paths and FOS-paths. The two types of paths are called *WS-paths* and *FOS-paths*; they are defined as follows:

Definition 6.7.1 (WS-paths, FOS-paths). Let N be a free choice net.

A *WS-path* (path without synchronization) of N is a PP-path $\pi = p_1 t_1 \dots t_n p_{n+1}$ of N such that for all $i \in [1, n]$ t_i is a switch transition.

An *FOS-path* (path from the only synchronization) of N is a TP-path $\pi = t_1 p_1 \dots t_n p_n$ of N such that t_1 is a synchronization transition, and $p_1 \dots t_n p_n$ is a WS-path.

When interpreted w.r.t. a strict decomposition we can make the following statement about the components of a WS-path, and FOS-path respectively:

Proposition 6.7.1. *Let $(N, Cover)$ be an SSD fc net, and $\pi = x_1 \dots x_n$ a path in N.*

1. *If π is a WS-path then we have:*
$$Ks(\pi) = \{K\} \text{ for some distinct } K \in Cover.$$

2. *If π is a FOS-path then we have:*
$$Ks(x_2 \dots x_n) = \{K\} \;\&\; K \in Ks(x_1) \text{ for some distinct } K \in Cover.$$

Proof. (1.) and (2.) are straightforward: consider that *Cover* is strict, and employ Prop. 6.2.4(1) and 6.2.2(2a,b). □

We associate the respective component to a WS- or FOS-path:

Definition 6.7.2 ($DK(\pi)$). Let $(N, Cover)$ be an SSD fc net, and π either a WS- or FOS-path of N. We define the *designated component of* π, denoted by $DK(\pi)$, to be the component associated with π by Prop. 6.7.1(1), or (2) respectively.

Simple Links. We are now ready to introduce our first type of link. Simple links — when interpreted w.r.t. to a strict decomposition — give a route to traverse from one component to another, possibly by passing through other components via synch transitions. They are essentially PP-paths divided up into segments; the segments will naturally be FOS-paths, and a WS-path in the case of the initial segment. Formally, we define:

Definition 6.7.3 (simple links). Let N be a free choice net.

A *simple link of* N is a non-empty and finite family of segments $\lambda = \{S_{\lambda\text{-}i}\}_{i \in [1, n_\lambda]}$, $n_\lambda \in \mathbb{N}$, where

- $S_{\lambda\text{-}1}$ is a WS-path, and
- for $i \in [2, n_\lambda]$ $S_{\lambda\text{-}i}$ is an FOS-path

such that $last(S_{\lambda\text{-}i}) \in {}^\bullet\!first(S_{\lambda\text{-}i+1})$ for all $i \in [1, n_\lambda - 1]$.

We call n_λ the *number of segments of* λ, and $S_{\lambda\text{-}i}$ the *ith segment of* λ.

We equip simple links with the following attributes; note how they obtain their full meaning with the "decomposition attributes".

Definition 6.7.4 (attributes of simple links). Let N be a free choice net, and $\lambda = \{S_{\lambda\text{-}i}\}_{i \in [1, n_\lambda]}$ a simple link in N.

We define:

- the *initial place of* λ as $p^{in}_{\lambda} := \mathit{first}(S_{\lambda\text{-}1})$,
- the *final place of* λ as $p^{fi}_{\lambda} := \mathit{last}(S_{\lambda\text{-}n_\lambda})$, and
- for $i \in [1, n_\lambda - 1]$ the *exiting synchronization place of* $S_{\lambda\text{-}i}$ as $p^{ex}_{\lambda\text{-}i} := \mathit{last}(S_{\lambda\text{-}i})$.

W.r.t. a strict cover *Cover* of N we further define:

- for $i \in [1, n_\lambda]$ the *ith component of* λ as $K_{\lambda\text{-}i} := DK(S_{\lambda\text{-}i})$,
- the *components of* λ as $\mathcal{K}_\lambda := \{K_{\lambda\text{-}i}\}_{i\in[1,n_\lambda]}$,
- the *initial component of* λ as $K^{in}_{\lambda} := K(p^{in}_{\lambda}) = K_{\lambda\text{-}1}$,
- the *final component of* λ as $K^{fi}_{\lambda} := K(p^{fi}_{\lambda}) = K_{\lambda\text{-}n_\lambda}$, and
- for $i \in [2, n_\lambda]$ the *entering synchronization place of* $S_{\lambda\text{-}i}$ as $p^{en}_{\lambda\text{-}i}$ given by $p^{en}_{\lambda\text{-}i} \in {}^\bullet\mathit{first}(S_{\lambda\text{-}i})$ & $K(p^{en}_{\lambda\text{-}i}) = K_{\lambda\text{-}i}$ (this uniquely defines a place by Prop. 6.2.2(1a)).

We say λ is a simple link from K^{in}_{λ} to K^{fi}_{λ}.

Example 6.7.1. Figure 6.8 shows a simple link together with its attributes.

Indirect Links. In the context of a strict decomposition, indirect links show how two components are indirectly connected via a middle component: an indirect link is made up of two simple links that overlap in their initial place.

Definition 6.7.5 (indirect links). Let N be a free choice net.

An *indirect link of* N is a pair $\lambda = (\lambda_l, \lambda_r)$, where λ_l and λ_r are simple links of N such that $p^{in}_{\lambda_l} = p^{in}_{\lambda_r}$. We call λ_l the *left link of* λ, and λ_r the *right link of* λ.

Definition 6.7.6 (attributes of indirect links). Let (N, Cover) be an SSD fc net, and $\lambda = (\lambda_l, \lambda_r)$ an indirect link of N. We define:

- the *initial component of* λ as $K^{in}_{\lambda} := K^{fi}_{\lambda_l}$,
- the *final component of* λ as $K^{fi}_{\lambda} := K^{fi}_{\lambda_r}$,
- the *middle component of* λ as $K^{mi}_{\lambda} := K^{in}_{\lambda_l} = K^{in}_{\lambda_r}$, and
- the *components of* λ as $\mathcal{K}_\lambda = Ks(\lambda_l) \cup Ks(\lambda_r)$.

We say λ is an indirect link from K^{in}_{λ} to K^{fi}_{λ}.

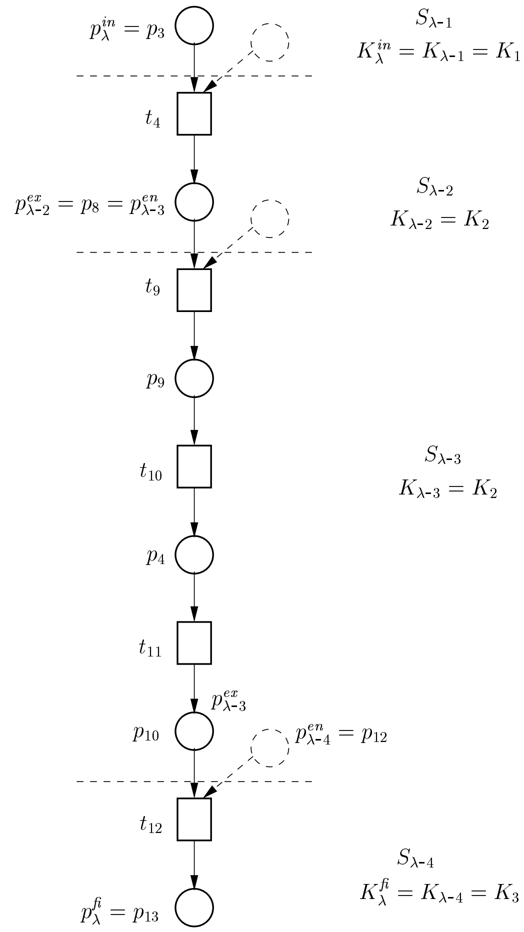


Figure 6.8: A simple link λ of the system in Figure 6.5; λ leads from K_1 to K_3, and we have $\mathcal{K}_\lambda = \{K_1, K_2, K_3\}$

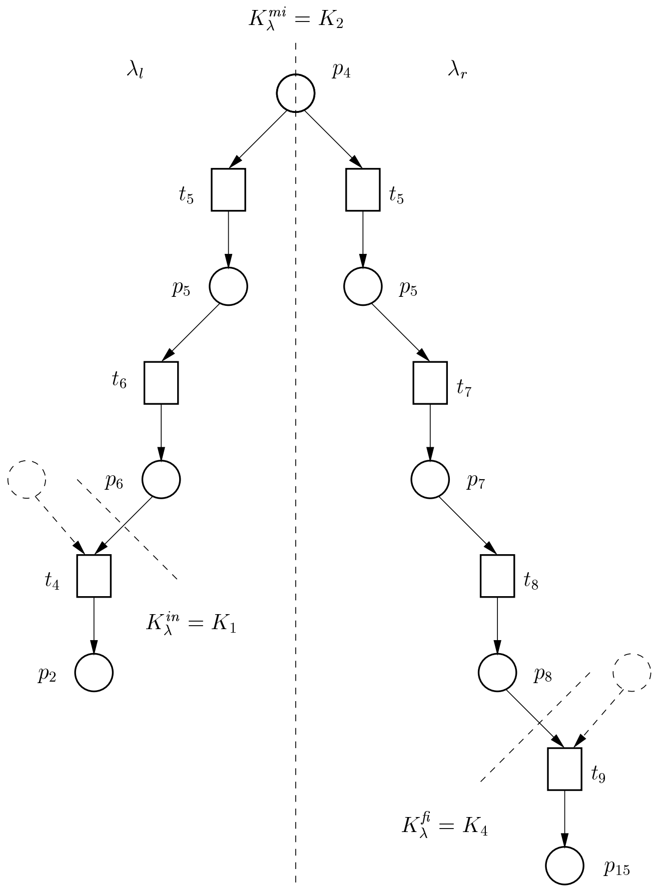


Figure 6.9: An indirect link λ of the system in Figure 6.5; λ leads from K_1 to K_4, and we have $\mathcal{K}_\lambda = \{K_1, K_2, K_4\}$

Note that the initial component of an indirect link is the final component of its left link. This is so because the left link has to be understood as a backwards connection to the middle component.

Example 6.7.2. Figure 6.9 shows an indirect link together with its attributes.

Links. Links are then simply defined by:

Definition 6.7.7 (links). Let N be a free choice net. A *link of* N is either a simple or an indirect link of N.

K_λ^{in}, K_λ^{fi}, and $\mathcal{K}_\lambda$ are attributes for both types of links, and thus we shall assume them for links in a generic way.

Freezing the Components of a Link. In Section 6.6 we have seen that when a live SSD fc system is computed in conformity with a course it can happen that some of the components become frozen. We will now prove the following interesting property about links in such systems:

> If we can freeze the initial component of a link λ by some $\mathcal{K}$-course $\mathcal{C}$ then, provided that $\mathcal{K}$ and the components of λ do not overlap, we can steer the system into a state where all the components of λ are frozen. This also works in the other direction when the final component of λ is frozen.

Formally, this amounts to:

Lemma 6.7.1 (freeze the components of a link). *Let* $(\mathcal{N}, \mathit{Cover})$ *be a live SSMD fc system,* $M \in \mathit{Reach}(\mathcal{N})$*,* $\mathcal{K} \subseteq \mathit{Cover}$*, and* $\mathcal{C}$ *a* $\mathcal{K}$*-course at* M*. For all links* λ *of* N *with* $\mathcal{K}_\lambda \cap \mathcal{K} = \emptyset$ *we have:*

1. *If* $K_\lambda^{in} \in \mathit{frozen}(M, \mathcal{C})$ *then there is* $r \in \mathit{CRuns}_N(M, \mathcal{C})$ *such that we have* $\mathcal{K}_\lambda \subseteq \mathit{frozen}(M', \mathcal{C}')$ *for* $\mathcal{C}' = \mathit{cont}(\mathcal{C}, r)$*, and* M' *given by* $M[r\rangle M'$*.*

2. *If* $K_\lambda^{fi} \in \mathit{frozen}(M, \mathcal{C})$ *then there is* $r \in \mathit{CRuns}_N(M, \mathcal{C})$ *such that we have* $\mathcal{K}_\lambda \subseteq \mathit{frozen}(M', \mathcal{C}')$ *for* $\mathcal{C}' = \mathit{cont}(\mathcal{C}, r)$*, and* M' *given by* $M[r\rangle M'$*.*

Proof. Let $\mathcal{N}$, Cover, M, $\mathcal{K}$, $\mathcal{C}$, and λ be given as specified in the lemma. We shall prove the property separately for the two types of links. In the next two paragraphs we will establish (1.) and (2.) assuming λ to be a simple link.

(S1.) We suppose $K^{in}_\lambda = K_{\lambda\text{-}1} \in \mathit{frozen}(M, \mathcal{C})$, and demonstrate by induction on n_λ that there is $r \in \mathit{CRuns}_N(M, \mathcal{C})$ such that $(*)$ $\forall i \in [1, n_\lambda]$. $K_{\lambda\text{-}i} \in \mathit{frozen}(M', \mathcal{C}')$, where $\mathcal{C}' = \mathit{cont}(\mathcal{C}, r)$, M' is given by $M[r\rangle M'$. This will certainly prove the first part of the lemma for simple links.

Case $n_\lambda = 1$: we take r to be the empty sequence; since $K_{\lambda\text{-}1}$ is the only component of λ, $(*)$ is satisfied by assumption.

Case $n_\lambda > 1$: certainly $\lambda' = \{S_{\lambda\text{-}i}\}_{i \in [1, n_\lambda - 1]}$ is a simple link such that the induction hypothesis applies; then there is $r \in \mathit{CRuns}_N(M, \mathcal{C})$ such that we obtain $\forall i \in [1, n_\lambda - 1]$. $K_{\lambda\text{-}i} \in \mathit{frozen}(M', \mathcal{C}')$, where $\mathcal{C}' = \mathit{cont}(\mathcal{C}, r)$, M' is given by $M[r\rangle M'$. If $K_{\lambda\text{-}n_\lambda} \in \mathit{frozen}(M', \mathcal{C}')$ we can clearly return r as the required run; otherwise we proceed as follows to bring $K_{\lambda\text{-}n_\lambda}$ into a frozen state.

First of all, pick a $K_{\lambda\text{-}n_\lambda}$-computation $r'_{K_{\lambda\text{-}n_\lambda}}$ which brings $K_{\lambda\text{-}n_\lambda}$ from $M'(K_{\lambda\text{-}n_\lambda})$ to $p^{en}_{\lambda\text{-}n_\lambda}$; this must be possible by Prop. 6.2.17(2). Recall that $K_{\lambda\text{-}n_\lambda} \notin \mathcal{K}$ by assumption. Then, by Theorem 6.6.1 there is $r' \in \mathit{CRuns}_N(M', \mathcal{C}')$ such that we have (i) $r' \uparrow K_{\lambda\text{-}n_\lambda} = r'_{K_{\lambda\text{-}n_\lambda}}$, or (ii) $r' \uparrow K_{\lambda\text{-}n_\lambda} \in \mathit{Prefixes}(r'_{K_{\lambda\text{-}n_\lambda}})$ & $K_{\lambda\text{-}n_\lambda} \in \mathit{frozen}(M'', \mathcal{C}'')$, where $\mathcal{C}'' = \mathit{cont}(\mathcal{C}', r')$, M'' such that $M'[r'\rangle M''$.

By Prop. 6.6.5(2) frozen components stay frozen, and so $K_{\lambda\text{-}i} \in \mathit{frozen}(M'', \mathcal{C}'')$ for all $i \in [1, n_\lambda - 1]$. But we also have $K_{\lambda\text{-}n_\lambda} \in \mathit{frozen}(M'', \mathcal{C}'')$, just as we aimed for. This is trivial in the case when (ii) holds. In the case of (i) consider that we clearly have $M''(K_{\lambda\text{-}n_\lambda}) = p^{en}_{\lambda\text{-}n_\lambda}$ (Prop. 6.2.8); but then $K_{\lambda\text{-}n_\lambda} \in \mathit{frozen}(M'', \mathcal{C}'')$ follows from $K_{\lambda\text{-}(n_\lambda - 1)} \in \mathit{frozen}(M'', \mathcal{C}'')$ with Prop. 6.6.7.

Altogether $r.r'$ clearly satisfies all of our requirements (with Prop. 6.6.4(2)).

(S2.) The proof of the second part is similar to (S1.); the only difference is that we freeze the components of λ in reverse order. Formally, we assume $K^{fi}_\lambda = K_{\lambda\text{-}n_\lambda} \in \mathit{frozen}(M, \mathcal{C})$, and show by induction on n_λ that there is $r \in \mathit{CRuns}_N(M, \mathcal{C})$ such that we have $(*)$ $\forall i \in [1, n_\lambda]$. $K_{\lambda\text{-}i} \in \mathit{frozen}(M', \mathcal{C}')$, where $\mathcal{C}' = \mathit{cont}(\mathcal{C}, r)$, M' is given by $M[r\rangle M'$.

Case $n_\lambda = 1$: again we take r to be the empty sequence; as in (S1.) $(*)$ holds by assumption.

Case $n_\lambda > 1$: this time we consider $\{S_{\lambda\text{-}i}\}_{i \in [2, n_\lambda]}$. Clearly, we can transform this family of segments into a simple link λ' such that the induction hypothesis applies; then we can assume $r \in \mathit{CRuns}_N(M, \mathcal{C})$ such that $\forall i \in [2, n_\lambda]$. $K_{\lambda\text{-}i} \in \mathit{frozen}(M', \mathcal{C}')$, where $\mathcal{C}' = \mathit{cont}(\mathcal{C}, r)$, M' is given by $M[r\rangle M'$.

If $K_{\lambda\text{-}1} \in \mathit{frozen}(M', \mathcal{C}')$ we are already done; otherwise we pick a $K_{\lambda\text{-}1}$-computation from $M'(K_{\lambda\text{-}1})$ to $p^{ex}_{\lambda\text{-}1}$, and proceed analogously to (S1.). By the corresponding argument we obtain $r' \in \mathit{CRuns}_N(M', \mathcal{C}')$ such that $\forall i \in [1, n_\lambda]$. $K_{\lambda\text{-}i} \in$

frozen$(M'', \mathcal{C}'')$, where $\mathcal{C}'' = cont(\mathcal{C}', r')$, and M'' is given by $M'[r'\rangle M''$. But then clearly, $r.r'$ provides a sequence as required.

Now that we know the lemma holds for simple links, we can also prove it for indirect links. So let us assume λ to be an indirect link for the next two paragraphs.

(I1.) We presuppose $K_\lambda^{in} = K_{\lambda_l}^{fi} \in frozen(M, \mathcal{C})$, and need to prove: there is $r \in CRuns_N(M, \mathcal{C})$ satisfying $\mathcal{K}_\lambda \subseteq frozen(M', \mathcal{C}')$, where $\mathcal{C}' = cont(\mathcal{C}, r)$, M' such that $M[r\rangle M'$. With the lemma being established for simple links this is not difficult; we first freeze the components of λ_l, then the ones of λ_r. Formally:

By (S2.) there is $r \in CRuns_N(M)$ such that we have $\mathcal{K}_{\lambda_l} \subseteq frozen(M', \mathcal{C}')$, where $\mathcal{C}' = cont(\mathcal{C}, r)$, M' such that $M[r\rangle M'$. Then, by $K_{\lambda_r}^{in} = K_{\lambda_l}^{in} \in frozen(M', \mathcal{C}')$ and (S1.) there is $r' \in CRuns_N(M', \mathcal{C}')$ such that we have $\mathcal{K}_{\lambda_r} \subseteq frozen(M'', \mathcal{C}'')$, where $\mathcal{C}'' = cont(\mathcal{C}', r')$, M'' such that $M'[r'\rangle M''$. By Prop. 6.6.5(2) the components of λ_l are also frozen by $\mathcal{C}''$ at M'', and so we have $\mathcal{K}_\lambda \subseteq frozen(M'', \mathcal{C}'')$. But then the sequence $r.r'$ clearly fulfils all of our requirements.

(I2.) Follows by the symmetric argument; this time we first freeze the components of λ_r then the ones of λ_l. $\square$

The Link Theorem. The "freeze the components of a link" lemma enables us to prove an important theorem about links in live SSMD fc systems. Informally, it says:

> Let K_F be a component, and λ a link such that K_F is not a component of λ. If a K_F-course γ_F freezes the initial or final component of λ then K_F does *not* synchronize infinitely often with any of the components of λ on γ_F.

Theorem 6.7.1 (Link Theorem). *Let $(\mathcal{N}, Cover)$ be a live SSMD fc system, and $M \in Reach(\mathcal{N})$. Further, let λ be a link of N, $K_F \in Cover$ with $K_F \notin \mathcal{K}_\lambda$, and γ_F a K_F-course at M.*

1. *If $K_\lambda^{in} \in frozen(M, \{\gamma_F\})$ then we have $InfSPartners(K_F, \gamma_F)$*[6] *$\cap \mathcal{K}_\lambda = \emptyset$.*
2. *If $K_\lambda^{fi} \in frozen(M, \{\gamma_F\})$ then we have $InfSPartners(K_F, \gamma_F) \cap \mathcal{K}_\lambda = \emptyset$.*

[6]Recall Def. 6.2.11.

Proof. Let $\mathcal{N}$, *Cover*, M, λ, K_F, and γ_F as specified above, and set $\mathcal{C}_F = \{\gamma_F\}$.

(1.) To prove the first clause let $K^{in}_\lambda \in \mathit{frozen}(M, \mathcal{C}_F)$, and to the contrary assume $K_s \in \mathit{Cover}$ such that (A) $K_s \in \mathit{InfSPartners}(K_F, \gamma_F)$, and (B) $K_s \in \mathcal{K}_\lambda$. By Lemma 6.7.1(1) and (B) there is $r \in \mathit{CRuns}_N(M, \mathcal{C}_F)$ such that $K_s \in \mathit{frozen}(M', \mathcal{C}'_F)$, where $\mathcal{C}'_F = \mathit{cont}(\mathcal{C}_F, r)$, M' such that $M[r\rangle M'$. Let γ'_F be given by $\mathcal{C}'_F = \{\gamma'_F\}$. γ'_F is a suffix of γ_F, and thus with (A) we obtain r_F, t_s such that $r_F.t_s \in \mathit{FinPrefixes}(\gamma'_F)$, and t_s is a synch transition with $K_s \in \mathit{Ks}(t_s)$. Further, since K_F is the only component of $\mathcal{C}'_F$ by Prop. 6.6.8 there must be $r' \in \mathit{CRuns}_N(M', \mathcal{C}'_F)$ such that $r' \uparrow K_F = r_F.t_s$, and hence $K_s \in \mathit{Ks}(r')$. But this means we have reached a contradiction: $K_s \in \mathit{frozen}(M', \mathcal{C}'_F)$, which by Prop. 6.6.5(1) implies K_s must not be involved in r'.

(2.) This is similar: simply use the second part of Lemma 6.7.1 instead of the first. ☐

The Link Theorem will make up half of the WNL Theorem, but it will be more convenient to employ it in the form of a corollary. For the formulation of this Link Corollary we first define:

Definition 6.7.8. Let $(\mathcal{N}, \mathit{Cover})$ be a live SSMD fc system, and K_c, K_1, $K_2 \in \mathit{Cover}$.

Let $M \in \mathit{Reach}(\mathcal{N})$. We say a K_c-course γ_c at M is (K_1, K_2)-*critical* iff

1. $K_1 \in \mathit{frozen}(M, \{\gamma_c\})$, and
2. $K_2 \in \mathit{InfSPartners}(K_c, \gamma_c)$.

We say K_c is *critical w.r.t.* K_1 *and* K_2 iff there exists a K_c-course γ_c at some $M \in \mathit{Reach}(\mathcal{N})$ such that γ_c is (K_1, K_2)-critical or (K_2, K_1)-critical.

We can now understand the statement of the Link Theorem as follows:

> For three components K_c, K_1, and K_2 in a live SSMD fc system we have: if K_c is critical w.r.t. K_1 and K_2 then K_1 and K_2 can only be linked via K_c.

Corollary 6.7.1 (Link Corollary). *Let $(\mathcal{N}, \mathit{Cover})$ be a live SSMD fc system, and K_c, K_1, $K_2 \in \mathit{Cover}$. If K_c is critical w.r.t. K_1 and K_2 then the following holds:*

1. *For any link λ from K_1 to K_2 we have $K_c \in \mathcal{K}_\lambda$.*
2. *For any link λ from K_2 to K_1 we have $K_c \in \mathcal{K}_\lambda$.*

Proof. Let $\mathcal{N}$, *Cover*, K_c, K_1, and K_2 be given as above. If K_c is critical w.r.t. K_1 and K_2 then by definition there exists a K_c-course γ_c at some $M \in \mathit{Reach}(\mathcal{N})$ such that γ_c is (a) (K_1, K_2)-critical, or (b) (K_2, K_1)-critical. In the case of (a), (1.) follows from the first part of Theorem 6.7.1, and (2.) from its second part. Conversely, if (b) holds, (1.) follows from the second part of Theorem 6.7.1, and (2.) from its first part. □

6.7.2 Wedges

We now turn to our second topological entity, *wedges*. A wedge is made up of two *TFS-paths*, which overlap in their first element. TFS-paths are special kinds of WS-paths: apart from not containing any synch transitions they start with a switch place and end in a synch place. If the two synch places of a wedge are different we consider it to be *proper*. Wedges are motivated by our interest in the matching of *swfsi's*: if a swfsi occurrence of a live SSMD fc system has two alternative matches in a cp bisimulation with another system of the same class, say $\mathcal{N}_2$, then there will be a proper wedge in $\mathcal{N}_2$.

The remainder of the section is organized as follows: in the next two paragraphs, we define TFS-paths and wedges. Then follow two paragraphs with preliminary observations. With their help we then obtain our key result in the final paragraph: the so-called Wedge Theorem complements the Link Corollary, and makes up the other half of the WNL Theorem.

TFS-paths. TFS-paths are defined as follows:

Definition 6.7.9 (TFS-paths). Let N be a free choice net.

A *TFS-path* (path from switch place To First Synchronization place) of N is a WS-path π of N such that $\mathit{first}(\pi)$ is a switch place, and $\mathit{last}(\pi)$ is a synch place.

Since TFS-paths are special kinds of WS-paths they naturally inherit the attributes and properties of the latter; in particular, we will employ $DK(\pi)$, and the fact that WS-paths are dynamic in the following sense:

Proposition 6.7.2 (WS-paths are dynamic). *Let $\mathcal{N}$ be a free choice system, and $M \in \mathit{Reach}(\mathcal{N})$. If π is a WS-path of N with $\mathit{first}(\pi) \in M$ then we have: $M[ts(\pi)\rangle$ & $\mathit{last}(\pi) \in M'$, where M' is such that $M[ts(\pi)\rangle M'$.*

Proof. Easy by induction on the length of π. Consider that for any WS-path $\pi'\, t\, p$ we have: t is of type switch, ${}^\bullet t = \{\mathit{last}(\pi')\}$, and $p \in t^\bullet$. □

Wedges. Wedges and proper wedges are then given by:

Definition 6.7.10 (wedges, proper wedges). Let N be a free choice net.

A *wedge of* N is a pair $W = (\pi_{W\text{-}l}, \pi_{W\text{-}r})$, where $\pi_{W\text{-}l}$ and $\pi_{W\text{-}r}$ are TFS-paths of N such that $\mathit{first}(\pi_{W\text{-}l}) = \mathit{first}(\pi_{W\text{-}r})$. We call $\pi_{W\text{-}l}$ the left TFS-path of W, and $\pi_{W\text{-}r}$ the right TFS-path of W.

We say a wedge W is *proper* iff $\mathit{last}(\pi_{W\text{-}l}) \neq \mathit{last}(\pi_{W\text{-}r})$.

We equip wedges with the following attributes:

Definition 6.7.11 (attributes of wedges). Let N be a free choice net, and W a wedge of N.

We define:

- the *choice place of* W as $p^{ch}_W := \mathit{first}(\pi_{W\text{-}l}) = \mathit{first}(\pi_{W\text{-}r})$,
- the *left synchronization interface of* W as $p^{si}_{W\text{-}l} := \mathit{last}(\pi_l)$,
- the *right synchronization interface of* W as $p^{si}_{W\text{-}r} := \mathit{last}(\pi_r)$,
- the *left synchronization partners of* W as $P^{sp}_{W\text{-}l} := \mathit{SPartners}(p^{si}_{W\text{-}l})$, and
- the *right synchronization partners of* W as $P^{sp}_{W\text{-}r} := \mathit{SPartners}(p^{si}_{W\text{-}r})$.

W.r.t. a strict cover *Cover* of N we further define:

- the *component of* W as $K_W := DK(\pi_{W\text{-}l}) = DK(\pi_{W\text{-}r})$,
- the *left synchronization components of* W as $\mathcal{K}^{sp}_{W\text{-}l} := Ks(P^{sp}_{W\text{-}l})$, and
- the *right synchronization components of* W as $\mathcal{K}^{sp}_{W\text{-}r} := Ks(P^{sp}_{W\text{-}r})$.

Example 6.7.3. Figure 6.10 shows a proper wedge together with its attributes.

Wedge has Markings. We now prove that for any given wedge in a live SSMD fc system we obtain the following reachable markings:

Lemma 6.7.2 (wedge has markings). *Let $\mathcal{N}$ be a live SSMD fc system, and W a wedge of N.*

1. (a) *There is $M \in \mathit{Reach}(\mathcal{N})$ such that $p^{ch}_W \in M$ & $P^{sp}_{W\text{-}l} \subseteq M$.*

   (b) *There is $M \in \mathit{Reach}(\mathcal{N})$ such that $p^{ch}_W \in M$ & $P^{sp}_{W\text{-}r} \subseteq M$.*

2. (a) *There is $M \in \mathit{Reach}(\mathcal{N})$ such that $p^{si}_{W\text{-}l} \in M$ & $P^{sp}_{W\text{-}r} \subseteq M$.*

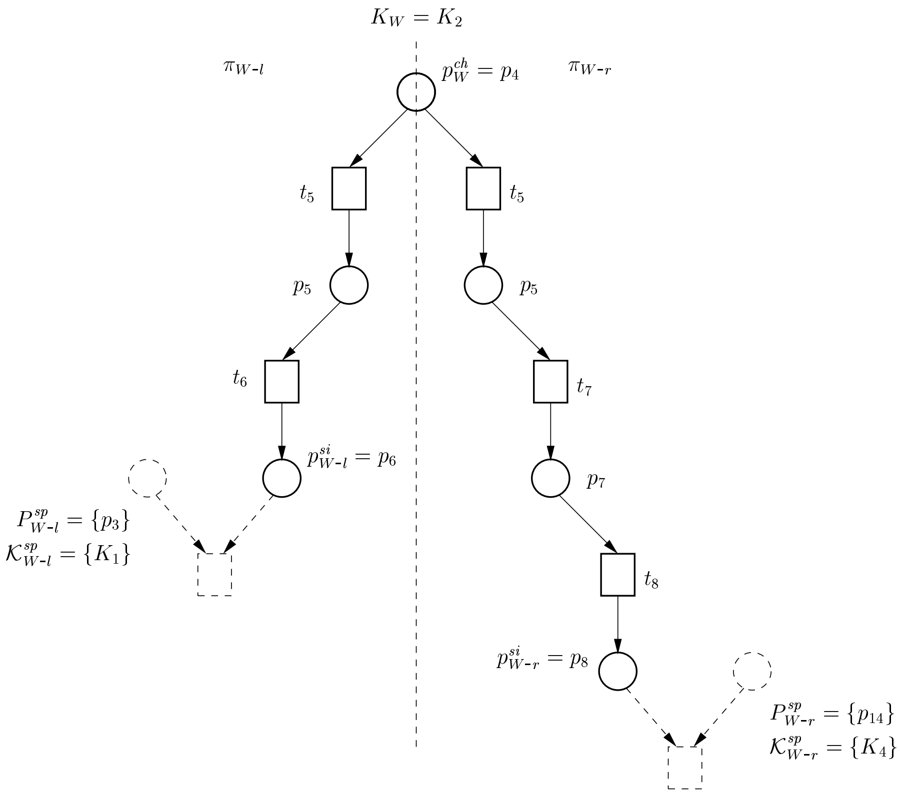


Figure 6.10: A proper wedge W of the system in Figure 6.5

(b) There is $M \in Reach(\mathcal{N})$ such that $p^{si}_{W\text{-}r} \in M$ & $P^{sp}_{W\text{-}l} \subseteq M$.

Proof. Let $\mathcal{N}$ and W be given as specified above; further assume a strict cover *Cover* for $\mathcal{N}$. Note that we clearly have $Ks(ts(\pi_{W\text{-}l})) = \{K_W\}$. We will employ this fact several times, and shall refer to it by (F).

(1a) By place-liveness (Prop. 6.2.14) we can assume $M \in Reach(\mathcal{N})$ such that $p^{ch}_W \in M$. $\pi_{W\text{-}l}$ is dynamic (Prop. 6.7.2), and so there must be M' with $M[ts(\pi_{W\text{-}l})\rangle M'$ and $p^{si}_{W\text{-}l} \in M'$. Then by Prop. 6.2.13 there is r, M'' such that $M'[r\rangle M''$, $P^{sp}_{W\text{-}l} \subseteq M''$, and $K_W(= K(p^{si}_{W\text{-}l})) \notin Ks(r)$. Further, by (F) and Prop. 6.2.3(2) we have $ts(\pi_{W\text{-}l})\ I_N\ r$, and so we can reshuffle these computations (Prop. 2.1.2): we obtain M''' such that $M[r\rangle M'''[ts(\pi_{W\text{-}l})\rangle M''$.

We claim that M''' is a marking as required. To see $P^{sp}_{W\text{-}l} \subseteq M'''$ consider the following: let $p \in P^{sp}_{W\text{-}l}$. We certainly have $K(p) \neq K_W$ (Prop. 6.2.2(1a)); then $p \in M'''$ follows immediately with $p \in M''$ and (F) by Prop. 6.2.9(3). The requirement $p^{ch}_W \in M'''$ is similarly obtained from $p^{ch}_W \in M$, $K(p^{ch}_W) = K_W$, and $K_W \notin Ks(r)$ via Prop. 6.2.9(3). Hence (1a) is proved.

(1b) follows from the symmetric argument of (1a).

(2a) By (1b) we can assume $M \in Reach(\mathcal{N})$ such that $p^{ch}_W \in M$ & $P^{sp}_{W\text{-}r} \subseteq$

M. $\pi_{W\text{-}l}$ is dynamic (Prop. 6.7.2), and so there is M' with $M[ts(\pi_{W\text{-}l})\rangle M'$ and $p^{si}_{W\text{-}l} \in M'$. We still have $P^{sp}_{W\text{-}r} \subseteq M'$; this follows from (F) and $K_W \notin \mathcal{K}^{sp}_{W\text{-}r}$ (by Prop. 6.2.2(1a)) with Prop 6.2.9(2). Altogether, M' is a marking as required.

(2b) is symmetric to (2a). □

The dp-l and dp-r Property. Given a wedge W of a SSD fc net we can consider whether there is a K_W-path which leads from $p^{si}_{W\text{-}l}$ to p^{ch}_{W} without passing through $p^{si}_{W\text{-}r}$; and naturally we can also consider the symmetric property. Accordingly, we define two properties, called **dp-l** (direct path left) and **dp-r** (direct path right):

Definition 6.7.12 (dp-l, dp-r). Let (N, Cover) be an SSD fc net. A wedge W of N can satisfy the following two properties:

1. **dp-l**: There is a K_W-path π from $p^{si}_{W\text{-}l}$ to p^{ch}_{W} such that $p^{si}_{W\text{-}r} \notin \pi$.

2. **dp-r**: There is a K_W-path π from $p^{si}_{W\text{-}r}$ to p^{ch}_{W} such that $p^{si}_{W\text{-}l} \notin \pi$.

We now prove a fact that will be crucial in the next paragraph: each proper wedge of a well-formed SSD fc net satisfies at least one of our two properties.

Lemma 6.7.3 (proper wedges satisfy dp-l$\vee$dp-r). *Let (N, Cover) be a well-formed SSD fc net, and W a proper wedge of N. We have:*

$$W \models \textbf{dp-l} \vee \textbf{dp-r}.$$

Proof. Let N, Cover, and W be given as above.

To the contrary suppose W satisfies neither **dp-l** nor **dp-r**. It is easy to see that under this assumption there cannot be a K_W-path from $p^{si}_{W\text{-}l}$ to p^{ch}_{W}. By $W \not\models$ **dp-l** any candidate path π will lead us to $p^{si}_{W\text{-}r}$ before reaching p^{ch}_{W}. By $W \not\models$ **dp-r** π will be forced from $p^{si}_{W\text{-}r}$ to $p^{si}_{W\text{-}l}$, again without passing through p^{ch}_{W}. Then, since $p^{si}_{W\text{-}l}$ and $p^{si}_{W\text{-}r}$ are distinct, any candidate path will circle between $p^{si}_{W\text{-}l}$ and $p^{si}_{W\text{-}r}$ without ever reaching p^{ch}_{W}. But this means we have reached a contradiction: by definition of S-components K_W is strongly connected, and so certainly there must be a K_W-path from $p^{si}_{W\text{-}l}$ to p^{ch}_{W}.

□

The Wedge Theorem. With the help of our previous insights we now prove a complement to the Link Corollary; the so-called Wedge Theorem says:

> For any proper wedge W in a live SSMD fc system we have: K_W is critical w.r.t. any pair of opposite synchronization components of W.

By employing Lemma 6.7.2 we first show that the **dp-l** and **dp-r** properties give rise to critical K_W-courses in the following way:

Lemma 6.7.4. *Let* $(\mathcal{N}, \mathit{Cover})$ *be a live SSMD fc system,* W *a proper wedge of* N*,* $K_l \in \mathcal{K}^{sp}_{W\text{-}l}$*, and* $K_r \in \mathcal{K}^{sp}_{W\text{-}r}$*.*

1. *If* $W \models$ **dp-l** *then there exists a* (K_r, K_l)*-critical* K_W*-course at some* $M \in \mathit{Reach}(\mathcal{N})$*.*

2. *If* $W \models$ **dp-r** *then there exists a* (K_l, K_r)*-critical* K_W*-course at some* $M \in \mathit{Reach}(\mathcal{N})$*.*

Proof. Let $\mathcal{N}$, Cover, W, K_l, and K_r be given as above. Since W is proper we have (P) $p^{si}_{W\text{-}l} \neq p^{si}_{W\text{-}r}$.

(1.) Assume $W \models$ **dp-l**, and with it a K_W-path π_{dpl} from $p^{si}_{W\text{-}l}$ to p^{ch}_W such that (dpl) $p^{si}_{W\text{-}r} \notin \pi_{dpl}$. We need to exhibit $M \in \mathit{Reach}(\mathcal{N})$, and a K_W-course γ_W at M such that (1) $K_r \in \mathit{frozen}(M, \{\gamma_W\})$, and (2) $K_l \in \mathit{InfSPartners}(K_W, \gamma_W)$.

For M we take the marking given by Lemma 6.7.2(2a); then M is such that (a) $p^{si}_{W\text{-}l} \in M$, and (b) $P^{sp}_{W\text{-}r} \subseteq M$.

We obtain γ_W from π_{dpl} and $\pi_{W\text{-}l}$ in the following way: assuming $\pi_{dpl} = p_0 t_1 \dots t_n p_{n+1}$, and $\pi_{W\text{-}l} = p'_0 t'_1 \dots t'_m p'_{m+1}$, we set $\pi_W = p_0 t_1 \dots t_n p'_0 t'_1 \dots t'_m$. Since $p_{n+1} = p^{ch}_W = p'_0$, and $p_0 = p^{si}_{W\text{-}l} = p_{m+1}$, π_W is obviously a circuit. Further, π_{dpl} and $\pi_{W\text{-}l}$ are both K_W-paths, and hence π_W is a K_W-circuit. Then, π_W induces an infinite K_W-path π^ω_W in the obvious way. We set γ_W to be $ts(\pi^\omega_W)$. γ_W is certainly a K_W-course at M: by Prop. 6.2.17(1) γ_W is an infinite K_W-computation at $\{p^{si}_{W\text{-}l}\}$, and from (a) and $K(p^{si}_{W\text{-}l}) = K_W$, we obtain $M(K_W) = \{p^{si}_{W\text{-}l}\}$.

We can make two observations about γ_W: (i) $\nexists t \in \gamma_W.\ p^{si}_{W\text{-}r} \in t^\bullet$, and (ii) let t_{sl} be given by ${p^{si}_{W\text{-}l}}^\bullet = \{t_{sl}\}$; t_{sl} occurs infinitely often on γ_W.

To see that (i) holds, note that any K_W-circuit π that contains t with $p^{si}_{W\text{-}r} \in t^\bullet$ must also contain $p^{si}_{W\text{-}r}$: by Prop. 6.2.2(1b) and $K(p^{si}_{W\text{-}r}) = K_W$, $p^{si}_{W\text{-}r}$ is the only possible successor of t on π, and if t is the last element of π then the first element must be $p^{si}_{W\text{-}r}$. On the other hand, we have $p^{si}_{W\text{-}r} \notin \pi_W$: $p^{si}_{W\text{-}r} \notin \pi_{dpl}$, and $p^{si}_{W\text{-}r} \notin \pi_{W\text{-}l}$; the first is given by (dpl), and the latter follows from (P) and because $p^{si}_{W\text{-}l}$ is the only synch place of $\pi_{W\text{-}l}$. Then (i) is immediate: there can be no t with $p^{si}_{W\text{-}r} \in t^\bullet$ on π_W, nor on the infinite path induced by it.

To verify (ii), note that on any path, t_{sl} is the only possible successor of $p^{si}_{W\text{-}l}$, and thus if a circuit contains $p^{si}_{W\text{-}l}$ it will also contain t_{sl}. Further, it is clear that all elements of a circuit π occur infinitely often on the infinite path induced by π. Since we have $p^{si}_{W\text{-}l} \in \pi_W$, altogether this certainly implies (ii).

With these observations it is now easy to prove that (1) and (2) are indeed satisfied. To see that (1) holds, note that from (a), (P), and $K(p^{si}_{W\text{-}l}) = K_W = K(p^{si}_{W\text{-}r})$ we can deduce $p^{si}_{W\text{-}r} \notin M$ (Prop. 6.2.6), and further with (b) that $K_W(p^{si}_{W\text{-}r}) \prec_M K_r$. Then with (i) it is clear that $K_r \in \mathit{frozen}(M, \{\gamma_W\})$. (2) is immediate with (ii) since naturally we have $K_l \in Ks(t_{sl})$.

(2.) follows from the symmetric argument. ☐

With Lemma 6.7.3 it is then clear:

Theorem 6.7.2 (Wedge Theorem). *Let $(\mathcal{N}, Cover)$ be a live SSMD fc system, W a proper wedge of N, $K_l \in \mathcal{K}^{sp}_{W\text{-}l}$, and $K_r \in \mathcal{K}^{sp}_{W\text{-}r}$. We have:*
K_W is critical w.r.t. K_l and K_r.

Proof. Immediate with Lemma 6.7.3 and Lemma 6.7.4. ☐

6.7.3 The WNL Theorem

We can now join together the statements of the Link Corollary and the Wedge Theorem, and thereby arrive at the WNL Theorem (proper W̲edge has N̲o direct L̲ink); informally it says:

> In live SSMD fc systems, opposite synchronization components of a proper wedge W can only be linked via K_W.

Theorem 6.7.3 (WNL Theorem). *Let $(\mathcal{N}, Cover)$ be a live SSMD fc system, W a proper wedge of N, $K_l \in \mathcal{K}^{sp}_{W\text{-}l}$, and $K_r \in \mathcal{K}^{sp}_{W\text{-}r}$.*

1. *For any link λ from K_l to K_r we have $K_W \in \mathcal{K}_\lambda$.*
2. *For any link λ from K_r to K_l we have $K_W \in \mathcal{K}_\lambda$.*

Proof. Follows directly from Corollary 6.7.1 and Theorem 6.7.2. ☐

6.8 The cp Transition System on SSMD fc Systems

We now return to issues that directly concern cp bisimilarity. In the following we investigate the cp transition system $\mathcal{T}_{cp}$ in the context of SSMD fc systems. This will provide us with insights about the matching in $\sim_{cp}$, which will be required in the following three sections. We work in the context of two SSMD fc systems $(\mathcal{N}_1, Cover_1)$, $(\mathcal{N}_2, Cover_2)$, and build on the setting of Section 6.4. If not stated otherwise or resolved by an earlier occurrence, i will range over $\{1, 2\}$.

First, we observe that preserving the compositionality in the spirit of $\mathcal{T}_{cp}$ (and thus $\sim_{cp}$) entails that the static compositionality given by a strict cover will be respected to a certain degree: (1) any joint process β naturally induces a *component match* $cm \cdot \beta$; (2) if β evolves into a new process via a cp transition, then, although in general $cm \cdot \beta$ will not be preserved, any change to $cm \cdot \beta$ will occur in a 'controlled fashion'.

After that, to obtain further insights, we define *trace functions* to capture the involvement of a set of components $\mathcal{K}_i \subseteq Cover_i$ in a joint process, or jfs. On the one hand, we find that in many ways our trace functions behave similarly to the projection functions on $Proc_i$ and $Runs_i$ (cf. Section 6.2.5.3); in particular, the corresponding propositions give us a tool for composing new jfs' by shuffling component traces of given jfs'. We will make use of this mainly in Section 6.9.

On the other hand, our trace functions help us to set up a connection between the matching in cp bisimilarity and the topology of the related systems. By fixing a component $K_i \in Cover_i$ and tracing its involvement in a jfs, we can gain information about the static structure of the opposite net $N_{\bar{i}}$; in particular, we shall infer specific simple links and TFS-paths. These findings prepare for the crucial application of the WNL Theorem in Section 6.10.

Component Match. At any process $M_i \in Proc_i$, the decomposition function K_i establishes a 1-to-1 correspondence between the subprocesses of M_i and the components of $Cover_i$: by strictness each $p_i \in M_i$ represents exactly one $K_i \in Cover_i$[7]; conversely, as we know from Prop. 6.2.6, each $K_i \in Cover_i$ is represented by exactly one $p_i \in M_i$. But then, any bijection between two processes $M_1 \in Proc_1$, $M_2 \in Proc_2$ naturally induces a bijection between $Cover_1$ and $Cover_2$; — any $\beta \in JProc$ gives rise to a *component match* in the following way:

[7] As convenient as the use of 'K' (or 'K_i') to denote both, the respective decomposition function and a fixed component, otherwise is, here it is unfortunate terminology. It will always be clear from the context, though, in which meaning 'K' (or 'K_i') is employed.

Definition 6.8.1. Let $\beta \in JProc$. Set $\beta_1 = \beta$, and $\beta_2 = \beta^{-1}$. The *component match induced by* β_i, $cm \cdot \beta_i : Cover_i \rightarrow Cover_{\bar{i}}$, is given by: $cm \cdot \beta_i(K_i) = K_{\bar{i}}(\beta_i(proj_i(\beta)(K_i)))$.

Proposition 6.8.1. *Let* $\beta \in JProc$.

1. $cm \cdot \beta_{\bar{i}} \circ cm \cdot \beta_i = \mathsf{id}_{Cover_i}$.

2. $cm \cdot \beta_i$ *is a bijection, and* $(cm \cdot \beta_i)^{-1} = cm \cdot \beta_{\bar{i}}$.

Proof. (1.) follows directly from the definitions. (2.) is immediate with (1.). □

Naturally, we also have:

Proposition 6.8.2. *Let* $\beta \in JProc$. $K_{\bar{i}} \circ \beta_i = cm \cdot \beta_i \circ K_i$.

Proof. This follows easily with Prop. 6.2.12(1). □

Convention 6.8.1. Let $\beta \in JProc$. If the interpretation is clear from the context we shall ignore the index i of $cm \cdot \beta_i$. Similarly, we shall discard the index i of the decomposition functions K_i and Ks_i if the meaning is resolved by the argument. In the context of a *joint component* $K \in cm \cdot \beta$, we set $K_i = proj_i(K)$, and analogously, in the context of a set of joint components $\mathcal{K}_i \subseteq cm \cdot \beta$ we set $\mathcal{K}_i = proj_i(\mathcal{K})$.

By design of $\mathcal{T}_{cp}$, a transition $\beta \overset{(t_1,t_2)}{\rightarrow}_{cp} \beta'$ respects the match of subprocesses given by β in that t_1 has the same (with respect to β) local effect on $proj_1(\beta)$ as t_2 has on $proj_2(\beta)$, and vice versa. Clearly, this does not guarantee that the corresponding component match will be preserved: in general, $cm \cdot \beta'$ can be different from $cm \cdot \beta$. However, component matches will be respected to a certain degree; namely as follows:

- At both β and β', the components involved in t_1 are matched to the components involved in t_2, and vice versa.

- Components not involved in t_1 are matched at β' exactly as at β, and vice versa.

Proposition 6.8.3. *Let* $\beta \overset{(t_1,t_2)}{\rightarrow}_{cp} \beta'$, *and set* $\mathcal{K}_{t_i} = Ks(t_i)$. *We have:*

1. $cm \cdot \beta(\mathcal{K}_{t_i}) = Ks(t_{\bar{i}})$,

2. $cm \cdot \beta'(\mathcal{K}_{t_i}) = Ks(t_{\bar{i}})$, *and*

3. $\forall K_i \in Cover_i \backslash \mathcal{K}_{t_i}.\ cm \cdot \beta'(K_i) = cm \cdot \beta(K_i)$.

Proof. Considering the definition of $\rightarrow_{cp}$ (Def. 6.4.2), (1.) follows with Prop. 6.8.2 and Prop. 6.2.2(3a), (2.) with Prop. 6.8.2, and Prop. 6.2.2(3b), and (3.) with Prop. 6.2.2(3a) and the definition of $cm\cdot\beta$. $\square$

We can refine this observation by taking the type of t_i into consideration; with Prop. 6.2.4 it is easy to see:

- If t_i is of type switch then $t_{\bar{i}}$ is of type switch and the component match will be preserved for all components.
- If t_i is of type synch then $t_{\bar{i}}$ is of type synch and the component match may change while complying to the rule given by Prop. 6.8.3.

From Prop. 6.4.10 we already know that in $\mathcal{T}_{cp}$ the type of transitions is respected, and so we only formalize the first insight:

Proposition 6.8.4. *Let $\beta \overset{(t_1,t_2)}{\rightarrow}_{cp} \beta'$. If t_i is of type switch then we have:*

1. *$t_{\bar{i}}$ is of type switch, and*
2. *$\forall K_i \in Cover_i.\ cm\cdot\beta'(K_i) = cm\cdot\beta(K_i)$.*

Proof. This is immediate with Prop. 6.8.3 and Prop. 6.2.4. $\square$

For jfs' we then obtain:

Proposition 6.8.5. *Let $\beta \in JProc$, and $\sigma \in JFS(\beta)$.*

1. *Let $K \in cm\cdot\beta$, and set $\mathcal{R} = cm\cdot\beta \backslash K$.*

$$\begin{aligned}&nosynch(K_i, proj_i(\sigma)) \implies \\ &nosynch(K_{\bar{i}}, proj_{\bar{i}}(\sigma))\ \&\ cm\cdot last(\sigma)(K_i) = K_{\bar{i}}\ \&\ cm\cdot last(\sigma)(\mathcal{R}_i) = \mathcal{R}_{\bar{i}}.\end{aligned}$$

2. *Let $\mathcal{K} \subseteq cm\cdot\beta$.*

$$Ks(proj_i(\sigma)) \subseteq \mathcal{K}_i \implies Ks(proj_{\bar{i}}(\sigma)) \subseteq \mathcal{K}_{\bar{i}}\ \&\ cm\cdot last(\sigma)(\mathcal{K}_i) \subseteq \mathcal{K}_{\bar{i}}.$$

Proof. This is straightforward by induction on the length of σ: (1.) follows with Prop. 6.8.3 and Prop. 6.8.4; for (2.) Prop. 6.8.3 is sufficient. $\square$

Tracing Components. As announced, we now define trace functions to capture the involvement of a set of components $\mathcal{K}_i \subseteq Cover_i$ in a joint process or jfs. We start with the trace function for *JProc*:

Definition 6.8.2. We define a *trace function* $\uparrow$: $JProc \times \mathcal{P}(Cover_i) \to \mathcal{P}(P_1 \times P_2)$ to trace the involvement of a set of components in a joint process as follows:

$$\beta \uparrow \mathcal{K}_i = \{(p_1, p_2) \in \beta \mid K(p_i) \in \mathcal{K}_i\}.$$

We collect some straightforward observations:

Proposition 6.8.6. *Let $\beta \in JProc$, and $\mathcal{K}_i \subseteq Cover_i$.*

1. $proj_i(\beta \uparrow \mathcal{K}_i) = proj_i(\beta) \uparrow \mathcal{K}_i$.
2. $proj_{\bar{i}}(\beta \uparrow \mathcal{K}_i) = \beta_i(proj_i(\beta) \uparrow \mathcal{K}_i)$.
3. $Ks_{\bar{i}}(proj_{\bar{i}}(\beta \uparrow \mathcal{K}_i)) = cm \cdot \beta(\mathcal{K}_i)$.
4. *Let* $\mathcal{K} \subseteq cm \cdot \beta$. $\beta \uparrow \mathcal{K}_1 = \beta \uparrow \mathcal{K}_2$.
5. *Let* $\beta' \in JProc$, *and* $p_i \in P_i \uparrow \mathcal{K}_i$ *such that* $p_i \in proj_i(\beta) \cap proj_i(\beta')$.
   $\beta \uparrow \mathcal{K}_i = \beta' \uparrow \mathcal{K}_i \implies \beta(p_i) = \beta'(p_i)$.

Proof. Easy with the definitions. ☐

Proposition 6.8.7. *Let $\beta \in JProc$, and $\mathcal{K}_i \subseteq Cover_i$. For all $\sigma \in JFS(\beta \uparrow \mathcal{K}_i)$ we have: $Ks(proj_i(\sigma)) \subseteq \mathcal{K}_i$.*

Proof. This follows from Prop. 6.4.2, Prop. 6.8.6(1), and Prop. 6.2.21(2). ☐

Our trace function behaves similarly to the projection function on $Proc_i$; analogously to Prop. 6.2.18(1) and (2) we have:

Proposition 6.8.8. *Let $\beta \xrightarrow{t}_{cp} \beta'$, and $\mathcal{K}_i \subseteq Cover_i$.*

$$Ks(t_i) \subseteq \mathcal{K}_i \implies \beta \uparrow \mathcal{K}_i \xrightarrow{t}_{cp} \beta' \uparrow \mathcal{K}_i.$$

Proof. This follows easily: consider that $Ks(t_i) \subseteq \mathcal{K}_i$ implies $Ks({}^\bullet t_i) \subseteq \mathcal{K}_i$ and $Ks(t_i{}^\bullet) \subseteq \mathcal{K}_i$ (Prop. 6.2.2(2)), and employ Prop. 6.4.1. ☐

Proposition 6.8.9. *Let $\mathcal{K}_i \subseteq Cover_i$.*

1. *Let* $\beta \xrightarrow{t}_{cp} \beta'$. $Ks(t_i) \cap \mathcal{K}_i = \emptyset \implies \beta \uparrow \mathcal{K}_i = \beta' \uparrow \mathcal{K}_i$.
2. *Let* $\sigma \in JFS$. $\mathcal{K}_i \cap Ks(proj_i(\sigma)) = \emptyset \implies last(\sigma) \uparrow \mathcal{K}_i = first(\sigma) \uparrow \mathcal{K}_i$.

Proof. (1.) Analogously to the previous proposition: by Prop. 6.2.2(2) we obtain $Ks({}^\bullet t_i) \cap \mathcal{K}_i = \emptyset = Ks(t_i{}^\bullet) \cap \mathcal{K}_i$; then simply consider Prop. 6.4.1. (2.) This easily follows from (1.) by induction on the length of σ. □

Justified by Prop. 6.8.9(1) we can then define our trace function for jfs' as follows:

Definition 6.8.3. We define a *trace function* $\uparrow$: $JFS \times \mathcal{P}(Cover_i) \rightarrow JPTS$ to trace the course of a set of components in a jfs inductively by:

$$\begin{array}{rcl} \beta\!\uparrow\!\mathcal{K}_i & = & \beta\!\uparrow\!\mathcal{K}_i \quad (\uparrow\!\mathcal{K}_i \text{ for } \mathit{JProc} \text{ as above}), \\ (\sigma(t_1, t_2)\beta)\!\uparrow\!\mathcal{K}_i & = & \left\{ \begin{array}{ll} (\sigma\!\uparrow\!\mathcal{K}_i)(t_1, t_2)(\beta\!\uparrow\!\mathcal{K}_i) & \text{if } Ks(t_i) \cap \mathcal{K}_i \neq \emptyset, \\ \sigma\!\uparrow\!\mathcal{K}_i & \text{otherwise.} \end{array} \right. \end{array}$$

Naturally, we have:

Proposition 6.8.10. *Let $L \subseteq JFS$, and $\mathcal{K}_i \subseteq Cover_i$. If L is prefix-closed then $L\!\uparrow\!\mathcal{K}_i$ is also prefix-closed.*

Proof. Easy by applying the definitions. □

Proposition 6.8.11. *Let $\sigma \in JFS$, and $\mathcal{K}_i \subseteq Cover_i$. $last(\sigma)\!\uparrow\!\mathcal{K}_i = last(\sigma\!\uparrow\!\mathcal{K}_i)$.*

Proof. Straightforward by induction on the length of σ. □

Our trace function for jfs' behaves in many ways analogously to the projection function on $Runs_i$. Moreover, with our insights of the previous paragraph it is not difficult to show that under certain circumstances our trace function can be understood as a projection of jfs' onto two joint sets of components:

Proposition 6.8.12. *Let $\sigma \in JFS$, and set $\beta = first(\sigma)$.*

1. *Let $K \in cm\cdot\beta$, and $\mathcal{R} = cm\cdot\beta\backslash K$. If $nosynch(K_i, proj_i(\sigma))$ then we have:*

   (a) $\sigma\!\uparrow\!K_1 = \sigma\!\uparrow\!K_2$, *and*

   (b) $\sigma\!\uparrow\!\mathcal{R}_1 = \sigma\!\uparrow\!\mathcal{R}_2$.

2. *Let $\mathcal{K} \subseteq cm\cdot\beta$. If $Ks(proj_i(\sigma)) \subseteq \mathcal{K}_i$ then we have: $\sigma\!\uparrow\!\mathcal{K}_1 = \sigma\!\uparrow\!\mathcal{K}_2$.*

Proof. (1.) This is straightforward by induction on the length of σ when employing Prop. 6.8.6(4), Prop. 6.8.3(1),(3), and Prop. 6.8.5(1). (2.) Similarly, but use the second part of Prop. 6.8.5 instead of the first. □

The following two propositions are analogous to Prop. 6.2.22 and 6.2.23. Note how together they provide a tool that allows us to infer new matchings by shuffling traces of given matchings. We will make use of this in the following section.

Proposition 6.8.13. *Let $\sigma \in JFS$, and set $\beta = first(\sigma)$.*

1. *Let $K \in cm{\cdot}\beta$, and $\mathcal{R} = cm{\cdot}\beta \backslash K$. If $nosynch(K_i, proj_i(\sigma))$ then we have:*
   (a) *$\sigma{\uparrow}K_i \in JFS(\beta{\uparrow}K_i)$ & $last(\sigma{\uparrow}K_i) = last(\sigma){\uparrow}K_i$, and*
   (b) *$\sigma{\uparrow}\mathcal{R}_i \in JFS(\beta{\uparrow}\mathcal{R}_i)$ & $last(\sigma{\uparrow}\mathcal{R}_i) = last(\sigma){\uparrow}\mathcal{R}_i$.*

2. *Let $\mathcal{K} \subseteq cm{\cdot}\beta$. If $Ks(proj_i(\sigma)) \subseteq \mathcal{K}_i$ then we have:*

$$\sigma{\uparrow}\mathcal{K}_i \in JFS(\beta{\uparrow}\mathcal{K}_i) \ \&\ last(\sigma{\uparrow}\mathcal{K}_i) = last(\sigma){\uparrow}\mathcal{K}_i.$$

Proof. (1.) and (2.) are both straightforward by induction on the length of σ: for (1.) employ Prop. 6.8.8 and Prop. 6.8.9(1); for (2.) Prop. 6.8.8 is sufficient. □

Proposition 6.8.14. *Let $\beta \in JProc$, and $\mathcal{K}_i$, $\mathcal{K}'_i \subseteq Cover_i$ such that $\mathcal{K}_i \cap \mathcal{K}'_i = \emptyset$ and $\mathcal{K}_i \cup \mathcal{K}'_i = Cover_i$. Then for any $L \subseteq JFS(\beta{\uparrow}\mathcal{K}_i)$, $L' \subseteq JFS(\beta{\uparrow}\mathcal{K}'_i)$ we have:*

$$L \otimes L' \subseteq JFS(\beta).$$

Proof. It is easy to see that for any $\beta \in JProc$, $\mathcal{K}_i$, $\mathcal{K}'_i \subseteq Cover_i$ the following holds: (1) $\mathcal{K}_i \cap \mathcal{K}'_i = \emptyset \implies \beta{\uparrow}\mathcal{K}_i \cap \beta{\uparrow}\mathcal{K}'_i = \emptyset$. (2) $\mathcal{K}_i \cup \mathcal{K}'_i = Cover_i \implies \beta{\uparrow}\mathcal{K}_i \cup \beta{\uparrow}\mathcal{K}'_i = \beta$. But then the proposition immediately follows from Prop. 6.4.15. □

Finally, we present a straightforward connection between our trace function and shuffle:

Proposition 6.8.15. *Let $\beta \in JProc$, $K \in cm{\cdot}\beta$, and set $\mathcal{R} = cm{\cdot}\beta \backslash K$.*

1. *Let $\sigma_K \in JFS(\beta{\uparrow}K_i)$, $\sigma_{\mathcal{R}} \in JFS(\beta{\uparrow}\mathcal{R}_i)$. For all $\sigma \in \sigma_K \otimes \sigma_{\mathcal{R}}$ we have: $\sigma_K = \sigma{\uparrow}K_i$ & $\sigma_{\mathcal{R}} = \sigma{\uparrow}\mathcal{R}_i$.*

2. *Let $\mathcal{B}_K \subseteq JFS(\beta{\uparrow}K_i)$, and $\mathcal{B}_{\mathcal{R}} \subseteq JFS(\beta{\uparrow}\mathcal{R}_i)$. For all $\sigma \in \mathcal{B}_K \otimes \mathcal{B}_{\mathcal{R}}$ we have: $\sigma{\uparrow}K_i \in \mathcal{B}_K$ & $\sigma{\uparrow}\mathcal{R}_i \in \mathcal{B}_{\mathcal{R}}$.*

Proof. (1.) This is easy to see with Prop. 6.8.7 and the definitions. (2.) follows from (1). □

Gaining Topological Entities. We now show how by tracing a single component $K_i \in Cover_i$ in a jfs $\sigma \in JFS$ we gain information about the topology of the opposite system: $\sigma{\uparrow}K_i$ induces a PP-path and a corresponding simple link in $N_{\bar{i}}$. First of all, note that for a single component we certainly have:

Proposition 6.8.16. *Let $\beta \in JProc$, and $K_i \in Cover_i$. $|\beta{\uparrow}K_i| = 1$.*

Proof. Obvious with Prop. 6.2.6. □

Thus, we can adopt the following convention:

Convention 6.8.2. Let $\beta \in \mathit{JProc}$, $\sigma \in \mathit{JFS}$, and $K_i \in \mathit{Cover}_i$. We shall identify $\beta \uparrow K_i$ with the joint place $p \in P_1 \times P_2$ given by $\beta \uparrow K_i = \{p\}$ (considering Prop. 6.8.16). Accordingly, we shall understand $\sigma \uparrow K_i$ as an element of $(P_1 \times P_2)(T(P_1 \times P_2))^*$. It will always be clear from the context whether we assume the conventional or this new interpretation of $\beta \uparrow K_i$, and $\sigma \uparrow K_i$ respectively.

It is now easy to anticipate: we will show that the $\bar{\imath}$-part of $\sigma \uparrow K_i$ is a PP-path. Since our standard projection function proj_i for jpts' abstracts away from places, we define a second projection function to get hold of the full $\bar{\imath}$-part of an entity $\sigma \uparrow K_i$:

Definition 6.8.4. We define a projection function $\mathit{proj}_i^* : (P_1 \times P_2)(T(P_1 \times P_2))^* \to P_i\,(T_i\,P_i)^*$ inductively by:

$$\begin{array}{rcl} \mathit{proj}_i^*((p_1, p_2)) & = & p_i, \\ \mathit{proj}_i^*(\sigma(t_1, t_2)(p_1, p_2)) & = & \mathit{proj}_i^*(\sigma)\, t_i\, p_i. \end{array}$$

We use a function $ts : P_i\,(T_i\,P_i)^* \to T_i^*$ to extract the transitions of a 'place/transition' sequence in the obvious way.

Proposition 6.8.17. *Let $\sigma \in \mathit{JFS}$, and $K_i \in \mathit{Cover}_i$. We have:*

$$\mathit{proj}_{\bar{\imath}}(\sigma \uparrow K_i) = ts(\mathit{proj}_{\bar{\imath}}^*(\sigma \uparrow K_i)).$$

Proof. Obvious with the definitions. □

Now, we indeed prove:

Proposition 6.8.18. *Let $\sigma \in \mathit{JFS}$, and $K_i \in \mathit{Cover}_i$; we set $\beta = \mathit{first}(\sigma)$, and $\beta' = \mathit{last}(\sigma)$.*

1. *$\mathit{proj}_{\bar{\imath}}^*(\sigma \uparrow K_i)$ is a PP-path in $N_{\bar{\imath}}$ that leads from $\mathit{proj}_{\bar{\imath}}(\beta \uparrow K_i)$ to $\mathit{proj}_{\bar{\imath}}(\beta' \uparrow K_i)$.*

2. *The path of (1), say $\pi_{\bar{\imath}}$, induces a simple link $\lambda_{\bar{\imath}}$ such that*

   (a) $p^{in}_{\lambda_{\bar{\imath}}} = \mathit{proj}_{\bar{\imath}}(\beta \uparrow K_i)$ & $p^{fi}_{\lambda_{\bar{\imath}}} = \mathit{proj}_{\bar{\imath}}(\beta' \uparrow K_i)$,

   (b) $K^{in}_{\lambda_{\bar{\imath}}} = cm \cdot \beta(K_i)$ & $K^{fi}_{\lambda_{\bar{\imath}}} = cm \cdot \beta'(K_i)$, and

   (c) $\mathcal{K}_{\lambda_{\bar{\imath}}} \subseteq Ks(\pi_{\bar{\imath}})$.

Proof. (1.) This is straightforward by induction on the length of σ. For the inductive case $\sigma \equiv \sigma'(t_1, t_2)\beta'$ consider the following two cases: (a) $K_i \notin Ks(t_i)$, and (b) $K_i \in Ks(t_i)$. If (a) holds then (1.) follows by induction hypothesis and Prop. 6.8.9(1). In the case of (b), (1.) is immediate by induction hypothesis and this property: $proj_{\bar{i}}(last(\sigma') \uparrow K_i) \in {}^\bullet t_{\bar{i}}$ & $proj_{\bar{i}}(\beta' \uparrow K_i) \in t_{\bar{i}}{}^\bullet$. The latter can be obtained by employing Prop. 6.4.1 and Prop. 6.2.12(3).

(2.) It is easy to see: any PP-path π can uniquely be divided up into one WS- and $n \geq 0$ FOS-segments, and thus induces a simple link λ. Furthermore, λ will satisfy: $p_\lambda^{in} = first(\pi)$, $p_\lambda^{fi} = last(\pi)$ & $\mathcal{K}_\lambda \subseteq Ks(\pi)$. Clearly, this implies (2.) up to (a) and (c); (b) follows with (a) and Prop. 6.8.6(3). $\square$

Definition 6.8.5. Let $\sigma \in JFS$, and $K_i \in Cover_i$. We call the PP-path given by Prop. 6.8.18(1) $path_{\bar{i}}(\sigma \uparrow K_i)$, and the link given by (2) $link_{\bar{i}}(\sigma \uparrow K_i)$.

Set $K_{\bar{i}} = cm\text{·}first(\sigma)(K_i)$. From our previous observations we know: if K_i does not synchronize on $proj_i(\sigma)$ then K_i and $K_{\bar{i}}$ stay matched to each other during σ, and $K_{\bar{i}}$ does not synchronize on $proj_{\bar{i}}(\sigma)$, either. This allows us to gain further information about the path and link induced by tracing K_i, or a component of $Cover_i$ other than K_i, on σ. First of all, we prove:

Proposition 6.8.19. *Let $\sigma \in JFS$, $K_i \in Cover_i$, and set $K_{\bar{i}} = cm\cdot first(\sigma)(K_i)$. If $nosynch(K_i, proj_i(\sigma))$ then we have:*

1. *$Ks(proj_{\bar{i}}(\sigma \uparrow K_i)) \subseteq \{K_{\bar{i}}\}$.*

2. *Let $\mathcal{K}_i \subseteq Cover_i \backslash K_i$. $K_{\bar{i}} \notin Ks(proj_{\bar{i}}(\sigma \uparrow \mathcal{K}_i))$.*

Proof. Let σ, K_i, $K_{\bar{i}}$ be given as above, and assume $nosynch(K_i, proj_i(\sigma))$. Note that by Prop. 6.8.5(1) we also have $nosynch(K_{\bar{i}}, proj_{\bar{i}}(\sigma))$. We shall employ this fact, and refer to it by (A).

(1.) With (A) it is easy to see that $Ks(proj_{\bar{i}}(\sigma \uparrow K_{\bar{i}})) \subseteq \{K_{\bar{i}}\}$. But since by Prop. 6.8.12(1a) $\sigma \uparrow K_1 = \sigma \uparrow K_2$, this immediately proves (1.).

(2.) Set $\mathcal{R}_i = Cover_i \backslash K_i$, and $\mathcal{R}_{\bar{i}} = cm\cdot first(\sigma)(\mathcal{R}_i)$. Clearly, $K_{\bar{i}} \notin \mathcal{R}_{\bar{i}}$, and thus with (A) we obtain $K_{\bar{i}} \notin Ks(proj_{\bar{i}}(\sigma \uparrow \mathcal{R}_{\bar{i}}))$. By Prop. 6.8.12(1b) this gives us $K_{\bar{i}} \notin Ks(proj_{\bar{i}}(\sigma \uparrow \mathcal{R}_i))$. But note how this already establishes (2.): clearly, for any $\mathcal{K}_i \subseteq \mathcal{R}_i$ we have $Ks(proj_{\bar{i}}(\sigma \uparrow \mathcal{K}_i)) \subseteq Ks(proj_{\bar{i}}(\sigma \uparrow \mathcal{R}_i))$. $\square$

We can now refine Prop. 6.8.18 as follows.

Proposition 6.8.20. *Let $\sigma \in JFS$, $K_i \in Cover_i$, $\beta = first(\sigma)$, and $\beta' = last(\sigma)$. If $nosynch(K_i, proj_i(\sigma))$ then we have:*

1. $path_{\bar{i}}(\sigma\!\uparrow\! K_i)$ *is a WS-path.*

2. *Let* $K_i' \in Cover_i$ *such that* $K_i' \neq K_i$.

   (a) $cm\cdot\beta(K_i) \notin Ks(path_{\bar{i}}(\sigma\!\uparrow\! K_i'))$, *and*

   (b) $cm\cdot\beta(K_i) \notin \mathcal{K}_{\lambda_{\bar{i}}}$, *where* $\lambda_{\bar{i}} = link_{\bar{i}}(\sigma\!\uparrow\! K_i')$.

Proof. (1.) is immediate with Prop. 6.8.19(1), 6.8.17, and 6.2.4. (2.) (a) follows with Prop. 6.8.19(2), 6.8.17, and 6.2.15. (b) is immediate with (a) and Prop. 6.8.18(2c). $\square$

Additionally making use of Prop. 6.4.11, from the first part of the previous proposition we further obtain:

Proposition 6.8.21. *Let* $\sigma \in JFS$, $K_i \in Cover_i$, $\beta = first(\sigma)$, *and* $\beta' = last(\sigma)$.
If we have (0) $nosynch(K_i, proj_i(\sigma))$,
(i) $proj_i(\beta)(K_i)$ *is of type switch,*
(ii) $p_i' \equiv proj_i(\beta')(K_i)$ *is of type synch, and*
(iii) $\sigma \overset{(t_1,t_2)}{\rightarrow}_{cp}$ *for* t_i *given by* $p_i'^{\bullet} = \{t_i\}$, *and some* $t_{\bar{i}} \in T_{\bar{i}}$,
then $path_{\bar{i}}(\sigma\!\uparrow\! K_i)$ *is a TFS-path.*

Proof. Let σ, K_i, β, and β' be given as above, and suppose that conditions (0)-(iii) are satisfied; set $\pi_{\bar{i}} = path_{\bar{i}}(\sigma\!\uparrow\! K_i)$, and $p_i = proj_i(\beta)(K_i)$. By Prop. 6.8.20(1) and (0) it is clear that $\pi_{\bar{i}}$ is a WS-path. We further need to show: (A) $first(\pi_{\bar{i}}) = proj_{\bar{i}}(\beta\!\uparrow\! K_i)$ is a switch place, and (B) $last(\pi_{\bar{i}}) = proj_{\bar{i}}(\beta'\!\uparrow\! K_i)$ is a synch place. By Prop. 6.4.11 and 6.8.6(2), (B) is a consequence of (ii) and (iii). In turn, (A) will follow by the same propositions and (i) if we can exhibit $\sigma'(t_1', t_2') \in JFS(\beta)$ such that $t_i' \in p_i^{\bullet}$.

By (i), (ii), and Prop. 6.8.6(1) we clearly have $\beta \uparrow K_i \neq \beta' \uparrow K_i$, and with Prop. 6.8.9(2) we obtain $K_i \in Ks(proj_i(\sigma))$. Then, considering condition (0) we can assume a prefix $\sigma'(t_1', t_2')$ of σ such that $Ks(t_i') = \{K_i\}$, and $K_i \notin Ks(proj_i(\sigma'))$. But by Prop. 6.8.9(2), 6.8.6(1), and 6.2.12(3) it is easy to see that $t_i' \in p_i^{\bullet}$, and altogether we have found entities $\sigma'(t_1, t_2)$ as required. $\square$

Prop. 6.8.20(2) and Prop. 6.8.21 will be crucial with respect to our application of the WNL Theorem in Section 6.10.

6.9 Is cp Bisimilarity K-decomposable?

We are now coming closer to our main results about cp bisimilarity on live SSMD fc systems: in Section 6.10 we will prove the SWFSI Matching Theorem; with its help we will then show in Section 6.11 that cp bisimilarity is K-decomposable, and further that cp bisimilarity is sw-(1)coherent and sw-(1)hereditary. This section gives introductory material: we present the concept of *K-decomposability*, and provide a first analysis of whether cp bisimilarity satisfies this property. In particular, we are led to the *Crucial Subgoal^K*, and sketch how it will follow if we can prove that swfsi-matching is deterministic.

In more detail, the section is organized as follows: first, we introduce a basic decomposition view for systems, the so-called *'K-split' view*. In the second part, we derive K-decomposability by translating the 'K-split' view into a decomposition property for cp bisimilarity. Then follow three parts with basic insights. Building on these we come up with the Crucial Subgoal^K, and sketch how it can be reduced to the subgoal "swfsi-matching is deterministic".

In the first paragraph we assume a SSMD fc system $(\mathcal{N}, Cover)$ with the setting of Section 6.2.5.3. Otherwise we work in the context of two live SSMD fc systems $(\mathcal{N}_1, Cover_1)$, $(\mathcal{N}_2, Cover_2)$, and build on the framework of the previous section. (To be precise, liveness will only be required from paragraph (5) onwards.)

(1.) The 'K-split' View. Assume a process $M \in Proc$, a component $K \in Cover$, and set $\mathcal{R} = Cover \backslash K$. We can naturally divide M into a *K-part*, and a *remaining part* as follows:

Definition 6.9.1. We define the *K-part of M* to be $M_K = M \uparrow K$, and the *remaining part of M (w.r.t. K)* to be $M_{R(K)} = M \uparrow \mathcal{R}$; we fix that M_K and $M_{R(K)}$ are always interpreted with respect to N.

With our observations of Section 6.2.5.3 it is easy to see: M_K characterizes the share of K in the behaviour of M up to synch of K, in turn $M_{R(K)}$ represents the share of $\mathcal{R}$ in the behaviour of M up to synch of K, and naturally these processes compute independently of each other. On the other hand, since the split into K and $\mathcal{R}$ covers all components of $Cover$, the behaviour of M up to synch of K exhaustively falls into the behaviour of M_K and the behaviour of $M_{R(K)}$. Altogether, this gives us our basic decomposition insight, the *'K-split' view*:

Up to synchronization of K, the process M behaves like the parallel composition of its subprocesses M_K and $M_{R(K)}$.

Definition 6.9.2. The *behaviour of M up to synchronization of K* is defined by: $\mathit{UpToSynch}(M, K) = \{r \in \mathit{Runs}_N(M) \mid \mathit{nosynch}(K, r)\}$.

Proposition 6.9.1 ('K-split' view).

$$\mathit{UpToSynch}(M, K) = \mathit{Runs}_N(M_K) \otimes \mathit{Runs}_N(M_{R(K)}).$$

Proof. The '$\subseteq$'-direction follows with Prop. 6.2.22(1) and the obvious fact that N is covered by K and $\mathcal{R}$. The '$\supseteq$'-direction is immediate with Prop. 6.2.23 and 6.2.21(1). $\square$

(2.) K-decomposability. We are going to translate the 'K-split' view into a decomposition property for cp bisimilarity; this is how we obtain the concept of *K-decomposability.* Assume a joint process $\beta \in \mathit{JProc}$, a joint component $K \in cm{\cdot}\beta$, and set $\mathcal{R} = cm{\cdot}\beta \backslash K$. Analogously to Def. 6.9.1 we divide β into a *K-part* and a *remaining part*:

Definition 6.9.3. We define the *K-part of β* to be $\beta_K = \beta \uparrow K_i$, and the *remaining part of β (w.r.t. K)* to be $\beta_{R(K)} = \beta \uparrow \mathcal{R}_i$, where $i = 1$, or 2 equivalently (considering Prop. 6.8.6(4)). We fix that β_K and $\beta_{R(K)}$ are always interpreted with respect to N_1 and N_2.

If the 'K-split' view translates into cp bisimilarity then we will naturally expect:

> Whenever $\beta \in \sim_{cp}$ there exists a cp bisimulation $\mathcal{B}$ for β such that the part of $\mathcal{B}$ that covers the behaviour up to synchronization of K_1, and K_2 respectively, is uniformly composed of a cp bisimulation for β_K, and a cp bisimulation for $\beta_{R(K)}$.

Note that we are not content with merely requiring the existence of cp bisimulations for β_K and β_R; this would ignore all behaviour from the point of K-synchronization onwards, and only give us a decomposition property at the level of SBPP.

Due to Prop. 6.8.5(1), runs of M_1 without synch of K_1 have to be matched against runs of M_2 without synch of K_2, and vice versa. This means, matches up to synch of K_1 and K_2 can very naturally be captured as follows:

Definition 6.9.4. Let $\mathcal{B}$ be a cp bisimulation for β. We define the *matching up to synchronization of K in $\mathcal{B}$* by:

$UpToSynch(\mathcal{B}, K) = \{\sigma \in \mathcal{B} \mid nosynch(K_i, proj_i(\sigma))\}$,
where $i = 1$, or 2 equivalently (considering Prop. 6.8.5(1)).

Now, we are ready to formulate:

Definition 6.9.5 (K-decomposability). Let $\mathcal{B}$ be a cp bisimulation for β. We say $\mathcal{B}$ is *decomposable w.r.t. K* (short: *K-decomposable*) iff there exist sets $\mathcal{B}_K$ and $\mathcal{B}_{R(K)}$ such that

1. $\mathcal{B}_K$ is a cp bisimulation for β_K,
2. $\mathcal{B}_{R(K)}$ is a cp bisimulation for $\beta_{R(K)}$, and
3. $UpToSynch(\mathcal{B}, K) = \mathcal{B}_K \otimes \mathcal{B}_{R(K)}$.

We say *cp bisimilarity is K-decomposable* iff for all $\beta \in \sim_{cp}$, and $K \in cm{\cdot}\beta$ there exists a cp bisimulation $\mathcal{B}$ for β which is K-decomposable.

(3.) First Basic Insight. First of all, we shall see: assuming $\beta \in \sim_{cp}$, it is straightforward to obtain cp bisimulations for β_K, and $\beta_{R(K)}$ respectively; we can read the required matching from any cp bisimulation $\mathcal{B}$ for β. There are even two ways of achieving this. On the one hand, we can extract cp bisimulations for β_K and $\beta_{R(K)}$ from $UpToSynch(\mathcal{B}, K)$:

Definition 6.9.6. Let $\mathcal{B}$ be a cp bisimulation for β. We define:

$$\begin{array}{ll} \mathcal{B}^u_K & = \{\sigma \!\uparrow\! K_i \mid \sigma \in UpToSynch(\mathcal{B}, K)\}, \text{ and} \\ \mathcal{B}^u_{R(K)} & = \{\sigma \!\uparrow\! \mathcal{R}_i \mid \sigma \in UpToSynch(\mathcal{B}, K)\}, \end{array}$$

where $i = 1$, or 2 equivalently (considering Prop. 6.8.12(1)).

Proposition 6.9.2. *Let $\mathcal{B}$ be a cp bisimulation for β.*

1. *$\mathcal{B}^u_K$ is a cp bisimulation for β_K.*
2. *$\mathcal{B}^u_{R(K)}$ is a cp bisimulation for $\beta_{R(K)}$.*

Proof. Let $\mathcal{B}$ be given as above.

(1.) We need to show: (1) $\mathcal{B}^u_K \subseteq JFS(\beta_K)$, (2) $\beta_K \in \mathcal{B}^u_K$, (3) $\mathcal{B}^u_K$ is prefix-closed, and (4) $\mathcal{B}^u_K$ satisfies the two bisimulation clauses of Def. 6.4.6. Since $\mathcal{B}$ is a cp bisimulation for β we clearly have: (i) $UpToSynch(\mathcal{B}, K) \subseteq JFS(\beta)$, (ii) $\beta \in UpToSynch(\mathcal{B}, K)$, and (iii) $UpToSynch(\mathcal{B}, K)$ is prefix-closed. Then (1) follows from (i) and Prop. 6.8.13(1a), (2) is immediate with (ii), and (3) is a consequence of (iii) and Prop. 6.8.10.

To prove that (4) holds, suppose $\sigma_K \in \mathcal{B}^u_K$ such that $(*)$ $proj_i(\sigma_K) \xrightarrow{t_i}$ for some $t_i \in T_i$, $i = 1, 2$. We need to find entities $t_{\bar{i}} \in T_{\bar{i}}$, $\sigma'_K \in JFS$ satisfying $\sigma_K \overset{(t_1,t_2)}{\rightarrow} \sigma'_K$, and $\sigma'_K \in \mathcal{B}^u_K$. By definition of $\mathcal{B}^u_K$ there must be $\sigma \in \mathcal{B}$ such that $\sigma_K = \sigma \uparrow K_i$. The latter entails $last(\sigma_K) = last(\sigma) \uparrow K_i$ (Prop. 6.8.11), and thus with $(*)$, Prop. 6.8.6(1), and Prop. 6.2.19 we obtain (a) $Ks(t_i) = \{K_i\}$, and (b) $proj_i(\sigma) \xrightarrow{t_i}$. Since $\mathcal{B}$ is a cp bisimulation, (b) implies we can assume $t_{\bar{i}} \in T_{\bar{i}}$, $\sigma' \in JFS$ such that (c) $\sigma \overset{(t_1,t_2)}{\rightarrow} \sigma'$, and (d) $\sigma' \in \mathcal{B}$. But clearly, $t_{\bar{i}}$ and $\sigma'_K \equiv \sigma' \uparrow K_i$ provide entities as required: by Prop. 6.8.13(1a) we have $\sigma'_K \in JFS$, and considering (a) and (c) we obtain $\sigma'_K = \sigma_K(t_1, t_2)(last(\sigma') \uparrow K_i)$; together this implies $\sigma_K \overset{(t_1,t_2)}{\rightarrow} \sigma'_K$. $\sigma'_K \in \mathcal{B}^u_K$ follows from (d) and the definition of $\mathcal{B}^u_K$.

(2.) This can be proved in analogous fashion; employ Prop. 6.8.13(1b) instead of (1a). $\square$

On the other hand, we can extract the required cp bisimulations from the *matching of 'K-only' behaviour*, and *'$\mathcal{R}$-only' behaviour* respectively:

Definition 6.9.7. Let $\mathcal{B}$ be a cp bisimulation for β.

Let $\mathcal{K} \subseteq cm{\cdot}\beta$. We define the *matching of '$\mathcal{K}$-only' behaviour in $\mathcal{B}$* by: $BehavOnly(\mathcal{B}, \mathcal{K}) = \{\sigma \in \mathcal{B} \mid Ks(proj_i(\sigma)) \subseteq proj_i(\mathcal{K}_i)\}$, where $i = 1$, or 2 equivalently (considering Prop. 6.8.5(2)).

Further, we define:

$$\begin{array}{ll} \mathcal{B}^o_K & = \{\sigma \uparrow K_i \mid \sigma \in BehavOnly(\mathcal{B}, K)\}, \text{ and} \\ \mathcal{B}^o_{R(K)} & = \{\sigma \uparrow \mathcal{R}_i \mid \sigma \in BehavOnly(\mathcal{B}, \mathcal{R})\}, \end{array}$$

where $i = 1$, or 2 equivalently (considering Prop. 6.8.12(2)).

Proposition 6.9.3. *Let $\mathcal{B}$ be a cp bisimulation for β.*

1. *$\mathcal{B}^o_K$ is a cp bisimulation for β_K.*

2. *$\mathcal{B}^o_{R(K)}$ is a cp bisimulation for $\beta_{R(K)}$.*

Proof. This can be proved analogously to Prop. 6.9.2; employ Prop. 6.8.13(2) instead of (1). $\square$

Clearly, we have:

Proposition 6.9.4. *Let $\mathcal{B}$ be a cp bisimulation for β.*

1. *BehavOnly*$(\mathcal{B}, K) \subseteq$ *UpToSynch*$(\mathcal{B}, K)$ *and* *BehavOnly*$(\mathcal{B}, \mathcal{R}) \subseteq$ *UpToSynch*$(\mathcal{B}, K)$*, and thus*

2. $\mathcal{B}^o_K \subseteq \mathcal{B}^u_K$ & $\mathcal{B}^o_{R(K)} \subseteq \mathcal{B}^u_{R(K)}$.

Proof. Obvious with Prop. 6.2.5. □

In general, the converse directions do not hold. However, if $\mathcal{B}$ is K-decomposable we do obtain $\mathcal{B}^u_K = \mathcal{B}^o_K$, and $\mathcal{B}^u_{R(K)} = \mathcal{B}^o_{R(K)}$. This is a consequence of the following:

Proposition 6.9.5. *Let $\mathcal{B}$ be a cp bisimulation for β, $\mathcal{B}_K \subseteq JFS(\beta_K)$, and $\mathcal{B}_{R(K)} \subseteq JFS(\beta_{R(K)})$. If $\mathcal{B}_K \otimes \mathcal{B}_{R(K)} =$ UpToSynch$(\mathcal{B}, K)$ then we have:*

1. $\mathcal{B}_K = \mathcal{B}^u_K = \mathcal{B}^o_K$*, and*

2. $\mathcal{B}_{R(K)} = \mathcal{B}^u_{R(K)} = \mathcal{B}^o_{R(K)}$.

Proof. Let $\mathcal{B}$, $\mathcal{B}_K$, and $\mathcal{B}_{R(K)}$ be given as above, and assume (A) $\mathcal{B}_K \otimes \mathcal{B}_{R(K)} =$ *UpToSynch*$(\mathcal{B}, K)$.

(1.) Considering Prop. 6.9.4(2), (1.) will follow if we can achieve (a) $\mathcal{B}_K \subseteq \mathcal{B}^o_K$, and (b) $\mathcal{B}^u_K \subseteq \mathcal{B}_K$.

To prove (a) assume $\sigma_K \in \mathcal{B}_K$. Since clearly $\beta \in$ *UpToSynch*$(\mathcal{B}, K)$, with (A) we obtain $\beta_{R(K)} \in \mathcal{B}_{R(K)}$, and further $\sigma \in$ *UpToSynch*$(\mathcal{B}, K)$, where σ is given by $\sigma_K \otimes \beta_{R(K)} = \{\sigma\}$. Moreover, if we consider that by Prop. 6.8.7 $Ks(proj_i(\sigma_K)) \subseteq \{K_i\}$, we can refine the latter to $\sigma \in$ *BehavOnly*$(\mathcal{B}, K)$. But this implies $\sigma \uparrow K_i \in \mathcal{B}^o_K$, and together with Prop. 6.8.15(1) we obtain $\sigma_K \in \mathcal{B}^o_K$ as required.

To verify (b) let $\sigma_K \in \mathcal{B}^u_K$. By definition of $\mathcal{B}^u_K$, there is $\sigma \in$ *UpToSynch*$(\mathcal{B}, K)$ such that $\sigma \uparrow K_i = \sigma_K$. By (A) we have $\sigma \in \mathcal{B}_K \otimes \mathcal{B}_{R(K)}$, and with Prop. 6.8.15(2) it is immediate that indeed $\sigma_K \in \mathcal{B}_K$.

(2.) This follows by analogous argumentation. □

Then, naturally we infer:

Proposition 6.9.6. *Let $\mathcal{B}$ be a cp bisimulation for β. The following two statements are equivalent:*

1. *$\mathcal{B}$ is K-decomposable.*

2. $\mathcal{B}^x_K \otimes \mathcal{B}^y_{R(K)} =$ *UpToSynch*$(\mathcal{B}, K)$*, where* $x, y \in \{u, o\}$.

Proof. By Prop. 6.9.2 and Prop. 6.9.3 it is immediate that (2) implies (1) for any combination x, $y \in \{u, o\}$. On the other hand, it follows from Prop. 6.9.5 that (1) implies (2). □

(4.) Second Basic Insight. Having observed that from $\mathit{UpToSynch}(\mathcal{B}, K)$ we can extract cp bisimulations for β_K and $\beta_{R(K)}$, we now show that conversely the following holds: a cp bisimulation for β_K and a cp bisimulation for $\beta_{R(K)}$ together provide complete matching for β up to synchronization of K; formally, they provide a *cp bisimulation for β up to K-synch.*

Definition 6.9.8. $\mathcal{B}_u$ is a *cp bisimulation for β up to K-synch* iff

1. $\mathcal{B}_u \subseteq \mathit{JFS}(\beta)$,
2. $\beta \in \mathcal{B}_u$,
3. $\mathcal{B}_u$ is prefix-closed, and
4. for $i = 1, 2$ the following is satisfied:

   if $\sigma \in \mathcal{B}_u$ and $\mathit{proj}_i(\sigma) \xrightarrow{t_i}$ for some $t_i \in T_i$ with $t_i \notin \mathit{synchT}_i(K_i)$ then there are $t_{\bar{i}} \in T_{\bar{i}}$, $\sigma' \in \mathit{JFS}$ such that $\sigma \xrightarrow{(t_1,t_2)} \sigma'$ and $\sigma' \in \mathcal{B}_u$.

Proposition 6.9.7. *Let $\mathcal{B}_K$ be a cp bisimulation for β_K, and $\mathcal{B}_{R(K)}$ a cp bisimulation for $\beta_{R(K)}$. $\mathcal{B}_K \otimes \mathcal{B}_{R(K)}$ is a cp bisimulation for β up to K-synch.*

Proof. Let $\mathcal{B}_K$ and $\mathcal{B}_{R(K)}$ be given as above, and set $\mathcal{B}_s = \mathcal{B}_K \otimes \mathcal{B}_{R(K)}$. We need to verify that the conditions (1)-(4) of Def. 6.9.8 are satisfied.

Since $\mathcal{B}_K$ and $\mathcal{B}_{R(K)}$ are cp bisimulations for β_K, and $\beta_{R(K)}$ respectively, we have: (i) $\mathcal{B}_K \subseteq \mathit{JFS}(\beta_K)$ & $\mathcal{B}_{R(K)} \subseteq \mathit{JFS}(\beta_{R(K)})$, (ii) $\beta_K \in \mathcal{B}_K$ & $\beta_{R(K)} \in \mathcal{B}_{R(K)}$, and (iii) $\mathcal{B}_K$ and $\mathcal{B}_{R(K)}$ are prefix-closed. Then, (1) follows from (i) and Prop. 6.8.14, (2) is immediate with (ii), and (3) is a consequence of (iii) and Prop. 6.4.14.

To prove (4), assume $\sigma \in \mathcal{B}_s$ such that $(*)$ $\mathit{proj}_i(\sigma) \xrightarrow{t_i}$ for some $t_i \in T_i$ with $t_i \notin \mathit{synchT}_i(K_i)$, $i = 1, 2$. We need to find entities $t_{\bar{i}} \in T_{\bar{i}}$, $\sigma' \in \mathit{JFS}$ satisfying $\sigma \xrightarrow{(t_1,t_2)} \sigma'$, and $\sigma' \in \mathcal{B}_s$. Since $t_i \notin \mathit{synchT}_i(K_i)$ we either have (a) $\mathit{Ks}(t_i) = \{K_i\}$ or (b) $\mathit{Ks}(t_i) \subseteq \mathcal{R}_i$ (Prop. 6.2.5). We first consider case (a). Set $\sigma_K = \sigma \uparrow K_i$. By definition of $\mathcal{B}_s$, (i), and Prop. 6.8.15(2) we have $\sigma_K \in \mathcal{B}_K$. By Prop. 6.8.11 we obtain $\mathit{last}(\sigma) \uparrow K_i = \mathit{last}(\sigma_K)$, and further with Prop. 6.8.6(1), $(*)$, (a), and Prop. 6.2.18(1) we infer $\mathit{proj}_i(\sigma_K) \xrightarrow{t_i}$. Since $\mathcal{B}_K$ is a cp bisimulation, this implies there must be $t_{\bar{i}} \in T_{\bar{i}}$, $\sigma'_K \in \mathit{JFS}$ such that $\sigma_K \xrightarrow{(t_1,t_2)} \sigma'_K$, and $\sigma'_K \in \mathcal{B}_K$. Set $\sigma' = \sigma\,(t_1, t_2)\,(\mathit{last}(\sigma'_K) \cup \mathit{last}(\sigma) \backslash \mathit{last}(\sigma_K))$. It is straightforward to show that $\sigma' \in \mathcal{B}_s$ Having proved (1) it is then clear that $\sigma' \in \mathit{JFS}$, and further that $\sigma \xrightarrow{(t_1,t_2)} \sigma'$. But altogether, this means $t_{\bar{i}}$ and σ' provide entities as required. Case (b) can be proved in the analogous way. $\square$

With Prop. 6.9.2 and 6.9.3 it follows:

Corollary 6.9.1. *Let $\mathcal{B}$ be a cp bisimulation for β, and choose $x, y \in \{o, u\}$. $\mathcal{B}^x_K \otimes \mathcal{B}^y_{R(K)}$ is a cp bisimulation for β up to K-synch.*

(5.) Third Basic Insight. With Prop. 6.8.16, liveness and Prop. 6.4.12 it is clear: if $\beta \in \sim_{cp}$ then β_K either corresponds to a joint switch place or to a joint synch place. In the latter case K_1 and K_2 will not be capable of performing any independent behaviour at β; we then naturally have:

Proposition 6.9.8. *If β_K corresponds to a joint synch place then every cp bisimulation for β is K-decomposable.*

Proof. Let $\mathcal{B}$ be a cp bisimulation for β, and assume β_K corresponds to a joint synch place. It is easy to see that: (1) $\mathit{UpToSynch}(\mathcal{B}, K) = \mathit{BehavOnly}(\mathcal{B}, \mathcal{R})$; (2) $\mathcal{B}^o_K = \{\beta_K\}$; and (3) $\mathit{BehavOnly}(\mathcal{B}, \mathcal{R}) = \mathcal{B}^o_{R(K)} \otimes \{\beta_K\}$ (this holds is general). From (2) and (3) we easily obtain $\mathcal{B}^o_K \otimes \mathcal{B}^o_{R(K)} = \mathit{BehavOnly}(\mathcal{B}, \mathcal{R})$, and further, considering (1), $\mathit{UpToSynch}(\mathcal{B}, K) = \mathcal{B}^o_K \otimes \mathcal{B}^o_{R(K)}$. But by Prop. 6.9.6 this means $\mathcal{B}$ is indeed K-decomposable. □

If β_K corresponds to a joint switch place then it is not obvious whether there exists a K-decomposable cp bisimulation for β whenever $\beta \in \sim_{cp}$. The only immediate observation we can make for this case is that in general cp bisimulations that are not K-decomposable do exist. Thus, to prove K-decomposability we will require deeper insights.

(6.) Key Idea. Let us summarize what we have achieved so far:

1. By Prop. 6.9.8 we can restrict our attention to the case when β_K is a joint switch place.

2. By Corollary 6.9.1 we know: it is always possible to bisimulate the behaviour of β up to synchronization of K in a 'composite' way; we can simply employ $\mathcal{B}^x_K \otimes \mathcal{B}^y_{R(K)}$, x, $y \in \{u, o\}$. The difficulty lies in whether we can extend such a partial bisimulation to one that covers the full behaviour of β.

Inspired by this, we adopt the following approach: we fix β_K to be of type switch, assume a cp bisimulation $\mathcal{B}$ for β, and set $\mathcal{B}_s = \mathcal{B}^u_K \otimes \mathcal{B}^o_{R(K)}$. (The choice of $x = u$, and $y = o$ is crucial; it will be discussed in Section 6.11.) Our aim is to show:

Subgoal 6.9.1 (Crucial Subgoal^K). *$\mathcal{B}_s$ can be extended to a full cp bisimulation for β.*

Clearly, if we achieve this goal, we can conclude that cp bisimilarity is indeed K-decomposable. In the following, we resolve the Crucial Subgoal^K by exhibiting a chain of sufficient subgoals. We will be led to the concept of *swfsi's*, and altogether K-decomposability will be reduced to our final subgoal 'the matching of swfsi's in $\sim_{cp}$ is deterministic'.

To achieve the Crucial Subgoal^K we need to extend $\mathcal{B}_s$ by matches for K_i-transitions of type synch and the behaviour beyond such synchronizations. Thus, our focus must be on *critical jfs'*:

Definition 6.9.9. We consider a jfs $\sigma \in \mathcal{B}_s$ to be *critical* iff we have $proj_i(\sigma) \xrightarrow{t_i}$ for some $t_i \in synchT(K_i)$, $i = 1$, or 2. We write $critical(\mathcal{B}_s)$ to denote the set of critical jfs' of $\mathcal{B}_s$.

It is now easy to formulate a condition that is sufficient to prove the Crucial Subgoal^K: if we achieve '*Subgoal 1:* $\forall \sigma \in critical(\mathcal{B}_s).\ \exists \sigma' \in \sim_{cp}.\ last(\sigma) = last(\sigma')$', then it will be possible to copy the required matches from a cp bisimulation that contains the respective tuples σ'.

Assume $\sigma \in critical(\mathcal{B}_s)$, and set $last(\sigma) \uparrow K_i = \{(p^1_{si}, p^2_{si})\}$. By definition of $\mathcal{B}_s$ and Prop. 6.8.15(2) we obtain (A) $\sigma \uparrow K_i \in \mathcal{B}^u_K$, and (B) $\sigma \uparrow \mathcal{R}_i \in \mathcal{B}^o_{R(K)}$. From (B) and the definition of $\mathcal{B}^o_{R(K)}$ we further infer: there must be $\sigma^\circ \in \mathcal{B}$ such that (a) $Ks(proj_i(\sigma^\circ)) \subseteq \mathcal{R}_i$, and (b) $\sigma^\circ \uparrow \mathcal{R}_i = \sigma \uparrow \mathcal{R}_i$. (b) and Prop. 6.8.11 entail (c) $last(\sigma^\circ) \uparrow \mathcal{R}_i = last(\sigma) \uparrow \mathcal{R}_i$. Exploiting liveness it will be straightforward to show that we can extend σ° to a jfs $\sigma^e \equiv \sigma^\circ.\sigma^\diamond \in \sim_{cp}$ such that: (d) $Ks(proj_i(\sigma^\diamond)) = \{K_i\}$, and (e) $proj_i(last(\sigma^e) \uparrow K_i) = \{p^i_{si}\}$. Importantly, by (d), Prop. 6.8.9(2), and (c) we obtain (f) $last(\sigma^e) \uparrow \mathcal{R}_i = last(\sigma) \uparrow \mathcal{R}_i$. Altogether, this means we have found $\sigma^e \in \sim_{cp}$ such that $last(\sigma^e)$ is *almost* identical with $last(\sigma)$: by (e) and (f), $last(\sigma^e)$ and $last(\sigma)$ can at most differ in their match of p^i_{si}. Thus, if we achieve '*Subgoal 2: in general,* $last(\sigma^e)(p^i_{si}) = p^{\bar{i}}_{si}$', we can conclude $last(\sigma) = last(\sigma^e)$, which will immediately imply that Subgoal 1 is satisfied.

On the other hand, from (A) and the definition of $\mathcal{B}^u_K$ we obtain: there must be $\sigma^\star \in \mathcal{B}$ such that (i) $nosynch(K_i, proj_i(\sigma^\star))$, and (ii) $\sigma^\star \uparrow K_i = \sigma \uparrow K_i$. (ii) and Prop. 6.8.11 imply $last(\sigma^\star) \uparrow K_i = \{(p^1_{si}, p^2_{si})\}$. In view of Subgoal 2 this means: $p^{\bar{i}}_{si}$ has successfully been employed as a match for p^i_{si} in the jfs $\sigma^\star$, which is contained in $\sim_{cp}$. The difficulty is that a priori we do not know whether $p^{\bar{i}}_{si}$ provides a valid '$\sim_{cp}$-match' for p^i_{si} in the context of $last(\sigma) \uparrow \mathcal{R}_i$ just as it does in the context of $last(\sigma^\star) \uparrow \mathcal{R}_i$. The hope then is: maybe we can show that the matching in cp bisimilarity is deterministic in a certain sense that would allow us to conclude: if $p_{si}^{\bar{i}}$ acts as the match of p_{si}^{i} in $\sigma^\star$ then $p_{si}^{\bar{i}}$ must also be the match of p_{si}^{i} in σ^e. Accordingly, we set: *'Subgoal 3: in general,* $last(\sigma^\star)(p_{si}^{i}) = last(\sigma^e)(p_{si}^{i})$*'*.

A close look at σ^e and $\sigma^\star$ tells us that in both jfs' p_{si}^{i} occurs in a very special way: (1) it is easy to derive that p_{si}^{i} is a synch place of K_i; (2) by (i) K_i does not synchronize on $\sigma^\star$, and considering (a) and (d) this also follows for σ^e; (3) by our assumption 'β_K is of type switch' we know that K_i starts off at a switch place in both, $\sigma^\star$ and σ^e. Together, (1) and (2) mean p_{si}^{i} occurs as *first synchronization interface* (short: *fsi*) of K_i w.r.t. $proj_i(\sigma^\star)$ and $proj_i(\sigma^e)$. Integrating (3), we further say: p_{si}^{i} occurs as *switch fsi* (short: *swfsi*) of K_i w.r.t. $proj_i(\sigma^\star)$ and $proj_i(\sigma^e)$. Now, it is clear: Subgoal 3 would immediately follow, if we knew that the matching of swfsi's in $\sim_{cp}$ was deterministic in the following sense: given a place $p_i \in P_i$ and a jfs $\sigma \in \sim_{cp\text{-}\beta}$, whenever p_i occurs as swfsi w.r.t. $proj_i(\sigma)$, it is matched to one particular place $p_{\bar{i}} \in P_{\bar{i}}$ in σ, where $p_{\bar{i}}$ only depends on p_i and β. Thus, as our last subgoal we put forward: *Subgoal 4: the matching of swfsi's in* $\sim_{cp}$ *is deterministic.*

Note that this last subgoal is far from obvious: it requires us to prove a strong statement about the interior of cp bisimulations. In the following section, we will achieve it with the help of the WNL Theorem.

6.10 swfsi Matching in cp Bisimilarity is Deterministic

Motivated by the previous section, we now want to show that for live SSMD fc systems the matching of swfsi's in cp bisimulations is deterministic. Let $\beta \in$ JProc, and $p_i \in P_i$. Concretely, we need to prove: for any two jfs' σ, $\sigma' \in \sim_{cp\text{-}\beta}$ that are *swfsi-adequate w.r.t.* p_i in that p_i occurs as swfsi w.r.t. $\mathit{proj}_i(\sigma)$, and $\mathit{proj}_i(\sigma')$ respectively, we have $\mathit{last}(\sigma)(p_i) = \mathit{last}(\sigma')(p_i)$. We will indeed achieve this result, and in fact we will prove it as a consequence of the WNL Theorem.

The argument goes as follows. In the beginning we observe that any jfs $\sigma \in \sim_{cp\text{-}\beta}$ that is swfsi-adequate w.r.t. p_i can be extended to a jfs σ' that is *'active swfsi'-adequate* (short: *aswfsi-adequate*) w.r.t. p_i, and crucially, p_i and its match will remain unaffected by the extension. The key insight then is: from any two given σ, $\sigma' \in \mathit{JFS}(\beta)$ that are aswfsi-adequate w.r.t. p_i we can infer specific topological entities for $N_{\bar{i}}$, namely a wedge and an indirect link; this will be possible via our observations of Section 6.8(3). Further, if $\mathit{last}(\sigma)(p_i) \neq \mathit{last}(\sigma')(p_i)$ then the wedge will be proper, and altogether the entities will be contradictory to the WNL Theorem. Thus, we conclude that aswfsi matching in JFS is deterministic, and carry over that swfsi matching in $\sim_{cp}$ must be deterministic as well.

We will organize and refine this material as follows. In the first part, we shall provide the necessary definitions and some straightforward insights. In particular, we shall see that we can restrict our attention to places that are *topological swfsi's* (short: *t-swfsi's*); swfsi matching in $\sim_{cp}$ can then be more conveniently studied as *fsi matching of t-swfsi's* in $\sim_{cp}$, and aswfsi matching in JFS as *afsi matching of t-swfsi's* in JFS. In the first part, we shall also establish the analogue of our initial observation on the connection between swfsi- and aswfsi-adequate jfs'. In the second part, we then prove: given $\beta \in \mathit{JProc}$ and a t-swfsi p_i of M^i, there exists at most one *afsi-match* for p_i in $\mathit{JFS}(\beta)$. We will call this result the SWFSI Matching Lemma. With its help it will then be immediate to derive our main result, the SWFSI Matching Theorem: given $\beta \in \sim_{cp}$ and a t-swfsi p_i of M^i, there exists exactly one *fsi-match* for p_i in $\sim_{cp\text{-}\beta}$. This makes the third part. In the final part, we will see that with the help of the SWFSI Matching Theorem we can indeed make predictions about the interior of cp bisimulations as required by Section 6.9(6). This will give rise to two SWFSI Prediction Theorems.

For the beginning of the first part we fix a live SSMD fc system $(\mathcal{N}, \mathit{Cover})$, and assume the setting of Def. 6.2.19. Otherwise (and as already done in this introduction), we work in the context of two live SSMD fc systems $(\mathcal{N}_1, \mathit{Cover}_1)$, $(\mathcal{N}_2, \mathit{Cover}_2)$, and build on the framework of Section 6.8. We shall use the following additional convention: in the context of $\beta \in \mathit{JProc}$, we set $M_i = \mathit{proj}_i(\beta)$.

6.10.1 Definitions and First Insights

swfsi's and t-swfsi's. We start with the formal definition of swfsi occurrences:

Definition 6.10.1. Let $M \in \mathit{Proc}$, $K \in \mathit{Cover}$, $r \in \mathit{Runs}(M)$, $p \in P$, and M' such that $M[r\rangle M'$. We say p *occurs as fsi of* K *w.r.t.* r (short: p *is* $\mathit{fsi}(K, r)$) iff

1. p is of type synch,
2. $M'(K) = p$, and
3. $\mathit{nosynch}(K, r)$.

We say p *occurs as switch fsi of* K *w.r.t.* r (short: p *is* $\mathit{swfsi}(K, r)$) if we further have

4. $M(K)$ is of type switch.

A place which can occur as swfsi relative to a given process M satisfies the following topological criterion:

Proposition 6.10.1. *Let $M \in \mathit{Proc}$, $p \in P$. If p is $\mathit{swfsi}(K, r)$ of some $K \in \mathit{Cover}$ w.r.t. some $r \in \mathit{Runs}(M)$ then there exists a TFS-path in N leading from $M(K(p))$ to p.*

Proof. Let $M \in \mathit{Proc}$, $p \in P$, $K \in \mathit{Cover}$, $r \in \mathit{Runs}(M)$, and set M' such that $M[r\rangle M'$. Assume p is $\mathit{swfsi}(K, r)$, and thereby that the conditions (1)-(4) of Def. 6.10.1 are satisfied. We need to exhibit a PP-path π such that: (a) $\mathit{nosynch}(\mathit{ts}(\pi))$, (b) $\mathit{first}(\pi) = M(K(p))$ is of type switch, and (c) $\mathit{last}(\pi) = p$ is of type synch.

By Prop. 6.2.8 we have $M(K)[r\!\uparrow\! K\rangle_K M'(K)$, and hence with Prop. 6.2.16(2) we obtain a PP-path π that leads from $M(K)$ to $M'(K)$ and satisfies $(*)$ $\mathit{ts}(\pi) = r\!\uparrow\! K$. Clearly, π is a PP-path as required: (a) is immediate with $(*)$ and (3); by Prop. 6.2.12(2) and condition (2) we have $K(p) = K$, and then (b) follows from (4); (c) is a consequence of (1) and (2). $\square$

Accordingly, we define the concept of *topological swfsi's*:

Definition 6.10.2. Let $M \in \mathit{Proc}$, and $p_{si} \in P$. We say p_{si} is a *topological swfsi* (short: *t-swfsi*) *of* M iff there exists a TFS-path in N leading from $M(K(p_{si}))$ to p_{si}. We denote the set of t-swfsi's of M by $\mathit{T\text{-}SWFSIs}(M)$.

We equip t-swfsi's with the following attributes:

Convention 6.10.1. Let $M \in Proc$. In the context of $p_{si} \in T\text{-}SWFSIs(M)$, we set $K_{si} = K(p_{si})$, and t_{si} such that ${p_{si}}^\bullet = \{t_{si}\}$. Given $M_i \in Proc_i$, we carry this convention over to $p^i_{si} \in T\text{-}SWFSIs(M_i)$ in the obvious way.

The runs w.r.t. which t-swfsi's will occur as fsi's can be captured as follows:

Definition 6.10.3. Let $M \in Proc$, $p_{si} \in T\text{-}SWFSIs(M)$, $r \in Runs(M)$, and M' such that $M[r\rangle M'$. We say r is *fsi-adequate w.r.t.* p_{si} iff $p_{si} \in M'$ & $nosynch(K_{si}, r)$. We denote the set of runs of M that are fsi-adequate w.r.t. p_{si} by $fsi\text{-}Runs(M, p_{si})$.

Naturally, whenever a t-swfsi is $fsi(K, r)$ for some K and r, it is also $swfsi(K, r)$. Altogether, we then have:

Proposition 6.10.2. *Let $M \in Proc$, $K \in Cover$, $r \in Runs(M)$, and $p_{si} \in P$.*

$$p_{si} \text{ is } swfsi(K, r) \iff p_{si} \in T\text{-}SWFSIs(M) \ \& \ K_{si} = K \ \& \ r \in fsi\text{-}Runs(M, p_{si}).$$

Proof. Consider the definitions of $swfsi$, $T\text{-}SWFSIs$, and $fsi\text{-}Runs$. The '$\Rightarrow$'-direction follows with Prop. 6.10.1 and Prop. 6.2.12(2). The '$\Leftarrow$'-direction is immediate with Prop. 6.2.12(2), and the definition of TFS-paths. $\square$

Example 6.10.1. Consider the system of Figure 6.5 and let M be the marking indicated in the figure. $p_4t_5p_5t_7p_7t_8p_8$ is a TFS-path leading from $M(K_2) = p_4$ to p_8, where $K(p_8) = K_2$. Thus, we have $p_8 \in T\text{-}SWFSIs(M)$. It is easy to see that $r \equiv t_5t_3t_7t_8 \in fsi\text{-}Runs(M, p_8)$. Independently, one can check that p_8 occurs as swfsi of K_2 w.r.t. r. Finally note that p_8 is *not* swfsi of K_2 w.r.t. the run $t_5t_6t_3t_4$.

The following proposition shows that each t-swfsi can indeed occur as swfsi. Note that together with Prop. 6.10.2 this means: t-swfsi's exactly capture the set of 'potential swfsi's'.

Proposition 6.10.3. *Let $M \in Proc$, and $p_{si} \in T\text{-}SWFSIs(M)$.*

$$|fsi\text{-}Runs(M, p_{si})| \geq 1.$$

Proof. Let M, and p_{si} be given as above. By definition there must be a TFS-path π leading from $M(K_{si})$ to p_{si}. Clearly, we have $ts(\pi) \in fsi\text{-}Runs(M, p_{si})$: by Prop. 6.7.2 $ts(\pi)$ is a valid run of M, which will enable p_{si}; with the definition of TFS-paths it is immediate that $nosynch(K_{si}, r)$. $\square$

Slightly stronger we also have:

Proposition 6.10.4. *Let $M \in Proc$, $p_{si} \in T\text{-}SWFSIs(M)$, and $r \in Runs(M)$ such that $K_{si} \notin Ks(r)$. Then there exists a run r' such that*

1. *$r.r' \in fsi\text{-}Runs(M, p_{si})$, and*

2. *$Ks(r') = \{K_{si}\}$.*

Proof. Let M, p_{si}, and r be given as above, and set M' such that $M[r\rangle M'$. We refer to our assumption $K_{si} \notin Ks(r)$ by (A).

We can assume a TFS-path π leading from $M(K_{si})$ to p_{si}. Again, $ts(\pi)$ provides r' as required: with Prop. 6.7.1(1) we infer $Ks(\pi) = \{K_{si}\}$, which implies (2.). To see that (1.) holds, consider: by Prop. 6.2.9(2) and (A) we obtain $M(K_{si}) = M'(K_{si})$, and so similarly to above $ts(\pi)$ is a valid run of M', which will enable p_{si}; with (A) and (2.) it is clear that $nosynch(K_{si}, r.r')$. $\square$

swfsi Matching in $\sim_{cp}$. Justified by Prop. 6.10.2 we shall restrict our attention to t-swfsi's, and study swfsi matching in $\sim_{cp}$ more conveniently as *fsi matching of t-swfsi's* in $\sim_{cp}$. Accordingly, we define:

Definition 6.10.4. Let $\beta \in JProc$, and $p^i_{si} \in T\text{-}SWFSIs(M_i)$.

Let $\sigma \in JFS(\beta)$. We say σ is *fsi-adequate w.r.t.* p^i_{si} iff $proj_i(\sigma) \in fsi\text{-}Runs(M_i, p^i_{si})$. We denote the set of jfs' of β that are fsi-adequate w.r.t. p^i_{si} by $fsi\text{-}JFS(\beta, p^i_{si})$.

We say $p_{\bar{i}} \in P_{\bar{i}}$ is a *fsi-match of* p^i_{si} *in* $\sim_{cp\text{-}\beta}$ iff there is $\sigma \in fsi\text{-}JFS(\beta, p^i_{si}) \cap \sim_{cp\text{-}\beta}$ such that $p_{\bar{i}} = last(\sigma)(p^i_{si})$. We denote the set of fsi-matches of p^i_{si} in $\sim_{cp\text{-}\beta}$ by $cp\text{·}fsi\text{-}Matches(\beta, p^i_{si})$.

With Prop. 6.10.3 it is immediate: given $\beta \in \sim_{cp}$, there exists at least one fsi-match in $\sim_{cp\text{-}\beta}$ for each t-swfsi of M_i:

Proposition 6.10.5. *Let $\beta \in \sim_{cp}$, and $p^i_{si} \in T\text{-}SWFSIs(M_i)$.*

1. *$|fsi\text{-}JFS(\beta, p^i_{si}) \cap \sim_{cp\text{-}\beta}| \geq 1$, and thus*

2. *$|cp\text{·}fsi\text{-}Matches(\beta, p^i_{si})| \geq 1$.*

Proof. Let β, and p^i_{si} be given as above.

(1.) By Prop. 6.10.3 we can assume $r_i \in fsi\text{-}Runs(M_i, p^i_{si})$. Clearly, $\sim_{cp\text{-}\beta}$ must contain a match for r_i, that is there must be $\sigma \in \sim_{cp\text{-}\beta}$ such that $proj_i(\sigma) = r_i$. But altogether this gives us $\sigma \in fsi\text{-}JFS(\beta, p^i_{si}) \cap \sim_{cp\text{-}\beta}$ as required.

(2.) is immediate from (1.). $\square$

With Prop. 6.10.4 we obtain a correspondingly stronger statement:

Proposition 6.10.6. *Let $\beta \in JProc$, $\mathcal{B}$ be a cp bisimulation for β, further $p^i_{si} \in T\text{-}SWFSIs(M_i)$, and $\sigma \in \mathcal{B}$ such that $K^i_{si} \notin Ks(proj_i(\sigma))$. Then there exists a jfs σ' such that*

1. *$\sigma.\sigma' \in fsi\text{-}JFS(\beta, p^i_{si}) \cap \mathcal{B}$, and*
2. *$last(\sigma.\sigma') \uparrow \mathcal{R}_i = last(\sigma) \uparrow \mathcal{R}_i$, where $\mathcal{R}_i = Cover_i \backslash K^i_{si}$.*

Proof. Let β, $\mathcal{B}$, p^i_{si}, and σ be given as above. By Prop. 6.10.4 we obtain a run r'_i such that (a) $proj_i(\sigma).r'_i \in fsi\text{-}Runs(M_i, p^i_{si})$, and (b) $Ks(r'_i) = \{K^i_{si}\}$. With similar reasoning as used for Prop. 6.10.5 we can infer a jfs σ' such that (i) $proj_i(\sigma') = r'_i$, and (ii) $\sigma.\sigma' \in fsi\text{-}JFS(\beta, p^i_{si}) \cap \mathcal{B}$. (ii) means σ' satisfies condition (1). With (b), (i), and Prop. 6.8.9(2) it is clear that σ' also meets condition (2). □

Note how Prop. 6.10.6 relates to our sketch of Section 6.9(6): it confirms that σ° can be extended to $\sigma^\circ.\sigma^\diamond$ as required.

aswfsi Matching in *JFS*. As explained in the beginning, we will gain our insight on swfsi matching in $\sim_{cp}$ via a result about 'active swfsi' (short: aswfsi) matching in *JFS*. We say a place p occurs as aswfsi w.r.t. a run r, iff p occurs as swfsi w.r.t. r, and the synch transition of p is enabled at r. Importantly, we consider a jfs σ to be aswfsi-adequate w.r.t. a place p_i, only if σ is swfsi-adequate w.r.t. p_i, and the synch transition t_i of p_i is enabled in the *joint context of* σ, that is $\sigma[t\rangle$ for some $t \in T$ satisfying $proj_i(t) = t_i$. Taking into account that we can study aswfsi matching in *JFS* as *'active fsi'* (short: *afsi*) *matching of t-swfsi's* in *JFS* we accordingly define:

Definition 6.10.5. Let $\beta \in JProc$, and $p^i_{si} \in T\text{-}SWFSIs(M_i)$.

Let $\sigma \in JFS(\beta)$. We say σ is *afsi-adequate w.r.t.* p^i_{si} iff $\sigma \in fsi\text{-}JFS(\beta, p^i_{si})$, and $\sigma[t\rangle$ for some $t \in T$ such that $proj_i(t) = t^i_{si}$. We denote the set of jfs' of β that are afsi-adequate w.r.t. p^i_{si} by $afsi\text{-}JFS(\beta, p^i_{si})$.

We say $p_{\bar{i}} \in P_{\bar{i}}$ is an *afsi-match of p^i_{si} in $JFS(\beta)$* iff there is $\sigma \in afsi\text{-}JFS(\beta, p^i_{si})$ such that $p_{\bar{i}} = last(\sigma)(p^i_{si})$. We denote the set of afsi-matches of p^i_{si} in $JFS(\beta)$ by $afsi\text{-}Matches(\beta, p^i_{si})$.

We also define:

Definition 6.10.6. Let $M \in Proc$, $p_{si} \in T\text{-}SWFSIs(M)$, $r \in Runs(M)$, and M' such that $M[r\rangle M'$. We say r is *afsi-adequate w.r.t.* p_{si} iff $r \in fsi\text{-}Runs(M, p_{si})$ and $M'[t_{si}\rangle$. We denote the set of runs of M that are afsi-adequate w.r.t. p_{si} by $afsi\text{-}Runs(M, p_{si})$.

Liveness ensures that every swfsi occurrence can become active later on; formally we write:

Proposition 6.10.7. *Let $M \in Proc$, and $p_{si} \in T\text{-}SWFSIs(M)$. For all $r \in fsi\text{-}Runs(M, p_{si})$ there is a run r' such that*

1. *$r.r' \in afsi\text{-}Runs(M, p_{si})$, and*
2. *$K_{si} \notin Ks(r')$.*

Proof. Let M, and p_{si} be given as above, and assume $r \in fsi\text{-}Runs(M, p_{si})$. By Prop. 6.2.13 there must be a run r' such that (a) $r.r'[t_{si}\rangle$, and (b) $K_{si} \notin Ks(r')$. Clearly, this implies $r.r' \in afsi\text{-}Runs(M, p_{si})$, and thus we have found a run as required. □

Thus, whenever a swfsi occurrence is matched in a cp bisimulation one has to anticipate the case when it becomes an aswfsi. Accordingly, we have: any fsi-match in $\sim_{cp}$ is also an afsi-match.

Proposition 6.10.8. *Let $\beta \in JProc$, and $p^i_{si} \in T\text{-}SWFSIs(M_i)$.*

1. *For all $\sigma \in fsi\text{-}JFS(\beta, p^i_{si}) \cap \sim_{cp\text{-}\beta}$ there exists a jfs σ' such that*
   (a) $\sigma.\sigma' \in afsi\text{-}JFS(\beta, p^i_{si})$, and
   (b) $last(\sigma.\sigma')(p^i_{si}) = last(\sigma)(p^i_{si})$,

   and thus:
2. *$cp\text{-}fsi\text{-}Matches(\beta, p^i_{si}) \subseteq afsi\text{-}Matches(\beta, p^i_{si})$.*

Proof. Let β, and p^i_{si} be given as above.

(1.) Assume $\sigma \in fsi\text{-}JFS(\beta, p^i_{si}) \cap \sim_{cp\text{-}\beta}$. Set $r_i = proj_i(\sigma)$; by definition, we have $r_i \in fsi\text{-}Runs(M_i, p^i_{si})$. Then, by Prop. 6.10.7 there is r'_i such that (i) $r_i.r'_i \in afsi\text{-}Runs(M_i, p^i_{si})$, and (ii) $K^i_{si} \notin Ks(r'_i)$. Clearly, $\sim_{cp\text{-}\beta}$ will contain a match for r'_i at σ, that is there must be a jfs σ' such that $\sigma.\sigma' \in \sim_{cp\text{-}\beta}$, and $(*)$ $proj_i(\sigma') = r'_i$. Further, since by (i) we have $r_i.r'_i[t^i_{si}\rangle$, $\sim_{cp}$ will provide a match for t^i_{si} at $\sigma.\sigma'$; this implies there must be $t \in T$ such that $\sigma.\sigma'[t\rangle$, and $proj_i(t) = t^i_{si}$. Considering that $proj_i(\sigma.\sigma') = r_i.r'_i$, with (i) it is then clear that $\sigma.\sigma' \in afsi\text{-}JFS(\beta, p^i_{si})$, and thus σ' provides a jfs satisfying condition (a). To see that σ' also satisfies (b) consider: with (ii), $(*)$ and Prop. 6.8.9(2) we obtain $last(\sigma.\sigma') \uparrow K^i_{si} = last(\sigma) \uparrow K^i_{si}$, and by Prop. 6.8.6(5) this clearly implies $last(\sigma.\sigma')(p^i_{si}) = last(\sigma)(p^i_{si})$.

(2.) is immediate from (1.). □

6.10.2 The SWFSI Matching Lemma

We now prove: *for any $\beta \in JProc$ and $p^i_{si} \in T\text{-}SWFSIs(M_i)$, there exists at most one afsi-match for p^i_{si} in $JFS(\beta)$.* This is the statement of the SWFSI Matching Lemma. In the following we demonstrate how it follows as a consequence of the WNL Theorem.

Using the findings of Section 6.8(3) we first observe that from a given afsi-adequate jfs we can extract well-known topological entities for the opposite system, namely a TFS-path and a family of simple links. Due to Prop. 6.4.13, the entities are such that the final component of each link is a synchronization partner of the TFS path's synchronization interface.

Convention 6.10.2. Let $\beta \in JProc$, $p^i_{si} \in T\text{-}SWFSIs(M^i)$, $\sigma \in afsi\text{-}JFS(\beta, p^i_{si})$, and set $\beta' = last(\sigma)$. In the context of σ we assume:

- $p^{\bar{i}}_{si} = proj_{\bar{i}}(\beta' \!\uparrow\! K^i_{si})$, or $\beta'(p^i_{si})$ equivalently (considering Prop. 6.8.6(2) and Prop. 6.2.12(2)).
- $p^{\bar{i}}_{to\text{-}si} = proj_{\bar{i}}(\beta \!\uparrow\! K^i_{si})$, and
- $K^{\bar{i}}_{si} = cm\!\cdot\!\beta(K^i_{si})$, or $K(p^{\bar{i}}_{si})$ equivalently (considering Prop. 6.8.6(3)).

Given $p^i_{sp} \in SPartners(p^i_{si})$ we further assume:

- $K^i_{sp} = K(p^i_{sp})$,
- $p^{\bar{i}}_{to\text{-}sp} = proj_{\bar{i}}(\beta \!\uparrow\! K^i_{sp})$,
- $K^{\bar{i}}_{sp} = cm\!\cdot\!\beta'(K^i_{sp})$, and
- $K^{\bar{i}}_{to\text{-}sp} = cm\!\cdot\!\beta(K^i_{sp})$.

For entities $a\text{-}\sigma$, $b\text{-}\sigma \in afsi\text{-}JFS(\beta, p^i_{si})$ we assume $a\text{-}p^{\bar{i}}_{si}$, $b\text{-}p^{\bar{i}}_{si}$, etc. in an analogous way.

Lemma 6.10.1 (topological entities). *Let $\beta \in JProc$, $p^i_{si} \in T\text{-}SWFSIs(M^i)$. For all $\sigma \in afsi\text{-}JFS(\beta, p^i_{si})$ we have:*

1. *There is a TFS-path $\pi_{\bar{i}}$ in $N_{\bar{i}}$ leading from $p^{\bar{i}}_{to\text{-}si}$ to $p^{\bar{i}}_{si}$,*

   namely $\pi_{\bar{i}} = path_{\bar{i}}(\sigma \!\uparrow\! K^i_{si})$.

2. *For all $p^i_{sp} \in SPartners(p^i_{si})$ there is a simple link $\lambda_{\bar{i}}$ in $N_{\bar{i}}$ such that*

   (a) $p^{in}_{\lambda_{\bar{i}}} = p^{\bar{i}}_{to\text{-}sp}$,

(b) $K^{in}_{\lambda_{\bar{i}}} = K^{\bar{i}}_{to\text{-}sp}$ & $K^{fi}_{\lambda_{\bar{i}}} = K^{\bar{i}}_{sp}$, *and*

(c) $K^{\bar{i}}_{si} \notin \mathcal{K}_{\lambda_{\bar{i}}}$,

namely $\lambda_{\bar{i}} = link_{\bar{i}}(\sigma \uparrow K^{i}_{sp})$.

3. *For all* $p^{i}_{sp} \in SPartners(p^{i}_{si})$ *we have* $K^{\bar{i}}_{sp} \in KSPartners(p^{\bar{i}}_{si})$.

Proof. Let β, p^{i}_{si}, and σ be given as above. Since $\sigma \in afsi\text{-}JFS(\beta, p^{i}_{si})$ we have: (i) $p^{i}_{si} \in proj_i(last(\sigma))$, (ii) $nosynch(K^{i}_{si}, proj_i(\sigma))$, and (iii) $last(\sigma) \xrightarrow{t}$ for $t \in T$ such that $proj_i(t) = t^{i}_{si}$.

(1.) follows from Prop. 6.8.18(1), and Prop. 6.8.21; to see that the latter applies consider (ii), the definition of t-swfsi's, (i) with Prop. 6.2.12(2), and (iii).

(2.) In turn, this follows as a consequence of Prop. 6.8.18(2) and Prop. 6.8.20(2); the latter applies by (ii), and since clearly $K^{i}_{sp} \neq K^{i}_{si}$ (Prop. 6.2.2(1a)).

(3.) is immediate with (i),(iii) and Prop. 6.4.13 when employing Prop. 6.8.2. □

Note that a TFS-path makes up half of a wedge, and a simple link makes up half of an indirect link. In particular, given two jfs' $a\text{-}\sigma$, $b\text{-}\sigma \in afsi\text{-}JFS(\beta, p^{i}_{si})$ for fixed β and p^{i}_{si}, we can combine their associated topological entities to obtain a wedge W, and a family of indirect links $\{\lambda\}$ such that each λ links opposite synchronization components of W without passing through K_W. The SWFSI Matching Lemma then follows with the following argument: if $a\text{-}\sigma$ and $b\text{-}\sigma$ gave rise to different afsi-matches then W would be a proper wedge, and together W and each λ would be such as forbidden by the WNL Theorem.

Lemma 6.10.2 (SWFSI Matching Lemma). *Assume* $\beta \in JProc$, *and* $p^{i}_{si} \in T\text{-}SWFSIs(M_i)$. *We have:*

$$|afsi\text{-}Matches(\beta, p^{i}_{si})| \leq 1.$$

Proof. Let $\beta \in JProc$, and $p^{i}_{si} \in T\text{-}SWFSIs(M_i)$. To the contrary suppose there are $a\text{-}p^{\bar{i}}_{si}$, $b\text{-}p^{\bar{i}}_{si} \in afsi\text{-}Matches(\beta, p^{i}_{si})$ with $a\text{-}p^{\bar{i}}_{si} \neq b\text{-}p^{\bar{i}}_{si}$; we will show that this assumption leads to a contradiction.

By definition, we can assume $a\text{-}\sigma$, $b\text{-}\sigma \in afsi\text{-}JFS(\beta, p^{i}_{si})$ such that $a\text{-}p^{\bar{i}}_{si} = a\text{-}\beta'(p^{i}_{si})$, and $b\text{-}p^{\bar{i}}_{si} = b\text{-}\beta'(p^{i}_{si})$, where $a\text{-}\beta' = last(a\text{-}\sigma)$, $b\text{-}\beta' = last(b\text{-}\sigma)$.

Then by Lemma 6.10.1(1), in $N_{\bar{i}}$ there is a TFS-path $a\text{-}\pi_{\bar{i}}$ leading from $a\text{-}p^{\bar{i}}_{to\text{-}si}$ to $a\text{-}p^{\bar{i}}_{si}$, and a TFS-path $b\text{-}\pi_{\bar{i}}$ leading from $b\text{-}p^{\bar{i}}_{to\text{-}si}$ to $b\text{-}p^{\bar{i}}_{si}$ respectively. Since clearly $a\text{-}p^{\bar{i}}_{to\text{-}si} = b\text{-}p^{\bar{i}}_{to\text{-}si}$, these two TFS-paths form a wedge, namely $W_{\bar{i}} = (a\text{-}\pi_{\bar{i}}, b\text{-}\pi_{\bar{i}})$. Crucially, by our assumption $a\text{-}p^{\bar{i}}_{si} \neq b\text{-}p^{\bar{i}}_{si}$, $W_{\bar{i}}$ is a proper wedge.

On the other hand, if we pick some $p^i_{sp} \in SPartners(p^i_{si})$ then by Lemma 6.10.1(2) we obtain a simple link $a\text{-}\lambda_{\bar{i}}$ in $N_{\bar{i}}$ leading from $a\text{-}K^{\bar{i}}_{to\text{-}sp}$ to $a\text{-}K^{\bar{i}}_{sp}$ with $p^{in}_{a\text{-}\lambda_{\bar{i}}} = a\text{-}p^{\bar{i}}_{to\text{-}sp}$, and $K^{\bar{i}}_{si} \notin \mathcal{K}_{a\text{-}\lambda_{\bar{i}}}$, and likewise a simple link $b\text{-}\lambda_{\bar{i}}$ leading from $b\text{-}K^{\bar{i}}_{to\text{-}sp}$ to $b\text{-}K^{\bar{i}}_{sp}$ with $p^{in}_{b\text{-}\lambda_{\bar{i}}} = b\text{-}p^{\bar{i}}_{to\text{-}sp}$, and $K^{\bar{i}}_{si} \notin \mathcal{K}_{b\text{-}\lambda_{\bar{i}}}$. Since clearly $a\text{-}p^{\bar{i}}_{to\text{-}sp} = b\text{-}p^{\bar{i}}_{to\text{-}sp}$, this time we can combine the two entities to an indirect link $\lambda_{\bar{i}} = (a\text{-}\lambda_{\bar{i}}, b\text{-}\lambda_{\bar{i}})$.

By Lemma 6.10.1(3) we have $a\text{-}K^{\bar{i}}_{sp} \in KSPartners(a\text{-}p^{\bar{i}}_{si})(= \mathcal{K}^{sp}_{W_{\bar{i}}\text{-}l})$, and $b\text{-}K^{\bar{i}}_{sp} \in KSPartners(b\text{-}p^{\bar{i}}_{si})(= \mathcal{K}^{sp}_{W_{\bar{i}}\text{-}r})$. But altogether this means we have indeed reached a contradiction to the WNL Theorem (Theorem 6.7.3): $\lambda_{\bar{i}}$ leads from $a\text{-}K^{\bar{i}}_{sp}$ to $b\text{-}K^{\bar{i}}_{sp}$ and satisfies $K^{\bar{i}}_{si}(= K_{W_{\bar{i}}}) \notin \mathcal{K}_{\lambda_{\bar{i}}}$. □

Convention 6.10.3. Let $\beta \in JProc$, and $p^i_{si} \in T\text{-}SWFSIs(M_i)$. If we have $afsi\text{-}Matches(\beta, p^i_{si}) \neq \emptyset$ then we denote the, by Lemma 6.10.2 uniquely given, afsi-match of p^i_{si} in $JFS(\beta)$ by $afsi\text{-}Match(\beta, p^i_{si})$.

6.10.3 The SWFSI Matching Theorem

Having established that afsi matching of t-swfsi's in *JFS* is deterministic, via Prop. 6.10.8(2) we can now conclude that fsi matching of t-swfsi's in $\sim_{cp}$ must be deterministic as well. Integrating Prop. 6.10.5(2) we obtain: *for any* $\beta \in \sim_{cp}$ *and* $p^i_{si} \in T\text{-}SWFSIs(M_i)$, *there exists* exactly one *fsi-match for* p^i_{si} *in* $\sim_{cp\text{-}\beta}$. In full detail, we have:

Theorem 6.10.1 (SWFSI Matching Theorem). *Assume* $\beta \in \sim_{cp}$, *and* $p^i_{si} \in T\text{-}SWFSIs(M_i)$.

1. $afsi\text{-}Match(\beta, p^i_{si})$ *exists, and*

2. $cp\cdot fsi\text{-}Matches(\beta, p^i_{si}) = \{afsi\text{-}Match(\beta, p^i_{si})\}$.

Proof. Considering Lemma 6.10.2 and Conv. 6.10.3 this follows from Prop. 6.10.8(2) and Prop. 6.10.5(2). □

Convention 6.10.4. Let $\beta \in \sim_{cp}$, and $p^i_{si} \in T\text{-}SWFSIs(M_i)$. We denote the, by Theorem 6.10.1 uniquely existing, fsi-match of p^i_{si} in $\sim_{cp\text{-}\beta}$ by $cp\cdot fsi\text{-}Match(\beta, p^i_{si})$.

6.10.4 The SWFSI Prediction Theorems

Naturally, we can employ the SWFSI Matching Theorem to make predictions about the interior of cp bisimulations.

As a direct consequence, we obtain:

Theorem 6.10.2 (SWFSI Prediction Theorem I). *Let $\beta \in JProc$, $\mathcal{B}$ a cp bisimulation for β, $\sigma \in \mathcal{B}$, $K_i \in Cover_i$, and $p^i_{si} \in P_i$.*

$$p^i_{si} \text{ is } swfsi(K_i, proj_i(\sigma)) \implies \\ p^i_{si} \in T\text{-}SWFSIs(M_i) \ \&\ K^i_{si} = K_i \ \&\ last(\sigma)(p^i_{si}) = cp\text{-}fsi\text{-}Match(\beta, p^i_{si}).$$

Proof. Considering Theorem 6.10.1 and Conv. 6.10.4 this follows with Prop. 6.10.2, and the fact that $\mathcal{B} \subseteq \sim_{cp\text{-}\beta}$. □

With Prop. 6.10.6 we can further predict:

Theorem 6.10.3 (SWFSI Prediction Theorem II). *Let $\beta \in JProc$, $\mathcal{B}$ a cp bisimulation for β, $p^i_{si} \in T\text{-}SWFSIs(M^i)$, and $\sigma \in \mathcal{B}$ such that $K^i_{si} \notin Ks(proj_i(\sigma))$. Then there exists a jfs σ' such that*

1. *$\sigma.\sigma' \in \mathcal{B}$,*
2. *$last(\sigma.\sigma') \uparrow K^i_{si} = \{(p_1, p_2)\}$, where $p_i = p^i_{si}$, $p_{\bar{i}} = cp\text{-}fsi\text{-}Match(\beta, p^i_{si})$ for i instantiated as above, and*
3. *$last(\sigma.\sigma') \uparrow \mathcal{R}_i = last(\sigma) \uparrow \mathcal{R}_i$, where $\mathcal{R}_i = Cover_i \backslash K^i_{si}$.*

Proof. Considering Theorem 6.10.1 and Conv. 6.10.4 this follows with Prop. 6.10.6, and the fact that $\mathcal{B} \subseteq \sim_{cp\text{-}\beta}$. □

Note how the two theorems complement each other: together they will make sure that, assuming entities as in Section 6.9(6), for any $\sigma \in critical(\mathcal{B}^u_K \otimes \mathcal{B}^o_{R(K)})$ there exists $\sigma' \in \mathcal{B}$ such that $last(\sigma) = last(\sigma')$.

6.11 cp Bisimilarity is K-decomposable, sw-(1)-coherent, and sw-(1)hereditary

Having established that swfsi matching in $\sim_{cp}$ is deterministic, we are now ready to present the two main results of Part II: cp bisimilarity on live SSMD fc systems is K-decomposable, and sw-(1)coherent and sw-(1)hereditary. We achieve these results by implementing the Crucial Subgoal^K of Section 6.9(6): given $\beta \in$ *JProc*, $K \in cm\cdot\beta$ such that β_K is of type switch, and a cp bisimulation $\mathcal{B}$ for β, we show that the set $\mathcal{B}^u_K \otimes \mathcal{B}^o_{R(K)}$ can be extended to a full cp bisimulation $\mathcal{B}'$ for β. We then immediately conclude that cp bisimilarity is K-decomposable. As a second consequence we achieve that cp bisimilarity is sw-(1)coherent and sw-(1)hereditary.

We work in the context of two live SSMD fc systems $(\mathcal{N}_1, Cover_1)$, $(\mathcal{N}_2, Cover_2)$, and build on the setting of Section 6.9 and 6.10.

Crucial Subgoal^K. For this paragraph, fix $\beta \in JProc$, and $K \in cm\cdot\beta$ such that β_K corresponds to a joint switch place. As usual, set $\mathcal{R} = cm\cdot\beta \backslash K$. We now translate our sketch of Section 6.9(6) into action.

For any cp bisimulation $\mathcal{B}$ for β we define:

$$\begin{array}{ll} \mathcal{B}_s & = \mathcal{B}^u_K \otimes \mathcal{B}^o_{R(K)}, \\ \mathcal{B}_c & = \{\sigma_s t_s \sigma_c \mid \sigma_s \in \mathcal{B}_s,\ \sigma_b t_s \sigma_c \in \mathcal{B},\ last(\sigma_s) = last(\sigma_b)\ \& \\ & \qquad\qquad\qquad proj_i(t_s) \in synchT_i(K_i) \text{ for } i = 1, \text{ or } 2\}, \\ \mathcal{B}' & = \mathcal{B}_s \cup \mathcal{B}_c. \end{array}$$

$\mathcal{B}_c$ is intended to provide matches for synchronizations of K_1, or K_2 respectively, and the behaviour beyond. The SWFSI Prediction Theorems will ensure that $\mathcal{B}_c$ indeed fulfils this role; with their help it is now possible to prove:

Lemma 6.11.1 (Crucial Subgoal^K). *$\mathcal{B}'$ is a cp bisimulation for β.*

Proof. To verify that $\mathcal{B}'$ is a cp bisimulation for β we need to show: (1) $\mathcal{B}' \subseteq JFS(\beta)$, (2) $\beta \in \mathcal{B}'$, (3) $\mathcal{B}'$ is prefix-closed, and (4) $\mathcal{B}'$ satisfies the two bisimulation clauses of Def. 6.4.6. Since $\mathcal{B}$ is a cp bisimulation for β, and $\mathcal{B}_s$ is a cp bisimulation for β up to K-synch (by Corollary 6.9.1), we clearly have: (i) $\mathcal{B} \subseteq JFS(\beta)$ & $\mathcal{B}_s \subseteq JFS(\beta)$, (ii) $\beta \in \mathcal{B}_s$, and (iii) $\mathcal{B}$ and $\mathcal{B}_s$ are prefix-closed. Then, (1) is immediate with (i), (2) follows from (ii), and (3) with (iii).

To prove (4), assume $\sigma \in \mathcal{B}'$ and $proj_i(\sigma) \stackrel{t_i}{\rightarrow}$ for some $t_i \in T_i$, $i = 1, 2$. We need to find $t_{\bar{i}} \in T_{\bar{i}}$, $\sigma' \in JFS$ such that $\sigma \stackrel{(t_1,t_2)}{\rightarrow} \sigma'$, and $\sigma' \in \mathcal{B}'$. We proceed by case analysis; clearly, one of the following three conditions must be satisfied: (a) $\sigma \in \mathcal{B}_c$, (b) $\sigma \in \mathcal{B}_s$ & $t_i \notin synchT_i(K_i)$, or (c) $\sigma \in \mathcal{B}_s$ & $t_i \in synchT_i(K_i)$.

If (a) holds then consider: by definition of $\mathcal{B}_c$ there are $\sigma_s \in \mathcal{B}_s$, and $\sigma_b.t_s.\sigma_c \in \mathcal{B}$ such that $\sigma = \sigma_s.t_s.\sigma_c$. Clearly, $proj_i(\sigma) \xrightarrow{t_i}$ implies $proj_i(\sigma_b.t_s.\sigma_c) \xrightarrow{t_i}$, and thus $\mathcal{B}$ will provide a match for t_i at $\sigma_b.t_s.\sigma_c$. In turn, by definition of $\mathcal{B}_c$ this match will be passed on to σ in $\mathcal{B}_c$, giving us $t_{\bar{i}}$ and σ' as required. Case (b) is also easy: since $\mathcal{B}_s$ provides a cp bisimulation up to K-synch (Corollary 6.9.1) the required entities certainly exist in $\mathcal{B}_s$, and hence in $\mathcal{B}'$.

(c) is the interesting case. With the help of the SWFSI Prediction Theorems we will show: there is $\sigma_b \in \mathcal{B}$ such that $last(\sigma_b) = last(\sigma)$. Clearly, this will settle (c): $\mathcal{B}$ will then contain a match for t_i at σ_b, which will be passed on to σ in $\mathcal{B}'$ via $\mathcal{B}_c$.

Let p^1_{si}, p^2_{si} be given by $last(\sigma) \uparrow K_i = \{(p^1_{si}, p^2_{si})\}$. In preparation, consider the following two basic facts: (F1) p^i_{si} is of type synch, and (F2) $proj_i(\beta)(K_i)$ is of type switch. (F1) follows with $t_i \in synchT_i(K_i)$ and Prop. 6.2.12(4); (F2) is a consequence of our basic assumption 'β_K corresponds to a joint switch place' and Prop. 6.8.6(1). Further, note that by definition of $\mathcal{B}_s$ and Prop. 6.8.15(2) we obtain $(*)$ $\sigma \uparrow K_i \in \mathcal{B}^u_K$, and $(\circ)$ $\sigma \uparrow \mathcal{R}_i \in \mathcal{B}^o_{R(K)}$.

On the one hand, by definition of $\mathcal{B}^u_K$ and $(*)$ there must be $\sigma^\star \in \mathcal{B}$ such that (c) $nosynch(K_i, proj_i(\sigma^\star))$, and (d) $\sigma^\star \uparrow K_i = \sigma \uparrow K_i$. (d) and Prop. 6.8.11 entail $last(\sigma^\star) \uparrow K_i = \{(p^1_{si}, p^2_{si})\}$, from which we infer (e) $proj_i(last(\sigma^\star))(K_i) = p^i_{si}$ (Prop. 6.8.6(1)), and (f) $last(\sigma^\star)(p^i_{si}) = p^{\bar{i}}_{si}$. With (F1), (e), (c), and (F2) it is then immediate that p^i_{si} is $swfsi(K_i, proj_i(\sigma^\star))$. Hence, we can apply Theorem 6.10.2, and with (f) we obtain: (IR) $p^i_{si} \in T\text{-}SWFSIs(proj_i(\beta))$, $K^i_{si} = K_i$ & $p^{\bar{i}}_{si} = cp\text{-}fsi\text{-}Match(\beta, p^i_{si})$.

On the other hand, the definition of $\mathcal{B}^o_{R(K)}$ and $(\circ)$ gives us $\sigma^\circ \in \mathcal{B}$ such that (u) $Ks(proj_i(\sigma^\circ)) \subseteq \mathcal{R}_i$, and (v) $\sigma^\circ \uparrow \mathcal{R}_i = \sigma \uparrow \mathcal{R}_i$. (v) and Prop. 6.8.11 imply (w) $last(\sigma^\circ) \uparrow \mathcal{R}_i = last(\sigma) \uparrow \mathcal{R}_i$. Then, by combining (u),(w), and our interim result (IR) with Theorem 6.10.3, we obtain $\sigma^\diamond$ such that (i) $\sigma^\circ.\sigma^\diamond \in \mathcal{B}$, (ii) $last(\sigma^\circ.\sigma^\diamond) \uparrow K_i = \{(p^1_{si}, p^2_{si})\}$, and (iii) $last(\sigma^\circ.\sigma^\diamond) \uparrow \mathcal{R}_i = last(\sigma) \uparrow \mathcal{R}_i$. Together, (ii) and (iii) give us $last(\sigma^\circ.\sigma^\diamond) = last(\sigma)$, and thus with $\sigma^\circ.\sigma^\diamond$ we have found σ_b as required. □

As intended, we have:

Lemma 6.11.2. *$\mathcal{B}'$ is K-decomposable.*

Proof. By construction of $\mathcal{B}'$ we clearly have $UpToSynch(\mathcal{B}', K) = \mathcal{B}^u_K \otimes \mathcal{B}^o_{R(K)}$. With Prop. 6.9.2(1) and 6.9.3(2) it then follows that $\mathcal{B}'$ is indeed K-decomposable. □

Two Important Consequences. With the previous lemmas it is now immediate:

Theorem 6.11.1. *$\sim_{cp}$ is K-decomposable.*

Proof. This follows from Lemma 6.11.1 and 6.11.2, and Prop. 6.9.8. $\square$

As a second consequence, we obtain:

Theorem 6.11.2.

1. $\sim_{cp} \models$ **sw-(1)her**.

2. $\sim_{cp} \models$ **sw-(1)coh**.

Proof. (1.) Let $\sigma\, t\beta\, t'\beta' \in \sim_{cp}$ such that (1) $t\ I\ t'$, and (2) t' is of type switch. Set $\beta_\sigma = last(\sigma)$, $\beta_\sigma^{pre} = preSet(\beta_\sigma, t)$, $\beta^{post} = postSet(\beta, t)$, and $\beta'_s = \beta'[\beta_\sigma^{pre} \backslash \beta^{post}]$. We need to prove that $\sigma\, t'\beta'_s \in \sim_{cp}$.

Choose $i = 1$, or 2. By (2) and Prop. 6.2.4, $Ks(proj_i(t'))$ is a singleton set; thereby justified, we let K_i be given by (a) $Ks(proj_i(t')) = \{K_i\}$. Further, we set $\mathcal{R}_i = Cover_i \backslash K_i$, and $K = (K_1, K_2)$, where $K_{\bar{i}} = cm \cdot \beta_\sigma(K_i)$. Observe that with (1), (a), and Prop. 6.2.10 we obtain (b) $Ks(proj_i(t)) \subseteq \mathcal{R}_i$.

Set $\sigma' = \beta_\sigma\, t\beta\, t'\beta'$, and $\sigma'_b = \beta_\sigma\, t'\beta'_s$. Note that σ' and σ'_b are related as follows: (C) $\sigma'_b = \sigma' \uparrow K_i \otimes \beta_\sigma \uparrow \mathcal{R}_i$. To see that (C) is indeed satisfied, consider the following three facts: (i) $\sigma' \uparrow K_i = (\beta_\sigma \uparrow K_i)\, t'\, (\beta' \uparrow K_i)$, (ii) $\beta_\sigma = \beta_\sigma \uparrow K_i \cup \beta_\sigma \uparrow \mathcal{R}_i$, and (iii) $\beta'_s = \beta' \uparrow K_i \cup \beta_\sigma \uparrow \mathcal{R}_i$. (i) follows with (a) and (b). (ii) is obvious. To understand (iii) recall that $\beta'_s = \beta'[\beta_\sigma^{pre} \backslash \beta^{post}]$; we split β' into $\beta' \uparrow K_i$ and $\beta' \uparrow \mathcal{R}_i$, and analyse the effect of the substitution operation separately for the two parts: (A) With (b) and Prop. 6.2.2(2) we obtain $K_i \notin Ks(proj_i(\beta_\sigma^{pre}))$ & $K_i \notin Ks(proj_i(\beta^{post}))$. Thus, $\beta' \uparrow K_i$ is not affected by the substitution at all. (B) By $\beta \xrightarrow{t'} \beta'$, (a), and Prop. 6.8.9(1) we have $\beta' \uparrow \mathcal{R}_i = \beta \uparrow \mathcal{R}_i$, and from $\beta_\sigma \xrightarrow{t} \beta$ we infer $\beta_\sigma = \beta[\beta_\sigma^{pre} \backslash \beta^{post}]$. Then, clearly $\beta' \uparrow \mathcal{R}_i[\beta_\sigma^{pre} \backslash \beta^{post}] = \beta_\sigma \uparrow \mathcal{R}_i$. Together with (A) this immediately implies (iii).

Clearly, there is a cp bisimulation $\mathcal{B}$ for β_σ such that $\sigma' \in \mathcal{B}$ (Prop. 6.4.4(1)). With (a) and (b) it is easy to see that $nosynch(K_i, proj_i(\sigma'))$, and thus we can infer $\sigma' \uparrow K_i \in \mathcal{B}^u_K$. On the other hand, we obtain $\beta_\sigma \uparrow \mathcal{R}_i \in \mathcal{B}^o_{R(K)}$ since obviously $\beta_\sigma \in \mathcal{B}$. Together with (C) this implies $\sigma'_b \in \mathcal{B}^u_K \otimes \mathcal{B}^o_{R(K)}$. But from Lemma 6.11.1 we know: there exists a cp bisimulation $\mathcal{B}'$ for β_σ which is based on $\mathcal{B}^u_K \otimes \mathcal{B}^o_{R(K)}$, and hence satisfies $\sigma'_b \in \mathcal{B}'$. Clearly, this gives us $\sigma.t'\beta'_s \in \sim_{cp}$ as required (considering prefix-closure of $\sim_{cp}$ and Prop. 6.4.4(2)).

(2.) follows by an analogous argument. $\square$

Note that to prove K-decomposability we could have based $\mathcal{B}'$ on the weaker $\mathcal{B}^o_K \otimes \mathcal{B}^o_{R(K)}$ just as well: it is easy to check that analogues of Lemma 6.11.1 and 6.11.2 can be proved by a similar argument. Furthermore, these analogues would have been sufficient to prove that $\sim_{cp}$ is sw-(1)coherent: if $\sigma t\beta$, $\sigma t'\beta' \in \sim_{cp}$ with $t \; I \; t'$ and t' of type switch, then $t'\beta'$ can be viewed as 'K-only matching', and $t\beta$ as '$R(K)$-only matching' with respect to $last(\sigma)$ and respective K. In contrast, to gain that $\sim_{cp}$ is sw-(1)hereditary it is essential to employ $\mathcal{B}^u_K$ rather than $\mathcal{B}^o_K$: if $\sigma t\beta t'\beta' \in \sim_{cp}$ with $t \; I \; t'$ and t' of type switch, then $t'\beta'$ can be understood as 'matching up to K-synch' but not as 'K-only matching' with respect to $last(\sigma)$ and respective K.

Had we based $\mathcal{B}'$ on $\mathcal{B}^o_K \otimes \mathcal{B}^u_{R(K)}$ (or even $\mathcal{B}^u_K \otimes \mathcal{B}^u_{R(K)}$) then we would have gained alternative (additional) backtracking capability: we could have inferred that $\sim_{cp}$ is *sw'-(1)hereditary*, where we assume t rather than t' to be of type switch (in Def. 6.4.18), or even more that $\sim_{cp}$ is *sw'-hereditary*, which is full hereditary with t assumed to be of type switch (in Def. 6.4.12). It remains open for now whether it is indeed possible to base $\mathcal{B}'$ on these alternative combinations. Our proof depends on the use of $\mathcal{B}^o_{R(K)}$: it guarantees that the SWFSI Prediction Theorem II (which relies on Prop. 6.10.6) can be applied, or — in terms of the sketch of Section 6.9(6) — that σ° can be extended to $\sigma^\circ.\sigma^\diamond$. The intuition behind this asymmetry can be interpreted as follows: it is up to K-behaviour to decide whether a t-swfsi will indeed occur as swfsi or not. Altogether, this explains why we have to resort to the (1)hereditary property, and split it according to the type of t'.

As far as padding is concerned we obtain that $\sim_{cp}$ is *sw'-coherent* even when basing $\mathcal{B}'$ on the weakest combination $\mathcal{B}^o_K \otimes \mathcal{B}^o_{R(K)}$. However, in view of Section 6.13 it is crucial to employ (1)coherent split according to t': we will exhibit a restriction that induces $\sim_{cp}$ to be sy-(1)coherent and sy-(1)hereditary, but it is not obvious how one could achieve the alternative padding and backtracking properties.

6.12 Interlude II

Having achieved our two main results on live SSMD fc systems, we shall now see how by further restricting our system class we can derive full results for cp and (h)cp bisimilarity, and furthermore for hp and (c)hhp bisimilarity.

By Lemma 6.4.1 we know that Theorem 6.11.2 amounts to solving half of the coincidence problem of cp and chcp bisimilarity in the positive direction. Full coincidence would follow if we could further show that cp bisimilarity is sy-(1)coherent and sy-(1)hereditary. By restricting the nondeterminism available at the postset of a synch transition, we effect that for live *sy-psd* fc systems the matching of synch transitions in $\sim_{cp}$ is deterministic. With this characteristic it is straightforward to achieve that cp bisimilarity satisfies the desired properties sy-(1)coherent and sy-(1)hereditary. Altogether, we then obtain coincidence between cp, hcp, and chcp bisimilarity for live *sy-psd* SSMD fc systems.

As explained in Section 6.1.1 and 6.1.2 there are two further gaps left. One amounts to the difference between bp and cp bisimilarity; overcoming it will achieve a decidability result for chhp bisimilarity. The second gap lies in the difference between hp and bp bisimilarity; additionally closing this gap will give us a coincidence result for hp, hhp, and chhp bisimilarity. Exploiting our knowledge about causes of Section 2.3 we impose a structural constraint on the postset of transitions, and thereby induce bp and cp bisimilarity to coincide for live *buffered* fc systems. By additionally restricting the nondeterminism available at the postset of a transition in a way that subsumes the sy-psd condition, we obtain that cp, bp, and hp bisimilarity coincide for live *spsd* buffered fc systems. Altogether, this gives us decidability of chhp bisimilarity for live sy-psd *buffered* SSMD fc systems, and coincidence between hp, hhp, and (c)hhp bisimilarity for live *spsd* buffered SSMD fc systems.

Accordingly, the remainder is structured as follows. In Section 6.13 we derive our coincidence result for the cp bisimilarities, and in Section 6.14 we at last obtain our results for the hp bisimilarities. The proofs will be straightforward, but technically involved. In particular, to derive our decidability result it is necessary to introduce the intermediate concept bp bisimilarity.

6.13 A Coincidence Result for cp, hcp, and chcp Bisimilarity

In this section we show that cp, hcp, and chcp bisimilarity coincide for the class of live SSMD *synch postset deterministic* (short: *sy-psd*) fc systems. When imposed on live fc systems the sy-psd restriction ensures that the matching of synch transitions in $\sim_{cp}$ is deterministic in the following sense: assuming $\sigma\,(t_1, t_2)\beta \in \sim_{cp}$ with t_i of type synch, $t_{\bar{i}}$ and $postSet(\beta, (t_1, t_2))$ are fully determined by $last(\sigma)({}^\bullet t_i)$. With this characteristic it is straightforward to achieve that cp bisimilarity is sy-(1)coherent and sy-(1)hereditary for live sy-psd fc systems. Together with Theorem 6.11.2 we then obtain that cp bisimilarity is (1)coherent and (1)hereditary for live SSMD sy-psd fc systems, which as we know from Lemma 6.4.1 immediately implies the above coincidence result.

In this section we either assume a fc system $\mathcal{N}$ within the framework of Def. 6.2.19, or we work in the context of two fc systems $\mathcal{N}_1$, $\mathcal{N}_2$, and the setting of Section 6.4. Usually, our results will concern systems of more restricted classes; if this is the case we will specify the system class in the respective proposition or theorem. As usual, if not already instantiated i will range over $\{1, 2\}$.

Basic Observation. Deciding on a cp match for a transition t_i at a joint process β generally involves two degrees of freedom: first, one has to fix an adequate transition $t_{\bar{i}}$ as the counterpart of t_i; second, one has to specify a bijection between the postplaces of t_i and the ones of $t_{\bar{i}}$. Note that the second degree of freedom is dependent on the first but not resolved by it.

A closer look reveals: if t_i is of type synch then the first degree of freedom will never apply; the match $t_{\bar{i}}$ is fully determined by $\beta({}^\bullet t_i)$.

Proposition 6.13.1. *Let $t_i \in T_i$ be of type synch, $\beta \xrightarrow{t}_{cp}$, and $\beta' \xrightarrow{t'}_{cp}$ such that $proj_i(t) = t_i = proj_i(t')$. If $\beta({}^\bullet t_i) = \beta'({}^\bullet t_i)$ then we have $proj_{\bar{i}}(t) = proj_{\bar{i}}(t')$.*

Proof. Let entities be given as above, set $t_{\bar{i}} = proj_{\bar{i}}(t)$, $t'_{\bar{i}} = proj_{\bar{i}}(t')$, and assume $\beta({}^\bullet t_i) = \beta'({}^\bullet t_i)$, say $P_{\bar{i}}$. On the one hand, considering the definition of $\rightarrow_{cp}$ we have ${}^\bullet t_{\bar{i}} = P_{\bar{i}} = {}^\bullet t'_{\bar{i}}$. On the other hand, since t_i is of type synch, we know that $t_{\bar{i}}$ and $t'_{\bar{i}}$ must also be of type synch (Prop. 6.4.10). But then it is clear: $t_{\bar{i}} = t^s_{\bar{i}} = t'_{\bar{i}}$, where $t^s_{\bar{i}}$ is given by $p_{\bar{i}}{}^\bullet = \{t^s_{\bar{i}}\}$ for any $p_{\bar{i}} \in P_{\bar{i}}$. □

On the other hand, there may well be more than one postset match to choose from. In the following, we design a property for transitions which, when imposed on live fc systems, ensures that this second degree of freedom is excluded.

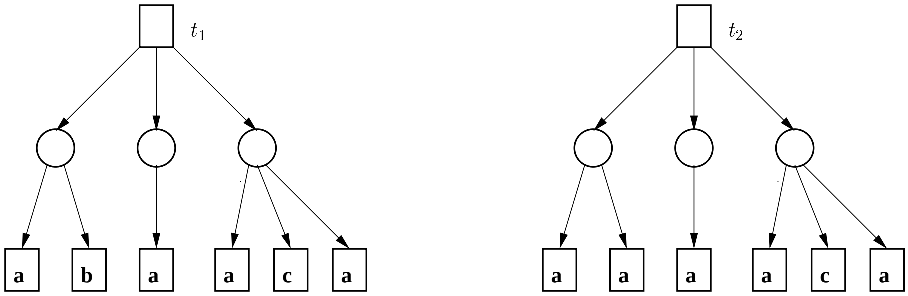


Figure 6.11: Examples of nondeterminism admitted (transition t_1) and disallowed (transition t_2) by the psd restriction

Thereby, we can achieve a system class for which the matching of synch transitions in $\sim_{cp}$ is deterministic in the above sense.

Postset Deterministic Transitions. We define a property for transitions, which in *live* fc systems ensures: let $\beta \overset{(t_1,t_2)}{\rightarrow} \beta'$ with $\beta' \in \sim_{cp}$; if t_1 or t_2 satisfy the property then $postSet(\beta', (t_1, t_2))$ is fully determined by t_1 and t_2. The property is called *postset deterministic*; it works by restricting the nondeterminism available at the postset of a transition.

Definition 6.13.1 (psd transitions). A transition $t \in T_N$ is *postset-deterministic* (short: *psd*) iff for any distinct p_a, $p_b \in t^\bullet$ we have

1. $\exists t_a \in {p_a}^\bullet.\ \forall t_b \in {p_b}^\bullet.\ l(t_a) \neq l(t_b)$, or symmetrically
2. $\exists t_b \in {p_b}^\bullet.\ \forall t_a \in {p_a}^\bullet.\ l(t_b) \neq l(t_a)$.

The psd restriction is illustrated in Figure 6.11. The following can be seen as the characteristic property of psd transitions:

Proposition 6.13.2. *Let $t_1 \in T_1$, $t_2 \in T_2$ such that t_1 or t_2 is psd. There exists at most one bijection $\beta_{post} : {t_1}^\bullet \to {t_2}^\bullet$ such that the following condition is satisfied:*
$(*)$ for all $(p_1, p_2) \in \beta_{post}$ we have

1. *$\forall t_1^x \in {p_1}^\bullet.\ \exists t_2^x \in {p_2}^\bullet.\ l(t_1^x) = l(t_2^x)$, and symmetrically*
2. *$\forall t_2^x \in {p_2}^\bullet.\ \exists t_1^x \in {p_1}^\bullet.\ l(t_1^x) = l(t_2^x)$.*

Proof. Let t_1, t_2 be given as above, and w.l.o.g. suppose t_2 is psd. To the contrary assume two distinct bijections β^a_{post}, $\beta^b_{post} : {t_1}^\bullet \to {t_2}^\bullet$ that both satisfy $(*)$. By distinctness, there must be $p_1 \in {t_1}^\bullet$ such that $\beta^a_{post}(p_1) \neq \beta^b_{post}(p_1)$; set $p_2^a = \beta^a_{post}(p_1)$, and $p_2^b = \beta^b_{post}(p_1)$. Since t_2 is psd, p_2^a and p_2^b satisfy condition (1) or

(2) of Def. 6.13.1. W.l.o.g. suppose that (1) holds, and assume $t_2^a \in p_2^{a\bullet}$ such that (A) $\forall t_2^b \in p_2^{b\bullet}.\ l(t_2^a) \neq l(t_2^b)$. Then, since β^a_{post} satisfies $(*)$ we can infer $t_1^x \in p_1{}^\bullet$ such that $l(t_1^x) = l(t_2^a)$. But note how this leads to a contradiction: since β^b_{post} also satisfies $(*)$ we should be able to find $t_2^b \in p_2^{b\bullet}$ with $l(t_2^b) = l(t_1^x) = l(t_2^a)$; but this is not possible by (A). $\square$

Definition 6.13.2. Let $t_1 \in T_1$, $t_2 \in T_2$ such that t_1 or t_2 is psd. If it exists we denote the uniquely given bijection of Prop. 6.13.2 by $psd(t_1, t_2)$.

In the context of a live fc system, given $M \in Procs$, $p \in M$, every post-transition of p can be enabled by some run of M such that t will directly consume p. With this observation it is then easy to see that postset matches of psd transitions in $\sim_{cp}$ on live fc systems must satisfy property $(*)$ of Prop. 6.13.2. Consequently, we obtain:

Proposition 6.13.3. *Assume $\mathcal{N}_1$ and $\mathcal{N}_2$ to be live fc systems. Let $\beta \overset{(t_1,t_2)}{\to}_{cp} \beta'$ such that $\beta' \in \sim_{cp}$, and t_1 or t_2 is psd. Then $psd(t_1, t_2)$ exists, and we have $postSet(\beta', (t_1, t_2)) = psd(t_1, t_2)$.*

Proof. Let entities be given as above. It is clear: the proposition will follow if we can show that $postSet(\beta', (t_1, t_2))$ satisfies condition $(*)$ of Prop. 6.13.2. Let $(p_1, p_2) \in postSet(\beta', (t_1, t_2))$, and assume $t_i^x \in p_i{}^\bullet$, $i = 1, 2$. We need to exhibit $t_{\bar{i}}^x \in p_{\bar{i}}{}^\bullet$ such that $l(t_1^x) = l(t_2^x)$.

Note that there must be a run $r_i \in Runs_i(proj_i(\beta'))$ satisfying (a) $r_i[t_i^x\rangle$, and (b) $\nexists t_i' \in r_i.\ p_i \in {}^\bullet t_i'$: if t_i^x is of type switch, then by Obs. 6.2.3(1) we can take r_i to be ε. If t_i^x is of type synch then by liveness there must be $r_i \in Runs_i(proj_i(\beta'))$ satisfying (a); by Obs. 6.2.3(2) we can clearly take r_i such that property (b) is also satisfied.

Certainly, there must be a match for $r_i\, t_i^x$ in $\sim_{cp\text{-}\beta'}$, that is we can assume $\sigma\, t^x \beta^x \in \sim_{cp\text{-}\beta'}$ such that $proj_i(\sigma) = r_i$, and $proj_i(t^x) = t_i^x$. Considering the cp transition function and property (b) of r_i, it is easy to see that we must have $t_{\bar{i}}^x \equiv proj_{\bar{i}}(t^x) \in p_{\bar{i}}{}^\bullet$, and $l(t_1^x) = l(t_2^x)$. But this means we have found $t_{\bar{i}}^x$ as required. $\square$

Live sy-psd fc Systems. Now, it is clear: if we restrict live fc systems by requiring that every synch transition is psd then for the corresponding system class we will obtain: the matching of synch transitions in $\sim_{cp}$ is fully determined by the respective preset match.

Definition 6.13.3 (sy-psd fc nets and systems). A fc net N is *sy-psd* iff for all $t \in T_N$ we have: t is of type synch $\implies$ t is psd.

A fc system $\mathcal{N}$ is *sy-psd* iff its underlying net is sy-psd.

Proposition 6.13.4. *Assume $\mathcal{N}_1$ and $\mathcal{N}_2$ to be live sy-psd fc systems. Let $t_i \in T_i$ be of type synch, and $\sigma\, t\beta, \sigma'\, t'\beta' \in \sim_{cp}$ with $proj_i(t) = t_i = proj_i(t')$. If $last(\sigma)({}^\bullet t_i) = last(\sigma')({}^\bullet t_i)$ then we have $t = t'$ & $postSet(\beta, t) = postSet(\beta', t')$.*

Proof. Considering the definition of sy-psd systems, this follows from Prop. 6.13.1 and Prop. 6.13.3. □

Consequences. With this characteristic, it is straightforward to prove that for live sy-psd fc systems cp bisimilarity is sy-(1)coherent and sy(1)-hereditary.

Theorem 6.13.1. *Let $\mathcal{N}_1$, $\mathcal{N}_2$ be live sy-psd fc systems.*

1. $\sim_{cp} \models$ **sy-(1)her**.

2. $\sim_{cp} \models$ **sy-(1)coh**.

Proof. (1.) Let (a) $\sigma\, t\beta\, t'\beta' \in \sim_{cp}$, (b) $t\ I\ t'$, and (c) t' be of type synch. We need to prove that $\sigma\, t'\beta'_s \in \sim_{cp}$, where $\beta'_s = \beta'[preSet(last(\sigma), t) \backslash postSet(\beta, t)]$. Another way of looking at β'_s is given by $\beta'_s = last(\sigma)[postSet(\beta', t') \backslash preSet(\beta, t')]$. Set $t_i = proj_i(t)$, and $t'_i = proj_i(t')$.

Clearly, we have $proj_i(\sigma)\, t_i\, t'_i \in Runs_i$, and $t_i\ I_i\ t'_i$. Then, by reshuffling t_i and t'_i (Prop. 2.1.2) we achieve $proj_i(\sigma) \xrightarrow{t'_i}$. By prefix-closure of $\sim_{cp}$ we infer $\sigma \in \sim_{cp}$, and thus there must be a match for t'_i at σ, that is we can assume $t^\diamond$, $\beta^\diamond$ such that (i) $proj_i(t^\diamond) = t'_i$, (ii) $last(\sigma) \xrightarrow{t^\diamond}_{cp} \beta^\diamond$, and (iii) $\sigma\, t^\diamond\beta^\diamond \in \sim_{cp}$.

By definition of $\rightarrow_{cp}$ and (b) we derive $(*)$ $last(\sigma)({}^\bullet t'_i) = \beta({}^\bullet t'_i)$. Together with (a), (iii), (c), and (i) this means we can apply Prop. 6.13.4 to $\sigma\, t\beta\, t'\beta'$ and $\sigma\, t^\diamond\beta^\diamond$; thereby we obtain: (A) $t' = t^\diamond$, and (B) $postSet(\beta', t') = postSet(\beta^\diamond, t')$. Further, by (ii), considering Prop. 6.4.1, $(*)$, and (B) we can infer $\beta^\diamond = last(\sigma) \backslash preSet(\beta, t') \cup postSet(\beta', t')$, and thus $\beta^\diamond = \beta'_s$. But with (A) and (iii) this means we have achieved $\sigma\, t'\beta'_s \in \sim_{cp}$ as required.

(2.) follows by an analogous argument. □

With Theorem 6.11.2 it is then immediate:

Theorem 6.13.2. *Let $\mathcal{N}_1$, $\mathcal{N}_2$ be live SSMD sy-psd fc systems.*

$$\sim_{cp} \models \textbf{(1)coh} \ \&\ \textbf{(1)her}.$$

Proof. This follows from Theorems 6.11.2 and 6.13.1 with Lemma 6.4.2. □

And further, with Lemma 6.4.1 we obtain our first coincidence result for a fc system class:

Theorem 6.13.3. *Two live SSMD sy-psd fc systems are cp bisimilar iff they are hcp bsimilar iff they are chcp bisimilar.*

Proof. This follows from Theorem 6.13.2 by Lemma 6.4.1. □

Note that we can slightly strengthen this result: making use of the fact that synch transitions are matched to synch transitions, we can generalize Prop. 6.13.4 to the case when only one of the two live fc systems is sy-psd. Then, we can carry over: in the above three theorems it is sufficient to require that only one of the two systems satisfies the sy-psd condition.

6.14 Results for hp and (c)hhp Bisimilarity

In this section we shall at last obtain our results on hp and (c)hhp bisimilarity. Considering the previous section it is clear that there are now two gaps left. One amounts to the difference between bp and cp bisimilarity; overcoming it will achieve a decidability result for chhp bisimilarity. The second gap lies in the difference between hp and bp bisimilarity; additionally closing this gap will give us a coincidence result for hp, hhp, and chhp bisimilarity. We shall develop two restrictions to bridge over these gaps. By constraining the structure at the postset of transitions we induce bp and cp bisimilarity to coincide for live *buffered* fc systems. By additionally restricting the nondeterminism at the postset of transitions (in similar but stricter fashion than the sy-psd condition) we obtain that hp, bp, and cp bisimilarity coincide for live *spsd* buffered fc systems. With our previous results this gives us decidability of chhp bisimilarity for live sy-psd *buffered* SSMD fc systems, and coincidence between hp, hhp, and chhp bisimilarity for live *spsd* buffered SSMD fc systems.

We proceed as follows: first of all, we need to introduce our intermediate concept bp bisimilarity, and present necessary facts about it. Then, we close the gap between bp and cp bisimilarity, and furthermore the gap between hp and bp (and cp) bisimilarity. Finally, we can derive our results on hp and (c)hhp bisimilarity.

Convention 6.14.1. In the context of a free choice system $\mathcal{N}$, given $r \in Runs$ we denote the *switch places of* r by $switchP(r)$, that is we set $switchP(r) = \{p \in M \mid p \text{ is of type switch}\}$, where M is such that $M_0[r\rangle M$. Similarly, we denote the *synch places of* r by $synchP(r)$.

In the context of two systems $\mathcal{N}_1$, $\mathcal{N}_2$, we shall make use of our usual convention: given $r \in SRuns$ we set $r_i = proj_i(r)$, and given $t \in T_1 \times T_2$ we set $t_i = proj_i(t)$, for $i \in \{1, 2\}$.

6.14.1 bp Bisimilarity

bp bisimilarity is designed to reflect the interleaving aspect of the decomposition view DV-lfcs (short: *DVI-lfcs*) (cf. Section 6.1.1). In the following, we shall first present DVI-lfcs and then derive bp bisimilarity from it. After that we collect together the facts that have motivated bp bisimilarity as a convenient intermediate concept. Finally, we clarify the relationship between cp and bp bisimilarity.

DVI-lfcs. DVI-lfcs can be summarized as follows:

> The branching structure of a live fc system is structured by evolving *blocks*, which respect certain aspects of locality and causality while employing *pending synch places* as 'mediators'.

Fix a live fc system $\mathcal{N}$, and adopt the setting of Def. 6.2.19. We explain DVI-lfcs in more detail in the following three points.

(1) Each state of $\mathcal{N}$ can be decomposed into a set of *blocks* and a set of *pending synch places*:

Definition 6.14.1. Let $M \in Proc$.

A subset $b \subseteq M$ is a *block* of M iff there is $t \in T$ such that $M \xrightarrow{t}$ and $b = {}^\bullet t$. We denote the blocks of M by $blocks(M)$.

A place $p \in M$ is a *pending synch place* of M iff $\exists p' \in {}^\bullet(p^\bullet).\ p' \notin M$.

We generalize our notions to runs in the obvious way; that is for $r \in Runs$ we define $b \in blocks(r)$ iff $b \in blocks(M)$, where M is given by $M_0 \xrightarrow{r} M$, and similarly for pending synch places.

Due to the fc restriction for any block b we have ${}^\bullet(b^\bullet) = b$, and thus it is easy to see that any state uniquely partitions into its blocks and pending synch places. In more detail, we have: either (a) a block contains exactly one switch place which enables a non-empty set of switch transitions in the S-system way, or (b) a block consists of several synch places which together enable exactly one synch transition in the T-system way. Accordingly, we classify blocks into *switch blocks* and *synch blocks*.

(2) The system evolves while respecting the structure of its states: if a state M evolves into a new state M' by performing a transition t, the blocks of M will be affected only locally: (a) t will belong to exactly one block, namely ${}^\bullet t$. (b) Blocks other than ${}^\bullet t$ will remain unaffected by the transition. (c) ${}^\bullet t$ will be replaced by a set of new places, which can be structured into a set of new blocks and a set of 'locally' pending synch places; in the context of $M \backslash {}^\bullet t$, the locally pending synch places can either be pending as well, or they can form new blocks by joining with synch places that were pending at M.

(3) The evolving blocks are related by a natural notion of causality, which is obtained as the transitive closure of the following immediate cause relation: let b be a new block that is formed with the execution of a transition t; then, (a) b is immediately dependent on the block that has generated t, and (b) if b is formed

jointly with a previously pending synch place p, b is immediately dependent on the block that has generated t', where t' is the transition that earlier gave rise to p.[8]

bp Bisimilarity. In the following, we work in the context of two live fc systems $\mathcal{N}_1$ and $\mathcal{N}_2$ within a setting as in Section 6.4.

It is straightforward to design a notion of bisimilarity that reflects DVI-lfcs. *block preserving* (short: *bp*) *bisimilarity* ensures: (1) Two related states M_1 and M_2 are decomposed into blocks in the same fashion; we require that $blocks(M_1)$ and $blocks(M_2)$ are linked by a bijection. (2) Whenever two transitions t_1, t_2 are matched against each other at a pair of states M_1, M_2 then t_1 has the same local effect on M_1 as t_2 has on M_2 (with respect to the bijection that relates the blocks of M_1 and M_2); we will realize this by defining a *block preserving transition relation.* (3) The matching of blocks preserves the causal dependencies between them; this is conveniently achieved by basing bp bisimilarity on synchronous runs, and requiring that whenever two blocks b_1, b_2 are related at a joint run r then $mcauses(r_1, t_1)$, where t_1 is any transition enabled by b_1, exactly correspond to $mcauses(r_2, t_2)$, where t_2 is any transition enabled by b_2. This works since respecting the partial order of blocks exactly amounts to respecting the partial order of transitions.

Definition 6.14.2 (bp bisimilarity). A *bp tuple* is a pair (r, β), where $r \in SRuns$, β is a bijection between $blocks(r_1)$ and $blocks(r_2)$, and for all $(b_1, b_2) \in \beta$ we have: $\forall t_1 \in T_1,\ t_2 \in T_2.\ (b_1 = {}^\bullet t_1)\ \&\ (b_2 = {}^\bullet t_2) \Rightarrow (mcauses(r_1, t_1) = mcauses(r_2, t_2))$. We denote the *domain of bp tuples* by BP.

The *bp transition relation*, $\rightarrow_{bp} \subseteq BP \times JT \times BP$, is defined as follows:

$$(r, \beta) \stackrel{(t_1,t_2)}{\rightarrow}_{bp} (r', \beta') \iff$$

1. $r_1 \stackrel{t_1}{\rightarrow} r'_1\ \&\ r_2 \stackrel{t_2}{\rightarrow} r'_2$,
2. $l(t_1) = l(t_2)$,
3. $\beta({}^\bullet t_1) = {}^\bullet t_2$, and
4. $\beta' \lceil_{blocks(r_1) \setminus {}^\bullet t_1} = \beta \lceil_{blocks(r_1) \setminus {}^\bullet t_1}$.

A *bp bisimulation* is a relation $\mathcal{B} \subseteq BP$ that satisfies

[8] Blocks, places, and transitions have to be understood as occurrences here.

1. If $(r, \beta) \in \mathcal{B}$ and $r_1 \xrightarrow{t_1}$ for some $t_1 \in T_1$, then there are $t_2 \in T_2$, $(r', \beta') \in BP$ such that $(r, \beta) \xrightarrow{(t_1,t_2)}_{bp} (r', \beta')$ and $(r', \beta') \in \mathcal{B}$.

2. Vice versa.

Two runs $r_1 \in Runs_1$, $r_2 \in Runs_2$ are *bp bisimilar* w.r.t. $(r, \beta) \in BP$, written $(r, \beta) \in \sim_{bp}$, iff $proj_i(r) = r_i$ for $i = 1, 2$, and $(r, \beta) \in \mathcal{B}$ for some bp bisimulation $\mathcal{B}$. That is, we define $\sim_{bp} = \bigcup\{\mathcal{B} \mid \mathcal{B}$ is a bp bisimulation$\}$.

$\mathcal{N}_1$ and $\mathcal{N}_2$ are *bp bisimilar* iff $((\varepsilon, \varepsilon), \beta) \in \sim_{bp}$ for some β.

Properties of bp Bisimilarity. We now collect together the properties that commend bp bisimilarity as an intermediate concept in our quest for a coincidence result on hp and (c)hhp bisimilarity (cf. Section 6.1.1).

Firstly, it is easy to see that bp bisimilarity is decidable for finite-state systems: in the definition of BP we can use gsc's instead of synchronous runs just as well; then for finite-state systems BP will be a finite domain, and bp bisimilarity can be decided by exhaustive search.

Fact 6.14.1. *bp bisimilarity is decidable.*

Secondly, since block assignments are cause-preserving and bp tuples are based on synchronous runs, it immediately follows that bp bisimilarity is a strengthening of hp bisimilarity:

Fact 6.14.2. *bp bisimilarity implies hp bisimilarity. Formally, we have:*

$$(r, \beta) \in \sim_{bp} \Longrightarrow \ r \in \sim_{hp} .$$

Finally, with a bit more effort, we obtain that chhp bisimilarity implies bp bisimilarity. The proof rests on the following two insights: (1) Given a hp bisimulation $\mathcal{H}$, and a pair of runs $(r_1, r_2) \in \mathcal{H}$, we can obtain a bijection between the blocks of r_1 and the ones of r_2 by considering a maximal concurrent step of r_1 (or r_2) and observing how it is matched in $\mathcal{H}$. This is so because: (a) a maximal concurrent step γ of a run r exactly defines the blocks of r (each transition of γ corresponds to a block of r); and (b) by Prop. 4.3.2(2) maximal steps are matched against maximal steps. (2) It can be shown that in a *coherent* and *hereditary* hp bisimulation transitions are matched in accordance with any bijection of blocks that is obtained via insight (1); thus each chhp bisimulation can be transformed into a bp bisimulation.

Fact 6.14.3. *chhp bisimilarity implies bp bisimilarity.*

Altogether this means: if we achieve coincidence between bp bisimilarity and its corresponding coherent and hereditary version, we will obtain decidability of chhp bisimilarity (consider that by Fact 6.14.2 chbp bisimilarity will certainly imply chhp bisimilarity); if we additionally manage to show that hp and bp bisimilarity coincide, we can infer coincidence between hp, hhp, and chhp bisimilarity. However, we have worked with cp bisimilarity rather than bp bisimilarity, and to exploit the coincidence result of the previous section we will further need to overcome the discrepancy resulting from this simplification. Therefore, let us now analyse the difference between cp and bp bisimilarity.

bp and cp Bisimilarity. It is easy to see that cp bisimilarity implies bp bisimilarity: a bijection between two processes clearly implies a bijection between their blocks. Further, if a joint transition is compositionality preserving in the sense of cp bisimilarity it will certainly be block preserving w.r.t. the thus induced bijection of blocks. Finally, by Prop. 6.4.5 we know that cp bisimilarity is cause-preserving, which ensures that the induced block assignments are cause-preserving as well, and that the induced transitions produce synchronous runs.

Fact 6.14.4. *cp bisimilarity implies bp bisimilarity.*

The other direction does not hold: bp bisimilarity is still behavioural in that it abstracts away from 'behaviourally irrelevant' places; the counter-example of Figure 6.7, which proved non-coincidence between cp and chhp bisimilarity, also demonstrates that bp bisimilarity does not imply cp bisimilarity. However, as before, we could argue that the counter-example merely highlights *technical inaccuracies*, and that these might be overcome by employing a more sophisticated version of cp bisimilarity. This being so, asssume we have indeed managed to handle the technical inaccuracies, and that from a bijection between blocks we can now infer a matching between the places of the blocks in a style as it is required by the new cp bisimilarity. After all, there still remains an essential difference between cp and bp bisimilarity: bp bisimilarity can only provide a match for place occurrences that are active in that they take part in enabling a transition, whereas in cp bisimilarity we need to match places as soon as they come into existence; in particular, this creates a problem for the matching of *pending synch places*. In our context of live systems, any pending occurrence of a synch place can become active later on, and thus we could hope to solve the problem by employing future matches. However, to make this work we would need to know that retrieved matches are uniform in that they are valid for all possible futures, which is a priori not provided by bp bisimilarity: different computation

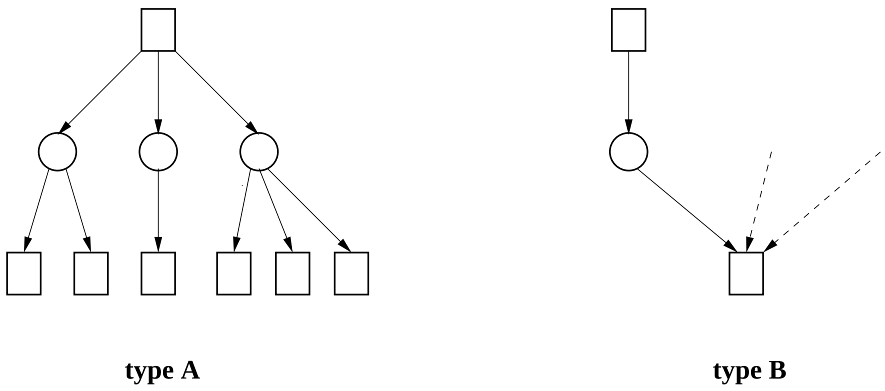


Figure 6.12: An example of the two substructures allowed at the postset of a transition in buffered fc nets

paths may lead to different matches of a pending place; future assignments could be dependent on the order in which independent transitions are linearized. We consider this issue of pending synch places to be the key difference between bp and cp bisimilarity (and equally between chhp and cp bisimilarity).

6.14.2 Overcoming the Gap between bp and cp Bisimilarity

We shall now see that the difference between bp and cp bisimilarity can be overcome by imposing a slight structural constraint: the two bisimilarities coincide for live *buffered* fc systems. The *buffered* restriction enforces a natural bijection between the synch places of a pair $(r_1, r_2) \in \sim_{hp}$ such that the block assignment of any $(\gamma, (r_1, r_2)) \in \sim_{bp}$ will be dictated to respect that bijection. This will resolve the issue of pending synch places and the technical inaccuracies at the same time. An assignment for switch blocks will exactly correspond to a bijection between switch places. Our restriction is defined as follows:

Definition 6.14.3 (buffered). A fc net N is *buffered* iff for each arc $(t, s) \in F_N \cap (T_N \times S_N)$ we have: $t^\bullet = \{s\}$ or s is of type switch.

A fc system $\mathcal{N} = (N, M_0)$ is *buffered* iff N is buffered, and M_0 contains only places of type switch.

As illustrated by Figure 6.12, in buffered fc systems the postset of each transition consists either of a set of switch places, or of exactly one synch place. Accordingly, we classify the transitions of our subclass into two types, called A and B. These 'buffered types' are orthogonal to our standard 'free choice types' switch and synch.

Definition 6.14.4. Let N be a buffered fc net. We say $t \in T_N$ is

- of *type A* iff all the places of $t^\bullet$ are of type switch, and
- of *type B* otherwise (that is iff $t^\bullet$ consists of a single place of type synch).

Let $\mathcal{N}$ be a buffered fc system. In the context of $\mathcal{N}$ we set *init* to be of type A.

In the following, we either assume a live buffered fc system $\mathcal{N}$ within the framework of Def. 6.2.19, or we work in the context of two live buffered fc systems $\mathcal{N}_1$, $\mathcal{N}_2$ and a setting as in Section 6.4. Our proof exploits our terminology and knowledge concerning causes of Section 2.3.

We start with two straightforward observations:

Proposition 6.14.1. *Let $r \in Runs$, M be given by $M_0[r\rangle M$, and $p \in M$.*

1. *p is of type switch $\iff$ $t^{gen(r,p)}$ is of type A, or equivalently*

   p is of type synch $\iff$ $t^{gen(r,p)}$ is of type B.
2. *p is of type synch $\implies$ $\left(t^{gen(r,p)}\right)^\bullet = \{p\}$.*

Proof. (1) is easy by induction on the length of r. (2) is immediate with (1). $\square$

Prop. 6.14.1(2) captures what for our proof is the key property of buffered fc systems: let $r \in Runs$; each synch place of r is uniquely identified by its generator event in r. Now, let $r \in SRuns$. Imagine r preserves the buffered types in that two related transitions of r are either both of type A or both of type B. Then, considering Prop. 6.14.1(2), there is an obvious way of obtaining a bijection between the synch places of r_1 and the ones of r_2: match $p_1 \in synchP(r_1)$ against $p_2 \in synchP(r_2)$ iff their generator events are related in r, that is iff $gen(r_1, p_1) = gen(r_2, p_2)$. We shall refer to this assignment by $syass(r)$.

Naturally, we would like to employ $syass(r)$ to obtain a translation from bp tuples contained in $\sim_{bp}$ to the domain of joint processes with the goal of transforming any bp bisimulation into a cp bisimulation. To implement this plan we will essentially have to prove that the following two requirements are satisfied for every $(r, \gamma) \in \sim_{bp}$: (1) r preserves types A and B, and hence $syass(r)$ is defined; (2) the block assignment γ respects $syass(r)$. By exploiting Prop. 6.14.1(2) and the fact that transitions of $\mathcal{T}_{hp}$ and the block assignments of bp tuples preserve maximal causes, it will be straightforward to show that the two requirements are indeed given. In fact, we will also achieve the analogues of (1) and (2) for $r \in \sim_{hp}$. This is important in view of the next section, and we shall include the corresponding insights here.

First of all, we shall see that in buffered fc systems structural synchronization (of places) coincides with behavioural synchronization (of observable threads in the sense of SW-3): whenever a transition t is enabled at a run r then each preplace of t is uniquely represented by a maximal cause of t in r. Formally, we have:

Proposition 6.14.2. *Let $r \in Runs$, and $t \in T$ such that $r \xrightarrow{t}$.*

1. $mcauses(r, t) = icauses(r, t)$ $(= gen(r, {}^\bullet t))$.
2. $|gen(r, {}^\bullet t)| = |{}^\bullet t|$.

Proof. Let r and t be given as above.

(1) Considering Prop. 2.3.2 we only need to show that any immediate cause cannot be subsumed by another immediate cause; that is:

$$(*)\ \forall i \in icauses(r, t).\ \nexists j \in icauses(r, t).\ i < j.$$

Recall that by definition we have $icauses(r, t) = gen(r, {}^\bullet t)$. If t is of type switch, then $gen(r, {}^\bullet t)$ is a singleton since ${}^\bullet t$ is a singleton. This immediately implies $(*)$. On the other hand, assume t to be of type synch. Consider $i \in icauses(r, t)$, which means $i = gen(r, p)$ for some $p \in {}^\bullet t$. It is easy to see that i cannot be subsumed by any other immediate cause: clearly, p is of type synch; but then by Prop. 6.14.1(2) t^i has only p as postplace, and thus $i <_{prox} |r| + 1$ is the only possible causal line leading from i to $|r| + 1$ in $r.t$.

(2) If t is of type switch then clearly both, ${}^\bullet t$ and $gen(r, {}^\bullet t)$ are singleton sets. If t is of type synch then all the places of ${}^\bullet t$ are of type synch, and (2) follows from Prop. 6.14.1(2). □

As a natural consequence of this, the issue of technical inaccuracies is resolved for (live) buffered fc systems. In particular, this means synchronous runs as well as the block assignments of bp tuples respect the free choice types switch and synch. Exploiting liveness, from the first fact and Prop. 6.14.2(1) we then obtain that types A and B are preserved by all $r \in \sim_{hp}$, which immediately implies our first requirement is achieved.

Proposition 6.14.3.

1. *Let $r \in SRuns$, and $t \in JT$ such that $r_i \xrightarrow{t_i}$ for $i \in \{1, 2\}$.*

   $mcauses(r_1, t_1) = mcauses(r_2, t_2) \implies$
   $\qquad t_1$ *is of type switch (synch)* $\iff$ t_2 *is of type switch (synch).*

2. *For all $r \in SRuns$ we have:*

   $\forall t \in r.\ t_1$ *is of type switch (synch)* $\iff$ t_2 *is of type switch (synch).*

3. *For all $(r, \gamma) \in BP$ we have:*

   $\forall b \in \gamma.\ b_1$ *is of type switch (synch)* $\iff$ b_2 *is of type switch (synch).*

4. *For all $r \in \sim_{hp}$ (and hence for all r such that $\exists \gamma.\ (r, \gamma) \in \sim_{bp}$) we have:*

   $\forall t \in r.\ t_1$ *is of type A (B)* $\iff$ t_2 *is of type A (B).*

Proof. (1) Let r, t be given as above, and assume $mcauses(r_1, t_1) = mcauses(r_2, t_2)$. By Prop. 6.14.2 (1) and (2) we obtain $|{}^\bullet t_1| = |{}^\bullet t_2|$. Since for $i \in \{1, 2\}$ $|{}^\bullet t_i| = 1$ iff t_i is of type switch ($|{}^\bullet t_i| > 1$ iff t_i of type synch) this immediately implies t_1 and t_2 must either be both of type switch or both of type synch.

(2) This is easy by induction on the length of r: for the inductive case consider that transitions of $\mathcal{T}_{hp}$ respect maximal causes, and consequently apply (1).

(3) Let $(r, \gamma) \in BP$, and $b \in \gamma$. For $i \in \{1, 2\}$ set t_i such that $r_i \xrightarrow{t_i}$ and ${}^\bullet t_i = b_i$. By definition of BP we have $mcauses(r_1, t_1) = mcauses(r_2, t_2)$, and thus by (1) t_1 and t_2 are either both of type switch or both of type synch. But then the analogue must hold for b_1 and b_2.

(4) Let $r \in \sim_{hp}$. We prove (4) by induction on the length of r. The base case vacuously holds. Assume $r \equiv r'.t$. By induction hypothesis and prefix-closure of $\sim_{hp}$ (4) holds for r', and we only have to show: t_1 is of type A (B) iff t_2 is of type A (B). By liveness we can choose $t_1^* \in (t_1{}^\bullet)^\bullet$ (Fact 6.2.2) and a run r_1^* such that $r_1.r_1^* \xrightarrow{t_1^*}$ and $|r| \in gen(r_1.r_1^*, {}^\bullet(t_1^*))$. Clearly, $\sim_{hp}$ contains a match for $r_1^*.t_1^*$ at r; this implies there must be $r^*.t^* \in SRuns$ such that $proj_1(r^*) = r_1^*$, and $proj_1(t^*) = t_1^*$. Since transitions of $\mathcal{T}_{hp}$ respect maximal causes, and by Prop. 6.14.2(1) they coincide with immediate causes, we infer $|r| \in gen(r_2.r_2^*, {}^\bullet(t_2^*))$. Assume t_1 is of type A (B). Then by choice of t_1^* it is easy to see that t_1^* must be of type switch (synch). By (2) this implies t_2^* is also of type switch (synch), and by Prop. 6.14.1(1) and $t^{|r|} = t$ we can conclude back that t_2 must be of type A (B) as required. The opposite direction follows from the symmetrical argument; alternatively consider the brackets. □

Furthermore, it is now straightforward to prove that transitions of $\mathcal{T}_{hp}$ and the block assignments of bp tuples will be in agreement with $syass(r)$.

Proposition 6.14.4.

1. *Let $r \in SRuns$, and $t \in JT$ such that $r_i \xrightarrow{t_i}$ for $i \in \{1, 2\}$, and t_1 or t_2 is of type synch.*

   (a) $mcauses(r_1, t_1) = mcauses(r_2, t_2) \implies {}^\bullet t_2 = (tr(r_2, gen(r_1, {}^\bullet t_1)))^\bullet$.

   (b) $r \xrightarrow{t} \implies {}^\bullet t_2 = (tr(r_2, gen(r_1, {}^\bullet t_1)))^\bullet$.

2. *Let* $(r, \gamma) \in BP$, *and* $b_1 \in blocks(r_1)$, $b_2 \in blocks(r_2)$ *such that* b_1 *or* b_2 *is of type synch.*

$$\gamma(b_1) = b_2 \implies b_2 = (tr(r_2, gen(r_1, b_1)))^\bullet.$$

Proof. (1a) Let r, t be as above, and assume $mcauses(r_1, t_1) = mcauses(r_2, t_2)$. By Prop. 6.14.3(1) we obtain that both, t_1 and t_2, must be of type synch, and by Prop. 6.14.2(1) we gain $gen(r_1, {}^\bullet t_1) = gen(r_2, {}^\bullet t_2)$. But then (1a) immediately follows from Prop. 6.14.1(2).

(1b) Since transitions of $\mathcal{T}_{hp}$ respect maximal causes, this is a direct consequence of (1a).

(2) also follows from (1a): let r, b_1, b_2 be given as above, and for $i \in \{1, 2\}$ set t_i such that $r_i \xrightarrow{t_i}$ and ${}^\bullet t_i = b_i$; clearly t_1 or t_2 must be of type synch, and by definition of BP we have $mcauses(r_1, t_1) = mcauses(r_2, t_2)$. ☐

The remainder is routine, and we shall only sketch the necessary steps; a full proof can be found in Appendix C.4.1. First, one defines a map *jproc* to translate every $\rho \equiv (r, \gamma) \in \sim_{bp}$ into a corresponding joint process. $jproc(\rho)$ gives a bijection between the places of r_1 and those of r_2 as follows: (1) synch places of r_1 are matched against synch places of r_2 according to $syass(r)$, which is defined due to Prop. 6.14.3(4); (2) switch places of r_1 are matched against switch places of r_2 as given by γ: each switch place is represented by exactly one switch block, and by Prop. 6.14.3(3) γ matches switch blocks against switch blocks. With Prop. 6.14.4(2) it is immediate that ρ respects its associated joint process in the following way:

let $b_1 \in blocks(r_1),\ b_2 \in blocks(r_2).\ \gamma(b_1) = b_2 \implies jproc(\rho)(b_1) = b_2.$

This ensures that transitions of $\mathcal{T}_{bp}$ that relate bp tuples contained in $\sim_{bp}$ translate into transitions of $\mathcal{T}_{cp}$:

let $t \in JT$, $\rho' \in BP$. $\rho \xrightarrow{t}_{bp} \rho'$ & $\rho' \in \sim_{bp} \implies jproc(\rho) \xrightarrow{t}_{cp} jproc(\rho')$.

With this, in turn, it is straightforward to show that any bp bisimulation can be transformed into a cp bisimulation, and hence we obtain:

Lemma 6.14.1. *For all* $\rho \in BP$ *we have:*

$$\rho \in \sim_{bp} \implies jproc(\rho) \text{ is defined } \& \ jproc(\rho) \in \sim_{cp}.$$

Finally, we conclude:

Theorem 6.14.1. *Two live buffered fc systems are bp bisimilar iff they are cp bisimilar.*

Proof. This is immediate with Fact 6.14.4 and Lemma 6.14.1. ☐

6.14.3 Overcoming the Gap between hp and bp Bisimilarity

We shall now close the only remaining gap. Building on the buffered restriction we design a constraint to additionally overcome the difference between hp and bp bisimilarity: we show that hp, bp, and cp bisimilarity coincide for the class of live *strictly postset-deterministic* (short: *spsd*) buffered fc systems.

Fix two live buffered fc systems $\mathcal{N}_1$, $\mathcal{N}_2$, and assume the framework of Section 6.4. Let $r \in \sim_{hp}$. From the previous section we know that the buffered condition enforces a natural bijection between the synch places of r_1 and those of r_2 such that the assignment is respected by transitions of $\mathcal{T}_{hp}$. If we achieved the analogue for switch places then certainly we could translate any hp bisimulation into a cp bisimulation. From the previous section we also know: (1) the switch places of r_1, and r_2 respectively, have been generated by transitions of type A; (2) r preserves types, and hence transitions of type A are matched against transitions of type A. Thus, as a first rule for obtaining an assignment for switch places it seems natural to adopt the following constraint: allow $p_1 \in switchP(r_1)$ to match $p_2 \in switchP(r_2)$ only if their generator events are related in r, that is only if $gen(r_1, p_1) = gen(r_2, p_2)$. The following observation implies that this rule is in agreement with how switch places are consumed by transitions of $\mathcal{T}_{hp}$:

Convention 6.14.2. For $t_i \in T_i$ of type switch we write ${}^\circ t_i$ to denote the singleton preplace of t_i.

Proposition 6.14.5. *Let $r \in SRuns$, and $t \in JT$ such that $r_1 \xrightarrow{t_1}$, $r_2 \xrightarrow{t_2}$, and t_i is of type switch, where $i = 1$, or 2.*

$$\begin{aligned} & mcauses(r_1, t_1) = mcauses(r_2, t_2) \implies \\ & \qquad t_{\bar{i}} \text{ is of type switch } \;\&\; {}^\circ t_{\bar{i}} \in (t^e)^\bullet, \text{ where } e = gen(r_i, {}^\circ t_i). \end{aligned}$$

Proof. Let r, t, and i be given as above, and assume (A) $mcauses(r_1, t_1) = mcauses(r_2, t_2)$. By Prop. 6.14.3(1) and (A) it is clear that both, t_1 and t_2, must be of type switch. On the other hand, by Prop. 6.14.2(1) (A) gives us $gen(r_1, {}^\circ t_1) = gen(r_2, {}^\circ t_2)$. Since we clearly have ${}^\circ t_{\bar{i}} \in (t^{gen(r_{\bar{i}}, {}^\circ t_{\bar{i}})})^\bullet$, this immediately implies the proposition. $\square$

In fact, our constraint corresponds to the definition of *syass*, but, since transitions of type A can have several postplaces, it can only act as a partial definition here. We additionally require a rule that tells us of how to assign the postplaces of two transitions of type A related in $\sim_{hp}$. Thus, on top of the buffered condition we want a restriction that enforces a natural bijection between the postplaces

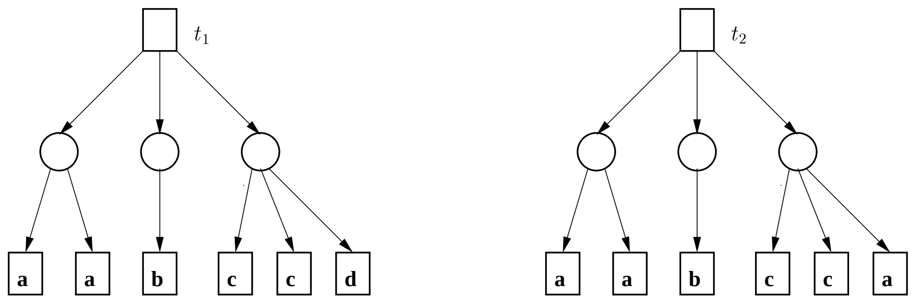


Figure 6.13: Examples of nondeterminism admitted (transition t_1) and disallowed (transition t_2) by the spsd restriction

of any such pair of transitions. Moreover, the assignment should be respected by transitions of $\mathcal{T}_{hp}$. This is exactly what we achieve with the spsd constraint. It works by restricting the nondeterminism at the postset of transitions (in the spirit of, but stricter than, the sy-psd condition of Section 6.13), and exploits that transitions of $\mathcal{T}_{hp}$ are not only cause-preserving but also label-preserving. It is defined as follows:

Definition 6.14.5. Let $\mathcal{N}$ be a net system.

$t \in T_N \cup \{init\}$ is *strictly postset-deterministic* (short: *spsd*) iff for any distinct $p_a, p_b \in t^\bullet$ we have: $\forall t_a \in p_a{}^\bullet.\ \forall t_b \in p_b{}^\bullet.\ l(t_a) \neq l(t_b)$.

$\mathcal{N}$ is *spsd* iff for all $t \in T_N \cup \{init\}$ we have: t is spsd. (If $\mathcal{N}$ is buffered fc, this is equivalent to: t is of type A $\implies$ t is spsd.)

The spsd restriction is illustrated in Figure 6.13. It is easy to see that it subsumes the sy-psd restriction:

Proposition 6.14.6. *If a fc system is spsd then it is also sy-psd.*

Proof. Obvious. $\square$

In the following, we assume $\mathcal{N}_1$ and $\mathcal{N}_2$ to be live *spsd* buffered fc systems. First, we shall see that the spsd restriction indeed enforces a natural bijection between the postplaces of two transitions t_1, t_2 of type A that are matched against each other in $\sim_{hp}$: there exists exactly one bijection $\beta_{post} : t_1{}^\bullet \to t_2{}^\bullet$ such that whenever two places are related in β_{post} then they have label-matching posttransitions.

Proposition 6.14.7. *Let $t \in JT \cup \{init\}$ such that*

$$\begin{array}{lll} t = init & \implies & r'' \in \sim_{hp} \text{ for some } r'', \text{ and} \\ t \in JT & \implies & t_1, t_2 \text{ are of type } A \ \ \& \ r'.t.r'' \in \sim_{hp} \text{ for some } r', r''. \end{array}$$

There exists exactly one bijection $\beta_{post} : {t_1}^\bullet \to {t_2}^\bullet$ such that the following condition is satisfied:

(∗) for all $(p_1, p_2) \in \beta_{post}$ we have

1. $\forall t_1^x \in {p_1}^\bullet .\ \exists t_2^x \in {p_2}^\bullet .\ l(t_1^x) = l(t_2^x)$*, and symmetrically*

2. $\forall t_2^x \in {p_2}^\bullet .\ \exists t_1^x \in {p_1}^\bullet .\ l(t_1^x) = l(t_2^x)$.

Proof. This can essentially be proved by employing the following three ingredients: (1) $\sim_{hp}$ matching is maximal cause preserving and Prop. 6.14.5; (2) concurrent steps have to be matched against concurrent steps (Prop. 4.3.2(1)); (3) $\sim_{hp}$ matching is label-preserving. The proof is lengthy, and has therefore be moved to Appendix C.4.2(1). □

Definition 6.14.6. Let t be defined as in Prop. 6.14.7. We denote the uniquely given bijection of Prop. 6.14.7 by $psd(t)$.

Now it is clear how we can obtain a bijection between the switch places of a pair $(r_1, r_2) \in \sim_{hp}$: match $p_1 \in switchP(r_1)$ against $p_2 \in switchP(r_2)$ iff $p_2 = psd(t^e)(p_1)$, where $e = gen(r_1, p_1)$. As anticipated t_1^e and t_2^e must both be of type A, and thus the assignment is defined. We shall refer to it by $swass(r)$. On the other hand, we still have to prove that $swass(r)$ will be respected by transitions of $\mathcal{T}_{hp}$. This will follow from Prop. 6.14.5 and the following observation:

Proposition 6.14.8. *Let $t \in JT \cup \{init\}$ such that $psd(t)$ is defined.*

$$\forall t_1^a \in ({t_1}^\bullet)^\bullet .\ \forall t_2^a \in ({t_2}^\bullet)^\bullet .\ l(t_1^a) = l(t_2^a) \implies {}^\circ t_2^a = psd(t)({}^\circ t_1^a).$$

Proof. Let t be given as above, and assume $t_1^a \in ({t_1}^\bullet)^\bullet$ and $t_2^a \in ({t_2}^\bullet)^\bullet$ such that (A) $l(t_1^a) = l(t_2^a)$. It is clear that t_1^a and t_2^a are of type switch satisfying ${}^\circ t_1^a \in {t_1}^\bullet$ and ${}^\circ t_2^a \in {t_2}^\bullet$. By definition of $psd(t)$ there exists $t_2^b \in (psd(t)({}^\circ t_1^a))^\bullet$ such that $l(t_1^a) = l(t_2^b)$. But by the spsd restriction and (A) this immediately implies ${}^\circ t_2^b = {}^\circ t_2^a$, which in turn proves the proposition. □

Now it is straightforward to prove:

Proposition 6.14.9. *Let $r \in \sim_{hp}$, and $t \in JT$ such that t_i is of type switch for $i = 1$, or 2.*

$$r \xrightarrow{t} \implies \begin{array}{l} t_1,\ t_2 \textit{ are both of type switch } \& \\ psd(t^e)({}^\circ t_1) \textit{ is defined } \& \\ {}^\circ t_2 = psd(t^e)({}^\circ t_1), \textit{ where } e = gen(r_1, {}^\circ t_1). \end{array}$$

Proof. Let r and t be given as above, and assume $r \xrightarrow{t}$. By Prop. 6.14.3(2) it is clear that both, t_1 and t_2, must be of type switch. It follows that t_1^e and t_2^e must be of type A (Prop. 6.14.1(1), 6.14.3(4)), and hence, $psd(t^e)$ is defined; $^\circ t_1 \in (t^e)^\bullet$ is obvious. Finally, considering that transitions in $\mathcal{T}_{hp}$ respect maximal causes as well as labels, with Prop. 6.14.5 and Prop. 6.14.8 we easily obtain $^\circ t_2 = psd(t^e)(^\circ t_1)$. □

The remainder is routine and analogous to Section 6.14.2; a full proof can be found in Appendix C.4.2(2). We define a map *jproc* to translate every $r \in \sim_{hp}$ into a corresponding joint process; this time *jproc* is composed of *syass* and *swass*. With Prop. 6.14.4(1b) and Prop. 6.14.9 it is then immediate that r respects $jproc(r)$ in the following way: let $t \in JT$. $r \xrightarrow{t} \implies jproc(r)(^\bullet t_1) = {^\bullet t_2}$. And thus, transitions of $\mathcal{T}_{hp}$ that are contained in $\sim_{hp}$ translate into transitions of $\mathcal{T}_{cp}$: $r \xrightarrow{t} r.t \;\&\; r.t \in \sim_{hp} \implies jproc(r) \xrightarrow{t}_{cp} jproc(r.t)$. This, in turn, makes it straightforward to prove that hp bisimilarity implies cp bisimilarity:

Lemma 6.14.2. *For all $r \in SRuns$ we have:*

$$r \in \sim_{hp} \implies jproc(r) \textit{ is defined } \& \; jproc(r) \in \sim_{cp} .$$

Finally we conclude:

Theorem 6.14.2. *Two live spsd buffered fc systems are hp bisimilar iff they are bp bisimilar iff they are cp bisimilar.*

Proof. This is immediate with Fact 6.14.2, Lemma 6.14.2, and Fact 6.14.4. □

6.14.4 Final Results

At last, we are able to derive our results for hp and (c)hhp bisimilarity. Closing the gap between bp and cp bisimilarity gives us our decidability result:

Theorem 6.14.3.

1. *Two live sy-psd buffered SSMD fc systems are cp bisimilar iff they are chhp bisimilar.*

2. *It is decidable whether two live sy-psd buffered SSMD fc systems are chhp bisimilar.*

Proof. (1.) Our results give rise to the following circuit of inclusions and equivalences, which clearly proves (1):

$$\sim_{chhp} \stackrel{a}{\subseteq} \sim_{bp} \stackrel{b}{=} \sim_{cp} \stackrel{c}{=} \sim_{chcp} \stackrel{d}{\subseteq} \sim_{chhp} .$$

(a) follows by Fact 6.14.3, (b) by Theorem 6.14.1 and the buffered restriction, (c) by Theorem 6.13.3 and the sy-psd restriction, and (d) by Fact 6.4.3(2).

(2.) follows from (1) and the decidability of cp bisimilarity (Fact 6.4.2). □

And, by furthermore closing the gap between hp and bp bisimilarity we obtain our coincidence result:

Theorem 6.14.4. *Two live spsd buffered SSMD fc systems are hp bisimilar iff they are hhp bisimilar iff they are chhp bisimilar.*

Proof. Considering Prop. 6.14.6 and Theorem 6.14.2 we can extend the above circuit of inclusions and equivalences as follows:

$$\sim_{chhp} \;\subseteq\; \sim_{hhp} \;\subseteq\; \sim_{hp} \;\overset{e}{=}\; \sim_{bp} \;\overset{e}{=}\; \sim_{cp} \;\overset{c}{=}\; \sim_{chcp} \;\overset{d}{\subseteq}\; \sim_{chhp},$$

where the unlabelled relations are trivial, and e corresponds to Theorem 6.14.2. □

Chapter 7

Final Remarks

Although hhp bisimilarity is obtained from hp bisimilarity by the seemingly small addition of a backtracking requirement the computational and distinguishing power of hhp bisimilarity is far greater than that of hp bisimilarity. The reason for this lies in the fact that by introducing backtracking we abandon the usual view of a concurrent system as a tree of future behaviour and take ourselves to a mathematically more involved structure: the truly-concurrent unfolding level where the interplay of causality, concurrency, and conflict is fully visible and exploitable. We hope to have shown throughout this thesis how by imposing restrictions on this interplay we can systematically approach the borderlines of power of a truly-concurrent concept such as hhp bisimilarity. We also hope this work demonstrates how mathematically intriguing working with true-concurrency can be. Finally, we hope the following conclusions will show how this thesis takes us a very small step towards a unified theory of true-concurrency, and that there may be benefits for automatic verification.

In Section 7.1 we summarize our results, draw conclusions, and discuss some shortcomings; in doing so directions for future work will arise. In Section 7.2 we inspect the undecidability of hhp bisimilarity. Finally, in Section 7.3 we outline two general directions for further research; in particular, we speculate what might be gained with respect to automatic verification.

7.1 Summary and Conclusions

7.1.1 Summary

Firstly we shall summarize the results of this thesis. In particular, this will provide an overview of the coincidence and decidability results we achieved. Note that whenever hp and hhp bisimilarity coincide for a system class then (recalling that hp bisimilarity is decidable for finite-state systems) hhp bisimilarity is decidable

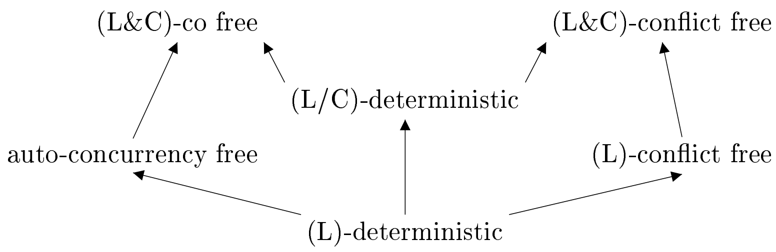


Figure 7.1: Classes with restricted nondeterminism

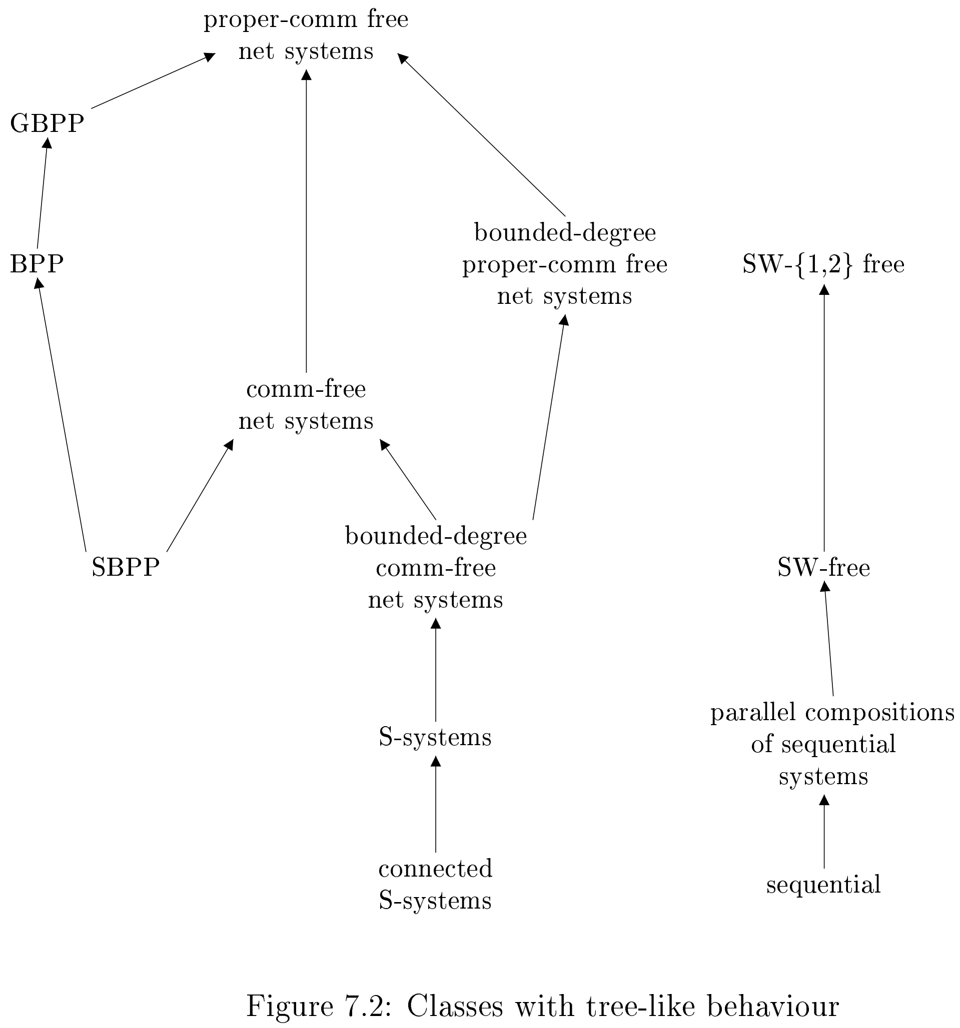


Figure 7.2: Classes with tree-like behaviour

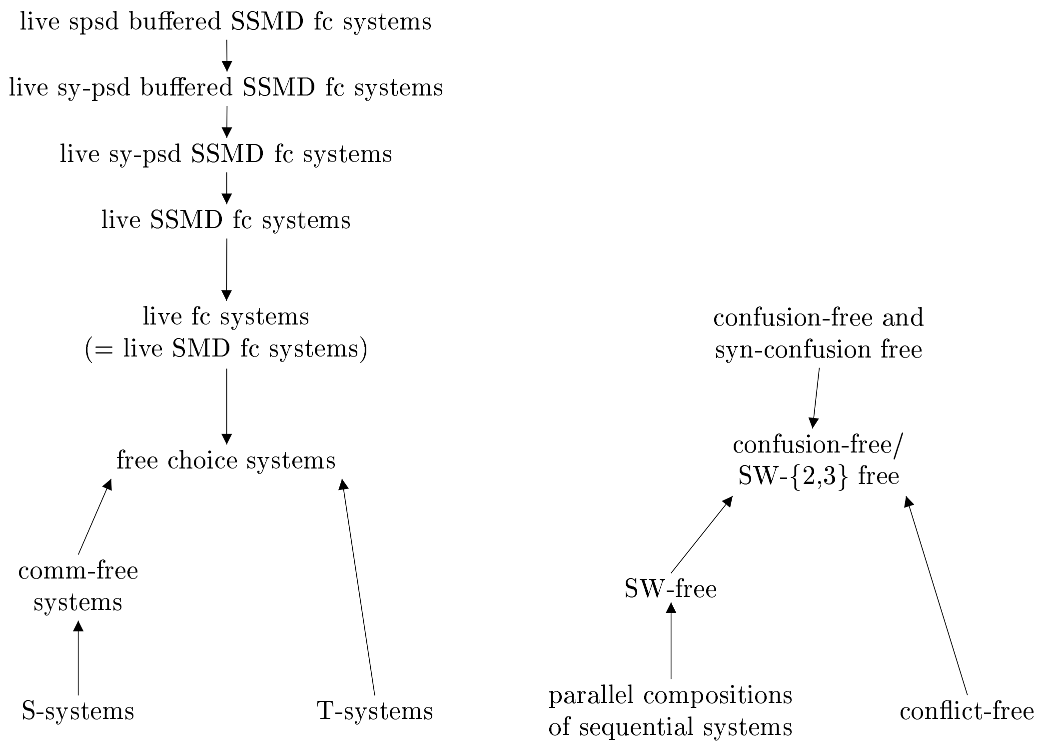


Figure 7.3: The free choice spectrum

for the finite-state fragment of that class. Also note that all the non-coincidence results already hold for finite-state systems. Figures 7.1 – 7.3 give a map of (most of) the system classes considered, illustrating their interconnections. Our results fall into five categories:

1. Restricted Backtracking. We constrained the hereditary condition in two different ways, which translated into two hierarchies of restricted backtracking bisimilarities: (n)hhp bisimilarity, and (n)nhhp bisimilarity for $n \in \mathbb{N}_0$. Via the decidability of (n)hhp bisimilarity we obtained decidability of hhp bisimilarity for two subclasses. For finite-state systems, and $n \in \mathbb{N}_0$ we have:

1. Restricted Backtracking	
Result	Sec.
(n) and (n+1)hhp bisimilarity do not coincide	3.3.2
(n)hhp bisimilarity is decidable	3.3.3
(n) and (n+1)nhp bisimilarity do not coincide	3.4.2
(n)nhp bisimilarity is decidable	3.4.3
hhp b. is decidable for bounded asynchronous systems	3.5.2
hhp b. is decidable for systems with transitive independence	3.5.3

The classes ‘bounded asynchronous systems’ and ‘systems with transitive independence' may seem disconnected from our remaining analysis of system classes; however, they provide important complementary insights on the role of synchronization for hhp bisimilarity. We will come back to this under Point 3 of the following section.

2. Basic Behavioural Classes. This group comprises the basic system classes we examined to establish a minimum of behavioural situations which must be allowed to keep hp and hhp bisimilarity distinct. We considered causality, concurrency, and conflict, and (L&C)-nondeterminism.

2. Basic Behavioural Classes			
Behavioural Class	Petri Net Class	C?	Sec.
sequential	connected S-systems	yes	4.2.1
conflict-free	T-systems	yes	4.2.1
causality-free		no	4.2.1
(L)-det. $\subset$ (L/C)-det.		yes	4.2.2
auto-concurrency free $\subset$ (L&C)-co free		no	4.2.3
(L)-conflict free $\subset$ (L&C)-conflict free		no	4.2.3

C? ... Coincidence between hp and hhp bisimilarity?

The most interesting result here is, perhaps, that (L&C)-nondeterminism due to conflict is not necessary for non-coincidence: hp and hhp bisimilarity do not coincide for (L)-conflict free systems. To prove this result we exhibited a new counter-example, counter-example 3, which is also important with respect to confusion-free and free choice systems.

3. Advanced Behavioural Classes. We then studied a group of more advanced behavioural classes: they admit all the situations identified to be necessary for non-coincidence; the classes also admit causality. In our analysis, we proved and built on the following composition and decomposition theory:

Composition and Decomposition	
Result	Sec.
(h)hp b. is composable w.r.t. decompositions into independent factors	4.3.2
for bsc-decomposable systems, (h)hp b. is decomposable w.r.t. the set of prime components	4.3.3

With this it was straightforward to obtain that hp and hhp bisimilarity coincide for parallel compositions of sequential systems, which confirmed that it is the mixture of concurrency with conflict and/or causality that stands behind the increased power of hhp bisimilarity. This led us to formulate and investigate the three SW-situations. Employing the composition and decomposition theory in an inductive argument we showed that the coincidence result carries over to bounded-degree SW-free systems.

SW-free systems exhibit tree-like behaviour in a strict sense, and thus two different directions to proceed emerged: (1) to stick to tree-like behaviour, but allow more interaction 'at one level'; intuitively, this amounts to investigating systems that exclude SW-1 and SW-2; (2) to allow 'proper' synchronization, in the sense of SW-1, but only in a controlled way, separated from conflict; intuitively, this amounts to excluding SW-2 and SW-3, or equivalently, the well-known situation of confusion.

For both SW-{1,2} free and confusion-free systems hp and hhp bisimilarity do not coincide. In the latter case this insight disproves a long-standing conjecture. Proof of non-coincidence was provided by counter-example 3; we also exhibited a more compact counter-example, counter-example 4. We defined the concept of syn-confusion to capture the kind of MNH situation these two counter-examples rely on. We suspect that, in the absence of confusion, liveness may play a role in excluding syn-confusion.

3. Advanced Behavioural Classes			
Behavioural Class	Petri Net Class	C?	Sec.
parallel compositions of sequential systems	S-systems	yes	4.3.4
bounded-degree SW-free	bounded-degree comm-free	yes	4.4.3
SW-{1,2} free	proper-comm free	no	4.4.2,5.4.2
SW-{2,3} free/confusion-free	free choice	no	4.4.2,4.5.2
SW-{1,3} free		no?	4.4.2
syn-confusion free		no	4.5.3
live		no[1]	4.6

C? ... Coincidence between hp and hhp bisimilarity?

[1] We also know: hhp b. is undecidable for live finite-state systems.

4. The BPP Spectrum. We addressed two standard classes of infinite-state verification: SBPP and BPP. Their partial order semantics are given in terms of comm-free net systems, and proper-comm free net systems respectively. Thus, by studying them we took further our investigation of classes with tree-like behaviour.

4. The BPP Spectrum			
Class	BC	Result	Sec.
SBPP	SW-free	hp and hhp b. coincide, and are decidable	5.3.2
BPP	SW-{1,2} free	hp and hhp b. do not coincide	5.4.2
		hp b. is decidable	5.4.3
		hp and distributed b. coincide	5.4.4
		hhp and chhp b. coincide, and are decidable	5.4.5

BC . . . Behavioural Characteristic

All the positive results follow by clear tableau decision procedures, which are based on composition and decomposition insights. For SBPP our theorems on (de)composition with respect to independent (prime) factors once more applied; for BPP specific decomposition views were developed. The decidability of hp bisimilarity and its coincidence with distributed bisimilarity is also induced via results on causal bisimilarity [Ace92a, KH94, Kie99]. All the results on hhp bisimilarity are new.

5. The Free Choice Spectrum. Finally, we studied classes of free choice systems, and thereby continued the theme of confusion (or SW-{2,3}): free choice systems allow synchronization but only in a controlled way, separated out from conflict. Having shown non-coincidence for the entire class, we concentrated on tackling live fc systems. On top of being confusion-free these appear to be syn-confusion free: they exclude all combinations of MNH and frame situations that have been employed in counter-examples so far. Yet, in comparison with the tree-like classes, live fc systems proved a very demanding class to tackle. Based on decomposition ideas we devised an approach that breaks the coincidence problem down into several subgoals, and thereby disentangles the difficulty. Crucial subgoals were established, and several coincidence and decidability results could be deduced for subclasses of live fc systems. For finite-state systems, we have:

5. The Free Choice Spectrum		
Class	Result	Sec.
free choice	hp and hhp b. do not coincide	6.3
live fc (= live SMD fc)	?	6
live SSMD fc	cp b. is K-decomposable	6.11
	cp b. is sw-(1)her & sw-(1)coh	6.11
live sy-psd SSMD fc	cp, hcp, and chcp b. coincide	6.13
live sy-psd buffered SSMD fc	chhp b. is decidable	6.14
live spsd buffered SSMD fc	hp, hhp, and chhp b. coincide	6.14

7.1.2 Main Conclusions and Future Work

Having summarized our results we now present several main conclusions and prognoses that we can deduce from this work. We also outline the most important directions for further research. The conclusions and prognoses apply to hhp bisimilarity, first of all; however, we expect they are relevant for true-concurrency investigations in general. In particular, they should apply to concepts that are truly-concurrent in that they, in some way, refer to the notion of event (cf. Section 1.2.1). This section can also be read in the context of Section 7.2.

1. The Dimensions of Past Operators. Past operators such as backtracking have two dimensions: (1) How far back in the history are we allowed to refer? (2) How often are we allowed to refer to the past? The full distinguishing and computational power of hhp bisimilarity can only be achieved by leaving these two dimensions unbounded.

This conclusion follows from our hierarchy results of Chapter 3. There are still gaps to close in our understanding of restricted backtracking. For example, one could study a variant of (n)nhp bisimilarity where Opponent is allowed to backtrack as many transitions 'in one block' as he likes. The counter-example of Section 3.3.2 is not strong enough to establish strictness for this hierarchy, but the counter-example derived from the undecidability proof (mentioned in Section 3.6) should still apply. In general, the relationship between (n)hhp and (n)nhp bisimilarity and their relation to the undecidability proof should be investigated in more detail. Thinking beyond hhp bisimilarity, the hierarchy ideas could be applied to approach other truly-concurrent concepts, e.g. CTL_P. Altogether, they could help to draw up a complete picture of the role past operators play in true-concurrency.

2. Composition and Decomposition. The idea of composition and decomposition is very natural for true-concurrency, and provides an important—if not *the*—technique to establish decidability (and tractability) results.

In contrast to the interleaving world, where decomposition must be considered with respect to syntax (e.g. with respect to the process algebra operator '||'), in true-concurrency decomposition has a semantic flavour: we can directly recognize whether a truly-concurrent system can be dissected into independent 'chunks' of behaviour. Accordingly, it may be possible to decide a truly-concurrent problem by a 'divide and conquer' approach: we have experienced throughout this thesis how decomposition characteristics of a system class can translate into (h)hp bisimilarity in a natural way, and thereby lead us to decidability and coincidence results. It will be interesting to see whether this approach can be taken beyond hhp bisimilarity (e.g. be used to tackle a truly-concurrent logic), and, orthogonally, beyond the system classes considered here. Regarding the last point, one would expect that the more complex the interplay of causality, concurrency, and conflict gets the more difficult it will be to obtain a fruitful (de)composition theory. This has already been experienced in this thesis when we moved from tree-like systems to systems with proper synchronization. In general, the idea of composition and decomposition seems important with respect to designing efficient algorithms, and it appears to be inherently related to partial order reduction.

3. Synchronization. (A) For system classes with tree-like behaviour truly-concurrent problems seem to be particularly natural and straightforward, both to tackle and to decide. (B) As soon as we admit synchronization (in the sense that SW-1 can occur) the interplay of causality, concurrency, and conflict is in general considerably more difficult, and so is the tackling of truly-concurrent problems. Systems with structurally controlled synchronization are still difficult to tackle, but, at least in the presence of liveness, they are probably within the decidability border. (C) If we assume that the systems under study are tightly synchronized (in that no thread can be left behind indefinitely) then true-concurrency seems to lose its computational power.

(A) is based on the experience that all the classes with tree-like behaviour we analysed were straightforward to handle via decomposition insights. We also achieved many coincidence results for this group, which indicates that true-concurrency brings about less subtlety here.

(B) In contrast, during our work on live fc systems we witnessed that admitting synchronization, even if it is structurally controlled and investigated in the presence of liveness, makes things tremendously more difficult. Our work also shows, though, that, building on structural exploits and decomposition insights, we are still able to expose regularities within truly-concurrent problems on such systems. This indicates that true-concurrency is within the decidability border here.

(C) is suggested by the decidability of hhp bisimilarity for (finite-state) bounded asynchronous systems. The prognosis is further supported by the proof of the undecidability of hhp bisimilarity in the general case [JN00]: one aspect of the proof, and—one could claim—of the power of true-concurrency in general, is that (finite-state) truly-concurrent systems can encode the $\omega \times \omega$ grid (cf. Section 7.2). Tightly synchronized systems do not have this expressive power.

It would be interesting to see whether the three rules of thumb can be confirmed with respect to other truly-concurrent problems, e.g. in logic. Indeed, we know of a case that goes against rule (A): model-checking the fixpoint version of the backtracking logic that characterizes hhp bisimilarity is already undecidable for finite-state comm-free net systems. This follows via a weak simulation of 2-counter machines, inspired by that of Jančar (cf. Section 1.3.2).[1] Is this a phenomenon particular to that logic, or is there a sense in which truly-concurrent logic problems can hide more irregularity than equivalence problems (relying less on the power of the interplay)?

4. The Infinite-State World. In the finite-state world truly-concurrent problems are typically harder than their interleaving counterparts, but in the infinite-state world this trend may be reversed. Standard infinite-state classes such as SBPP and BPP enjoy natural decomposition properties and good structural features which are particularly exploitable in true-concurrency.

This trend first emerged from the works [EK95] and [SN96], and could further be confirmed by our decidability and coincidence results on SBPP and BPP. In particular, the decidability of hhp bisimilarity on BPP shows that in the true-concurrency world an equivalence that is undecidable for finite-state systems can have a clear decision procedure for a standard class of infinite-state systems. The positive trend for true-concurrency is also motivated by the recent complexity results reported in [Las03] and [Jan03].

An important way to proceed is to investigate whether our results carry over to versions of SBPP and BPP with synchronization, such as BPP_τ. Are SBPP and BPP tractable because they have tree-like behaviour or is syntactically controlled synchronization still within the border of decidability? To clarify this point is particularly important in view of Paragraph (3) and (5): to complement our study of structurally controlled synchronization. Positive results for BPP_τ have been achieved for hp bisimilarity [KH94], distributed bisimilarity [Chr93], and pomset trace equivalence [SN96]; but hhp bisimilarity may well behave in a different way. In general, it will be interesting to see whether the positive trend for true-concurrency in infinite-state verification can further be substantiated, in particular with respect to logic.

[1] Many thanks to Julian Bradfield for pointing this out to me.

5. A Free Choice Hiatus? (A) With respect to truly-concurrent problems, such as deciding hhp bisimilarity, we speculate that live fc systems lie within the decidability border. It may even be the case that the interplay does not bring about much subtlety here. More specifically, chhp, hhp, and hp bisimilarity coincide for a subclass of live fc systems; chhp bisimilarity is decidable for a slightly less restricted class. (B) For the full free choice class the situation is less clear. (C) In general, free choice and live fc systems may prove difficult to tackle, but—on the positive side—they should still be approachable.

(A) It would be very satisfying to obtain a full understanding of how hhp bisimilarity behaves on live fc systems. Being able to build on Chapter 6 this should be comparatively straightforward now. We speculate that hhp bisimilarity is at least decidable here. If hhp bisimilarity turned out undecidable this would entail that the restrictions we imposed to bridge over the remaining subgoals highlighted situations that are significant with respect to the power of true-concurrency. Then, true-concurrency would seem very subtle indeed.

(B) A second task is to clarify the situation for the full free choice class. The fact that hhp and hp bisimilarity do not coincide for free choice systems indicates that the interplay is subtle here; but is it sufficiently subtle to induce undecidability? There are two contrasting intuitions: on the one hand, there is a sense in which free choice systems can simulate any mixture of causality, concurrency, and conflict by which transitions may be related; on the other hand, one could speculate that as long as the mixture is separated out in that the situation of confusion does not occur we do not obtain the same computational power. We will come back to this point in Section 7.2, where we report on some preliminary investigations on free choice systems.

(C) is witnessed by Chapter 6. We were able to approach live fc systems, but only by developing and employing various tools, which can be seen as the culmination of our previous insights and methods on true-concurrency. Although the techniques were designed with free choice systems and hhp bisimilarity in mind we hope that the principles will be relevant in general to tackle and dissect a problem of the true-concurrency world, maybe in logic or automatic synthesis. One would expect, though, that beyond free choice systems (the next obvious class to consider is *simple net systems*) things will be much, much more inscrutable. In that sense there certainly is a free choice hiatus.

7.1.3 Shortcomings and Future Work

Finally, we discuss some shortcomings in the work presented. In doing so further themes and prognoses for future work arise; a difficulty that applies to true-concurrency investigations in general is highlighted.

A Hierarchy of System Classes for True-Concurrency? In Section 1.1 we have motivated that a unified understanding of true-concurrency and its borderlines depends on being able to work relative to a well-established hierarchy of subclasses. We have taken care to choose our system classes as systematically as possible, but we have encountered certain pitfalls.

For most truly-concurrent problems (including hhp bisimilarity) models such as lats' or tsi's provide the primary semantic model: it is the behavioural level which will ultimately be relevant. However, one experience of our project is that this behavioural level is difficult to work with. Firstly, it seems, disentangling and classifying behaviour successfully is only possible up to a certain degree: to identify behavioural situations beyond the ones discussed in Chapter 4 seems not only technically awkward, but also prone to redundancy and ambiguity. Even at our basic level we experienced both: the two basic cases of confusion coincide with SW-2 and SW-3; there are many ways of defining a concept of syn-confusion. A related difficulty is that working with behavioural classes can be technically awkward: to prove something intuitively straightforward may turn out to be technically involved; an interesting proof may be drowned in technicalities. The first was experienced in showing that SW-free systems are CIS-decomposable (cf. Section 4.4.3); the second is exemplified by the proof of the decomposition result of Section 4.3.3. (Indeed, the issue of technial awkwardness may be related to difficulties experienced in work on characterizing regular event structures [NT02].)

Thus, it proved essential to move to models which maintain concepts of locality and distribution along with notions of structural interaction such as synchronization: to be able to refer to the structure that ultimately stands behind the complexity of the interplay. The problem with moving to such structural classes is that we sacrifice uniformity: there are various structural models, and various model-dependent restrictions have been employed (for example, it is also popular to work with distributed automata, or networks of agents connected via communication channels). Furthermore, working with structural subclasses will disguise which behavioural characteristics we ultimately rely on.

In view of a uniform theory of true-concurrency it is important to study the interplay between behaviour and structure; to establish a more general structure theory that encompasses the various structural models: to discern how structural features translate into aspects of behaviour; to understand the relationship between the different structural models and their restrictions—albeit, such connections will almost certainly be tedious to prove. It will surely be impossible to draw up hierarchies as formal and straightforward as those used in infinite-state verification and language theory. However, a map of informal connections and rules of thumb would in itself contribute much towards a uniform study of true-concurrency.

The Role of Nondeterminism. Throughout our analysis we have focused our attention on the interplay of causality, concurrency, and conflict. On the other hand, from Section 4.2.2 (and Sections 6.13, 6.14.3) we know: the distinguishing and computational power of hhp bisimilarity also depends on the amount of nondeterminism available in the system class under study. Therefore, to complete our understanding of hhp bisimilarity, and the power of true-concurrency in general, it seems important to systematically analyse the role of nondeterminism and its interaction with causality, concurrency, and conflict. An important step into this direction has already been undertaken: [Muk02] investigates the class of (finite-state) *trace-labelled systems*, and finds that hhp bisimilarity is decidable here.

Complexity. One obvious shortcoming is that we did not consider the aspect of complexity in this work. It should be interesting to analyse the complexities of our tableau-based decision procedures for SBPP and BPP, and compare them to the polynomial-time result that has been obtained for hp bisimilarity on BPP in [Las03]. Otherwise, with respect to finite-state systems, our decidability results typically follow from having shown that hhp bisimilarity coincides with hp bisimilarity for the respective system class. In this case, we can carry over upper bounds of DEXPTIME. For the algorithms that decide (n)hhp and (n)nhp bisimilarity, and conseqently those that decide hhp bisimilarity on bounded asynchronous systems and systems with transitive independence, the complexities may well be higher.

One could argue that the coincidence results provide a basis for the construction of more efficient algorithms for hp (and hhp) bisimilarity: the coincidence proofs uncover regularities within hp bisimilarity; consequently, the expensive gsc state space exploration may be avoidable. In general, there is a sense in which hhp bisimilarity is inherently amenable for partial order reduction: we only need to consider those hp bisimulations that are trace-consistent. In this spirit, coincidence results may be of practical use. In summary, we say:

> If hhp bisimilarity is decidable for a subclass then it may well be efficiently decidable due to structural exploits and inherent partial order reduction. If hhp bisimilarity coincides with hp bisimilarity for a subclass, this may lead us to efficient algorithms for checking the bisimilarities on that class.

Projecting to true-concurrency in general, and integrating what has been indicated in Section 1.2.2 following [KV01], we further suggest:

> Although, in general, truly-concurrent problems can be hard, even undecidable, we can hope to achieve efficient algorithms for subclasses by exploiting the higher structure of true-concurrency. If a truly-concurrent concept is found to be of high complexity one has to keep in mind that this may be attributed to the fact that, in general, a truly-concurrent measure may amount to an exponential compression of the corresponding interleaving state space.

7.2 The Undecidability of hhp Bisimilarity

We now discuss the undecidability of hhp bisimilarity, which has been established by Jurdziński and Nielsen in [JN00].

The undecidability proof proceeds in two steps: (1) The intermediate problem of checking *domino bisimilarity for origin constrained tiling systems* is introduced, and shown to be undecidable by a reduction from the halting problem of 2-counter machines. (2) Checking domino bisimilarity for origin constrained tiling systems is then in turn reduced to checking hhp bisimilarity on finite-state lats'. It is also shown that this reduction, and thus the undecidability result, can be strengthened to hhp bisimilarity on finite 1-safe Petri nets, which can be seen as a proper subclass of finite-state lats'.

The following two works helped pave the way for the undecidability proof. In [JN99] Jurdziński and Nielsen prove the undecidability of hhp *simulation* in two analogous steps, proceeding via the intermediate problem of determining the winner in *domino snake games*. In [MT98] Madhusudan and Thiagarajan address the problem of synthesizing controllers for discrete event systems; they show that in a truly-concurrent setting this problem is undecidable. The proof proceeds by a reduction from the problem of tiling the $\omega \times \omega$ grid, and exhibits an important technique: it is shown how a tiling system on the $\omega \times \omega$ grid can be encoded in a finite-state lats. A modified version of this 'gadget' is employed in the reductions from the domino tiling problems to hhp bisimulation and simulation. Tiling problems also play a role in truly-concurrent logic: CTL_P is shown undecidable by a reduction from the *recurring tiling problem* [PK95].

Altogether, this demonstrates that there is a fruitful connection between true-concurrency and the theory of tiling problems. This connection is not only profitable for concurrency theory but it feeds both ways: the two domino tiling problems, introduced in the context of hhp bisimulation and simulation, present an interesting addition to the 'toolbox' of undecidable problems: "the combinatorical and geometrical simplicity of domino problems renders them an ideal medium for [...] proving "bad behaviour" such as NP-hardness or undecidability" [Har85].

Since tiling systems are strong enough to encode Turing machines or 2-counter machines in a relatively straightforward way, the tiling connection mediates that there is a sense in which true-concurrency is fundamentally hard: in general, when considered at the level of their unfolding structure, finite-state true-concurrency models, such as lats', are almost Turing powerful. This has to be kept in mind whenever one tackles a truly-concurrent problem.

The insight that finite-state concurrent systems have the power to encode tiling systems involves two aspects: (1) There is a class of concurrent systems whose structure corresponds to the two-dimensional grid with the addition that the nodes are decorated with an encoding of a tiling specification; altogether there is a natural notion of tiling associated with systems of this class: we can faithfully mimic the building of a domino snake by a combination of forwards and backtracking moves. (2) Such systems can be folded into *finite-state* systems. It seems the basic situations causality, concurrency, and conflict can interact in a complicated way such that, in general, we cannot cover all the situations the unfolding will create by looking at a finite portion of it; the interplay must somehow cause a loss of regularity.

Crucial to an understanding of true-concurrency is then to further explain and disentangle the computational power of concurrent systems: to analyse which aspects of the interplay make true-concurrency so powerful. Naturally, this directly connects to this thesis: we have approximated the borderline from below by gaining positive results for hhp bisimilarity on classes with restricted behaviour. In a complementary approach we could approximate the borderline from above by investigating whether the tiling encoding is still possible for systems with restricted interplay. Live free choice systems seem too restricted to allow for a simulation of the encoding; we speculate that for this class hhp bisimilarity is decidable, and possibly even coincides with hp bisimilarity. However, free choice systems make an interesting candidate. On the one hand, they can simulate any mixture of causality, concurrency, and conflict by which transitions may be related: the respective transitions may only have to be separated out by other transitions to ensure that conflict and synchronization are kept apart. On the other hand, confusion may well be necessary to guarantee that the tiling movement can be simulated faithfully.

Some preliminary investigations have already been undertaken: the idea is to simulate the net systems used in the undecidability proof by free choice systems via a method of inserting dummy transitions so as to separate out conflict and synchronization. The question is whether the simulation can be done in a sufficiently faithful way so that a domino bisimulation will still give rise to a hhp bisimulation. At the moment it is not clear to me whether this is possible. The closest I could get was to exhibit a simulation that seems strong enough to allow a reduction from a strengthened version of domino bisimulation, so-called *bijective* domino bisimulation. The reduction from the halting problem of 2-counter machines to domino bisimulation relies on non-bijectiveness in an essential way; so the bijective version may well be decidable. One could speculate that the construction cannot be stretched any further: while many instances of confusion can be avoided it seems there is one particular situation that makes the move to bijective domino bisimulation unavoidable. Isolating this situation may bring us a step closer to understanding the power of true-concurrency.

7.3 General Outlook and Application

Where to go from here? In the last two sections we have highlighted how one could proceed to further explain and disentangle the hardness of true-concurrency. To obtain a fundamental understanding of true-concurrency it also seems crucial to strengthen the already existing links with trace theory and the field of tiling problems. Another important, complementary, direction to pursue is to back up what has been put forward in the introduction: that an analysis of hhp bisimilarity, and, more generally, investigations into the hardness of true-concurrency, should have practical benefits with respect to the automatic verification of concurrent systems. We hope the following points will further motivate this postulate.

Clearly, there should be direct exploits in areas where true-concurrency is a primary notion, e.g. in automatic synthesis. Indeed, we have already seen a connection: the undecidability proof of hhp bisimilarity employs a technique that has been developed in the context of automatic synthesis. In turn, one would expect that insights obtained with hhp bisimilarity in mind might feed towards this practically relevant topic. Above we suggested to complement the work on hhp bisimilarity by analysing truly-concurrent logic such as CTL_P; results in this area should be of direct interest to automatic synthesis: the higher expressiveness of true-concurrency logics is crucial to specify properties about the internal structure of a system, e.g. to express that two activities must be independent of each other. So far, in most work on automatic synthesis, the input specification is made up of two parts: a formula expressed in one of the classical logics, and some information about the architecture of the system to be synthesized. Since via truly-concurrent logics finer issues about the architecture could be expressed they could play an important role to ward off unwanted solutions. As [KV01] suggests "the real challenge that synthesis algorithms and tools face in the coming years is mostly [...] that of making automatically synthesized systems more practically useful."

Hopefully our insights into true-concurrency can also be employed to develop more efficient model checking techniques. One idea is to extend the partial order reduction methods. Simplified, they exploit the following insight: if two independent transitions t_1 and t_2 are both executed at a state s, then the resulting state s' is independent of the order in which t_1 and t_2 are interleaved; if, moreover, t_1 has no indirect influence over t_2 in that the state reached after executing only t_2 does not give rise to any behaviour that is not also visible at s' then t_2 can safely be pruned. There are connotations to the coincidence problem: investigating the difference between hp and hhp bisimilarity led us to analysing in which ways a transition can influence the behavioural environment of a parallel transition. A close understanding of this aspect may allow us to identify situations when it is possible to prune t_2 even if t_1 has indirect influence over t_2; namely if we know that, overall, this influence will not lead to a loss of information. A second, related, idea is to construct model checking algorithms of low complexity for system classes with restricted behaviour; working with truly-concurrent models will automatically integrate a degree of partial order reduction.

When one achieves positive results for classes with restricted behaviour, it is important to analyse whether they are useful in practice. Live free choice systems are a promising candidate for efficient techniques. The free choice constraint can usually be achieved by suitable abstractions; in contrast, liveness seems more difficult to obtain. There are, however, two areas, where liveness is natural: asynchronous circuit design and workflow analysis in business process modelling.[2] In these areas the analysability of live free choice systems has already been exploited (e.g. [Esp03]). Two more areas seem relevant: medical electronics, and computer integrated manufacturing (CIM). They have a high potential for the use of formal methods: medical electronics are safety-critical; failures in CIM are expensive. Importantly, it appears that systems in these areas will naturally satisfy liveness.

[2] Many thanks to Javier Esparza for providing this information on free choice nets.

Bibliography

[ÀBGS91] C. Àlvarez, J. L. Balcázar, J. Gabarró, and M. Sántha. Parallel complexity in the design and analysis of concurrent systems. In *PARLE '91: Parallel architectures and languages Europe, Vol. I*, volume 505 of *Lecture Notes in Comput. Sci.*, pages 288–303. Springer, 1991.

[Ace92a] L. Aceto. History preserving, causal and mixed-ordering equivalence over stable event structures. *Fundamenta Informaticae*, 17(4):319–331, 1992.

[Ace92b] L. Aceto. Relating distributed, temporal and causal observations of simple processes. *Fundamenta Informaticae*, 17(4):369–397, 1992.

[AY99] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *CONCUR'99: concurrency theory*, volume 1664 of *Lecture Notes in Comput. Sci.*, pages 114–129. Springer, 1999.

[BBK87] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *PARLE'87: Parallel architectures and languages Europe, Vol. II*, volume 259 of *Lecture Notes in Comput. Sci.*, pages 94–111. Springer, 1987.

[BBK93] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the Association for Computing Machinery*, 40(3):653–682, 1993.

[BC87] G. Boudol and I. Castellani. On the semantics of concurrency: partial orders and transition systems. In *Theory and Practice of Software Development (1987), Vol. 1*, volume 249 of *Lecture Notes in Comput. Sci.*, pages 123–137. Springer, 1987.

[BCHK93] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. Observing localities. *Theoretical Computer Science*, 114(1):31–61, 1993. Workshop on Concurrency and Compositionality (1991).

[BCHK94] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. *Formal Aspects of Computing*, 6:165–200, 1994.

[BCMS01] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In *Handbook of process algebra*, pages 545–623. North-Holland, 2001.

[BDKP91] E. Best, R. Devillers, A. Kiehn, and L. Pomello. Concurrent bisimulations in Petri nets. *Acta Informatica*, 28(3):231–264, 1991.

[Bed91] M. Bednarczyk. Hereditary history preserving bisimulation or what is the power of the future perfect in program logics. Technical report, Polish Academy of Sciences, Gdansk, 1991.

[Bes87] E. Best. Structure theory of Petri nets: the free choice hiatus. In *Advances in Petri nets 1986, Part I*, volume 254 of *Lecture Notes in Comput. Sci.*, pages 168–205. Springer, 1987.

[BGV91] W. Brauer, R. Gold, and W. Vogler. A survey of behaviour and equivalence preserving refinements of Petri nets. In *Advances in Petri nets 1990*, volume 483 of *Lecture Notes in Comput. Sci.*, pages 1–46. Springer, 1991.

[BHPS61] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Z. Phonetik Sprachwiss. Kommunikat.*, 14:143–172, 1961.

[Bra] J. C. Bradfield. Fixpoints and causality. Proposal for EPSRC Advanced Fellowship.

[BT87] E. Best and P. S. Thiagarajan. Some classes of live and safe Petri nets. *Concurrency and nets: Advances in Petri nets*, pages 71–94, 1987.

[Cas88] I. Castellani. *Bisimulations for Concurrency.* PhD thesis, University of Edinburgh, 1988.

[Cas01] I. Castellani. Process algebras with localities. In *Handbook of process algebra*, pages 945–1045. North-Holland, 2001.

[CDMP87] L. Castellano, G. De Michelis, and L. Pomello. Concurrency vs interleaving: an instructive example. *Bulletin of the EATCS*, 31:12–15, 1987.

[CE82] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of programs (1981)*, volume 131 of *Lecture Notes in Comput. Sci.*, pages 52–71. Springer, 1982.

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* The MIT Press, 1999.

[CH89] I. Castellani and M. Hennessy. Distributed bisimulations. *Journal of the Association for Computing Machinery*, 36(4):887–911, 1989.

[Che96] A. Cheng. *Reasoning About Concurrent Computational Systems.* PhD thesis, BRICS, University of Aarhus, Denmark, 1996.

[CHEP71] F. Commoner, A. W. Holt, S. Even, and A. Pnueli. Marked directed graphs. *J. Comput. System Sci.*, 5:511–523, 1971.

[CHM93a] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In *CONCUR '93: concurrency theory*, volume 715 of *Lecture Notes in Comput. Sci.*, pages 143–157. Springer, 1993.

[CHM93b] S. Christensen, Y. Hirshfeld, and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *IEEE Symposium on Logic in Computer Science (1993)*, pages 386–396. IEEE, 1993.

[Chr92] S. Christensen. Distributed bisimilarity is decidable for a class of infinite state-space systems. In *CONCUR '92: concurrency theory*, volume 630 of *Lecture Notes in Comput. Sci.*, pages 148–161. Springer, 1992.

[Chr93] S. Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, University of Edinburgh, 1993.

[CHS92] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In *CONCUR '92: concurrency theory*, volume 630 of *Lecture Notes in Comput. Sci.*, pages 138–147. Springer, 1992.

[CHS95] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121(2):143–148, 1995.

[CN95] A. Cheng and M. Nielsen. Open maps (at) work. Research Series RS-95-23, BRICS, Department of Computer Science, University of Aarhus, 1995.

[Com72] F. Commoner. Deadlocks in Petri nets. Technical Report CA-7206-2311, Applied Data Inc., 1972.

[DD89] Ph. Darondeau and P. Degano. Causal trees. In *Automata, languages and programming (1989)*, volume 372 of *Lecture Notes in Comput. Sci.*, pages 234–248. Springer, 1989.

[DD90] Ph. Darondeau and P. Degano. Causal trees: interleaving + causality. In *Semantics of systems of concurrent processes (1990)*, volume 469 of *Lecture Notes in Comput. Sci.*, pages 239–255. Springer, 1990.

[DDM87] P. Degano, R. DeNicola, and U. Montanari. Observational equivalences for concurrency models. In *Formal Description of Programming Concepts III, IFIP WG 2.2 working conference (1986)*, volume 259 of *Lecture Notes in Comput. Sci.*, pages 105–129. Springer, 1987.

[DDNM89] P. Degano, R. De Nicola, and U. Montanari. Partial orderings descriptions and observations of nondeterministic concurrent processes. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency (1988)*, volume 354 of *Lecture Notes in Comput. Sci.*, pages 438–466. Springer, 1989.

[DE95] J. Desel and J. Esparza. *Free choice Petri nets*. Cambridge University Press, 1995.

[DM97] V. Diekert and Y. Métivier. Partial commutation and traces. In *Handbook of formal languages, Vol. 3*, pages 457–533. Springer, 1997.

[DNMV90] R. De Nicola, U. Montanari, and F. Vaandrager. Back and forth bisimulations. In *CONCUR' 90: concurrency theory*, volume 458 of *Lecture Notes in Comput. Sci.*, pages 152–165. Springer, 1990.

[EK95] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and basic parallel processes. In *Computer Aided Verification (1995)*, volume 939 of *Lecture Notes in Comput. Sci.*, pages 353–366. Springer, 1995.

[EN94] J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Journal of Informatik Processing and Cybernetics*, 30(3):143–160, 1994.

[Eng91] J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6):575–591, 1991.

[ES91] J. Esparza and M. Silva. Circuits, handles, bridges and nets. In *Advances in Petri nets 1990*, volume 483 of *Lecture Notes in Comput. Sci*, pages 210–242. Springer, 1991.

[Esp97a] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.

[Esp97b] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31(1):13–25, 1997.

[Esp98] J. Esparza. Decidability and complexity of Petri net problems – an introduction. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Comput. Sci.*, pages 374–428. Springer, 1998.

[Esp03] J. Esparza. A polynomial-time algorithm for checking consistency of free-choice signal transition graphs. In *ACSD'03: Application of Concurrency to System Design*, pages 61–70. IEEE, 2003.

[FH99] S. Fröschle and T. Hildebrandt. On plain and hereditary history-preserving bisimulation. In *Mathematical foundations of computer science (1999)*, volume 1672 of *Lecture Notes in Comput. Sci.*, pages 354–365. Springer, 1999.

[Frö98] S. Fröschle. On the decidability problem of strong history-preserving bisimilarity. Progress Report, September 1998.

[Frö99] S. Fröschle. Decidability of plain and hereditary history-preserving bisimulation for BPP. In *EXPRESS'99: Workshop on Expressiveness in Concurrency*, volume 27 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1999.

[Frö00a] S. Fröschle. Proof of Theorem 6.6.1 (computations of components in the context of a course). Unpublished Notes, 2000.

[Frö00b] S. Fröschle. Restricting choice. Unpublished Draft, 2000.

[Frö03] S. Fröschle. History preserving bisimilarity via open maps. Manuscript, 2003.

[GH94] J. F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):354–371, 1994.

[GL73] H. J. Genrich and K. Lautenbach. Synchronisationsgraphen. *Acta Informatica*, 2:143–161, 1973.

[Hac72] M. H. T. Hack. Analysis of production schemata by Petri nets. Master's thesis, MIT, Dept. Electrical Engineering, Cambridge, Mass., 1972.

[Har85] D. Harel. Recurring dominoes: making the highly undecidable highly understandable. *Annals of Discrete Mathematics*, 24(24):51–72, 1985.

[Hen88] M. Hennessy. Axiomatising finite concurrent processes. *SIAM Journal on Computing*, 17(5):997–1017, 1988.

[Hir94] Y. Hirshfeld. Petri nets and the equivalence problem. In *Computer science logic (1993)*, volume 832 of *Lecture Notes in Comput. Sci.*, pages 165–174. Springer, 1994.

[HJM96] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed basic parallel processes. *Mathematical Structures in Computer Science*, 6(3):251–259, 1996.

[HM96] Y. Hirshfeld and F. Moller. Decidability results in automata and process theory. In *Logics for concurrency*, volume 1043 of *Lecture Notes in Comput. Sci.*, pages 102–148. Springer, 1996.

[HT95] D. T. Huỳnh and L. Tian. On deciding readiness and failure equivalences for processes. *Information and Computation*, 117(2):193–205, 1995.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 1979.

[Hüt94] H. Hüttel. Undecidable equivalences for basic parallel processes. In *Theoretical aspects of computer software (1994)*, volume 789 of *Lecture Notes in Comput. Sci.*, pages 454–464. Springer, 1994.

[Jan95] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995. STACS'94.

[Jan03] P. Jančar. Strong bisimilarity on basic parallel processes is PSPACE-complete. In *IEEE Symposium on Logic in Computer Science (2003)*, pages 216–??. IEEE, 2003.

[JLL77] N. D. Jones, L. H. Landweber, and Y. E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4(3):277–299, 1977.

[JM96] L. Jategaonkar and A. R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154(1):107–143, 1996. ICALP'93.

[JN99] M. Jurdziński and M. Nielsen. Hereditary history preserving simulation is undecidable. Research Series RS-99-1, BRICS, Department of Computer Science, University of Aarhus, 1999.

[JN00] M. Jurdziński and M. Nielsen. Hereditary history preserving bisimilarity is undecidable. In *Theoretical Aspects of Computer Science (2000)*, volume 1770 of *Lecture Notes in Comput. Sci.*, pages 358–369. Springer, 2000.

[JNW96] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

[KH94] A. Kiehn and M. Hennessy. On the decidability of non-interleaving process equivalences. In *CONCUR '94: concurrency theory*, volume 836 of *Lecture Notes in Comput. Sci.*, pages 18–33. Springer, 1994.

[Kie94] A. Kiehn. Comparing locality and causality based equivalences. *Acta Informatica*, 31(8):697–718, 1994.

[Kie99] A. Kiehn. A note on distributed bisimulations. Manuscript, June 1999.

[KS90] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[KV01] O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *IEEE Symposium on Logic in Computer Science (2001)*, pages 16–19. IEEE, 2001.

[Lam86] L. Lamport. On interprocess communication. *Distributed Computing*, 1(2):77–101, 1986.

[Las03] S. Lasota. A polynomial-time algorithm for deciding true concurrency equivalences of basic parallel processes. In *Mathematical Foundations of Computer Science (2003)*, volume 2747 of *Lecture Notes in Comput. Sci.*, pages 521–530. Springer, 2003.

[LS95] F. Laroussinie and Ph. Schnoebelen. A hierarchy of temporal logics with past. *Theoretical Computer Science*, 148(2):303–324, 1995. STACS'94.

[May84] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.

[May98] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems.* PhD thesis, TU Munich, 1998.

[Maz89] A. Mazurkiewicz. Basic notions of trace theory. In *Linear time, branching time and partial order in logics and models for concurrency (1988)*, volume 354 of *Lecture Notes in Comput. Sci.*, pages 285–363. Springer, 1989.

[Mil80] R. Milner. *A calculus of communicating systems*, volume 92 of *Lecture Notes in Computer Science.* Springer, 1980.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[MM93] R. Milner and F. Moller. Unique decomposition of processes. *Theoretical Computer Science*, 107(2):357–363, 1993.

[Mol96] F. Moller. Infinite results. In *CONCUR '96: concurrency theory*, volume 1119 of *Lecture Notes in Comput. Sci.*, pages 195–216. Springer, 1996.

[Moo56] E. F. Moore. Gedanken-experiments on sequential machines. In *Automata studies*, Annals of mathematics studies, no. 34, pages 129–153. Princeton University Press, 1956.

[MP97] U. Montanari and M. Pistore. Minimal transition systems for history-preserving bisimulation. In *Theoretical Aspects of Computer Science (1997)*, volume 1200 of *Lecture Notes in Comput. Sci.*, pages 413–425. Springer, 1997.

[MP00] A. Muscholl and D. Peled. Analyzing message sequence charts. In *2nd Workshop of the SDL Forum Society on SDL and MSC*, 2000. Invited Talk.

[MT98] P. Madhusudan and P. S. Thiagarajan. Controllers for discrete event systems via morphisms. In *CONCUR'98: concurrency theory*, volume 1466 of *Lecture Notes in Comput. Sci.*, pages 18–33. Springer, 1998.

[MT01] P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In *Automata, languages and programming (2001)*, volume 2076 of *Lecture Notes in Comput. Sci.*, pages 396–407. Springer, 2001.

[MT02] P. Madhusudan and P. S. Thiagarajan. A decidable class of asynchronous distributed controllers. In *CONCUR'02: concurrency theory*, volume 2421 of *Lecture Notes in Comput. Sci.*, pages 145–160. Springer, 2002.

[Muk02] M. Mukund. Hereditary history preserving bisimulation is decidable for trace-labelled systems. In *Foundations of Software Technology and Theoretical Computer Science*, volume 2556 of *Lecture Notes in Comput. Sci.*, pages 289–300. Springer, 2002.

[MW82] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. In *Logics of programs (1981)*, volume 131 of *Lecture Notes in Comput. Sci.*, pages 253–281. Springer, 1982.

[NC94] M. Nielsen and C. Clausen. Bisimulation, games, and logic. In *Results and trends in theoretical computer science (1994)*, volume 812 of *Lecture Notes in Comput. Sci.*, pages 289–306. Springer, 1994.

[Nor86] C. W. Norman. *Undergraduate Algebra*. Oxford Science Publications, 1986.

[NT02] M. Nielsen and P. S. Thiagarajan. Regular event structures and finite Petri Nets: The conflict-free case. In *Applications and Theory of Petri nets (2002)*, volume 2360 of *Lecture Notes in Comput. Sci.*, pages 335–351. Springer, 2002.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science: 5th GI-Conference (1981)*, volume 104 of *Lecture Notes in Comput. Sci.*, pages 167–183. Springer, 1981.

[Pen92] W. Penczek. On undecidability of propositional temporal logics on trace systems. *Information Processing Letters*, 43(3):147–153, 1992.

[Pen93] W. Penczek. Temporal logics for trace systems: on automated verification. *International Journal of Foundations of Computer Science*, 4(1):31–67, 1993.

[Pet81] J. L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.

[Pin97] J. E. Pin. Syntactic semigroups. In *Handbook of formal languages, Vol. 1*, pages 680–746. Springer, 1997.

[Pis99] M. Pistore. *History Dependent Automata*. PhD thesis, Computer Science Department, University of Pisa, 1999.

[PK95] W. Penczek and R. Kuiper. Traces and logic. In *The Book of Traces*, pages 307–381. World Scientific, 1995.

[PP90] D. Peled and A. Pnueli. Proving partial order liveness properties. In *Automata, languages and programming (1990)*, volume 443 of *Lecture Notes in Comput. Sci.*, pages 553–571. Springer, 1990.

[PPH97a] D. Peled, V. Pratt, and G. Holzmann, editors. *Partial order methods in verification (1996)*, volume 29 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages xi–xiv (Preface). American Mathematical Society, 1997.

[PPH97b] D. Peled, V. Pratt, and G. Holzmann, editors. *Partial order methods in verification (1996)*, volume 29 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 359–403 (Debate'90: An Electronic Discussion on True Concurrency). American Mathematical Society, 1997.

[PR90] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *IEEE Symposium on Foundations of Computer Science (1990)*, pages 746–757. IEEE, 1990.

[Pra86] V. Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[Rab97] A. Rabinovich. Complexity of equivalence problems for concurrent systems of finite agents. *Information and Computation*, 139(2):111–129, 1997.

[RE98] G. Rozenberg and J. Engelfriet. Elementary net systems. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Comput. Sci.*, pages 12–121. Springer, 1998.

[Rei82] W. Reisig. *Petri Nets*. Springer, 1982.

[RT86] G. Rozenberg and P. S. Thiagarajan. Petri nets: basic notions, structure, behaviour. In *Current trends in concurrency (1985)*, volume 224 of *Lecture Notes in Comput. Sci.*, pages 585–668. Springer, 1986.

[RT88] A. Rabinovich and B. A. Trakhtenbrot. Behavior structures and nets. *Fundamenta Informaticae*, 11(4):357–403, 1988.

[ŞEM03] A. Ştefănescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. In *CONCUR'03: concurrency theory*, volume 2761 of *Lecture Notes in Comput. Sci.*, pages 27–41. Springer, 2003.

[Sén97] G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *Automata, languages and programming (1997)*, volume 1256 of *Lecture Notes in Comput. Sci.*, pages 671–681. Springer, 1997.

[Sén01] G. Sénizergues. $L(A) = L(B)$? Decidability results from complete formal systems. *Theoretical Computer Science*, 251(1-2):1–166, 2001.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: preliminary report. In *ACM Symposium on Theory of Computing (1973)*, pages 1–9. Association for Computing Machinery, 1973.

[SN96] K. Sunesen and M. Nielsen. Behavioural equivalence for infinite systems—partially decidable! In *Application and theory of Petri nets (1996)*, volume 1091 of *Lecture Notes in Comput. Sci.*, pages 460–479. Springer, 1996.

[Sti96] C. Stirling. Modal and temporal logics for processes. In *Logics for Concurrency*, volume 1043 of *Lecture Notes in Comput. Sci.*, pages 149–237. Springer, 1996.

[Sti01] C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, 255(1-2):1–31, 2001.

[Sti02] C. Stirling. Deciding DPDA equivalence is primitive recursive. In *Automata, languages and programming (2002)*, volume 2380 of *Lecture Notes in Comput. Sci.*, pages 821–832. Springer, 2002.

[Tau89] D. Taubner. *Finite representations of CCS and TCSP programs by automata and Petri nets*, volume 369 of *Lecture Notes in Computer Science*. Springer, 1989.

[Tru91] J. K. Truss. *Discrete Mathematics for Computer Scientists*. Addison-Wesley, 1991.

[TV84] P. S. Thiagarajan and K. Voss. A fresh look at free choice nets. *Information and Control*, 61(2):85–113, 1984.

[TW02] P. S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. *Information and Computation*, 179(2):230–249, 2002. LICS'97.

[vG01] R. J. van Glabbeek. The linear time–branching time spectrum. I. The semantics of concrete, sequential processes. In *Handbook of process algebra*, pages 3–99. North-Holland, 2001.

[vGG89a] R. J. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions. In *Mathematical foundations of computer science (1989)*, volume 379 of *Lecture Notes in Comput. Sci.*, pages 237–248. Springer, 1989.

[vGG89b] R. J. van Glabbeek and U. Goltz. Partial order semantics for refinement of actions — neither necessary nor always sufficient but appropriate when used with care. *Bulletin of the EATCS*, 38:154–163, 1989.

[vGG01] R. J. van Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37(4-5):229–327, 2001.

[vGV87] R. J. van Glabbeek and F. W. Vaandrager. Petri net models for algebraic theories of concurrency. In *PARLE'87: Parallel architectures and languages Europe, Vol. II*, volume 259 of *Lecture Notes in Comput. Sci.*, pages 224–242. Springer, 1987.

[Vog91] W. Vogler. Deciding history preserving bisimilarity. In *Automata, languages and programming (1991)*, volume 510 of *Lecture Notes in Comput. Sci.*, pages 495–505. Springer, 1991.

[Vog95] W. Vogler. Generalized OM-bisimulation. *Information and Computation*, 118(1):38–47, 1995.

[Wal98] I. Walukiewicz. Difficult configurations — on the complexity of LTrL. In *Automata, Languages and Programming (1998)*, volume 1443 of *Lecture Notes in Comput. Sci.*, pages 140–151. Springer, 1998.

[Win89] G. Winskel. An introduction to event structures. In *Linear time, branching time and partial order in logics and models for concurrency (1988)*, volume 354 of *Lecture Notes in Comput. Sci.*, pages 364–397. Springer, 1989.

[WN97] G. Winskel and M. Nielsen. Categories in concurrency. In *Semantics and logics of computation (1995)*, pages 299–354. Cambridge University Press, 1997.

Appendix A

Relating to Chapter 3

Here we will give the detailed proof of the decidability of hhp bisimilarity for the full class of finite-state systems with transitive independence relation. As described in Section 3.5.3 the essence of the proof is the observation that concurrently occurring self-loops have always to be matched with self-loops. We will first give the precise definition of what it means for a self-loop to occur concurrently, and then formulate and prove the corresponding lemma.

Definition A.0.1. Assume a given system S. Let t be a self-loop transition of S, and let r be some run of S. We say the self-loop t is *concurrently occurring at* r iff

- t is enabled at r, and
- there exists t', s. t. $t\ I\ t'$ and we have $r \xrightarrow{t'} r.t'$ or $BEn(r, t')$.

Lemma A.0.1. *Let $\mathcal{H}$ be a hp bisimulation relating two systems with transitive independence relation, S_1, S_2.*

- *Whenever $(r_1.t_1, r_2.t_2) \in \mathcal{H}$, and t_1 is a concurrently occurring self-loop at r_1, then t_2 is a self-loop as well.*
- *Vice versa.*

Proof. To prove the first part of the lemma let $(r_1.t_1, r_2.t_2) \in \mathcal{H}$ and let t_1 be a concurrently occurring self-loop at r_1. First assume we have $t'_1\ I\ t_1$, such that $r_1 \xrightarrow{t'_1} r_1.t'_1$. Clearly we have $(r_1.t_1.t_1, r_2.t_2.t^*_2) \in \mathcal{H}$ for some $t^*_2\ D\ t_2$, and $(r_1.t_1.t_1.t'_1, r_2.t_2.t^*_2.t'_2) \in \mathcal{H}$ for some t'_2, s. t. $t'_2\ I\ t_2$ and $t'_2\ I\ t^*_2$. With transitivity of independence the latter leads to a contradiction with the requirement $t^*_2\ D\ t_2$, unless $t^*_2 = t_2$. But if $t^*_2 = t_2$, then t_2 must be a self-loop because it can occur twice consecutively.

Secondly, assume we have $t_1' \; I \; t_1$, such that $BEn(r_1, t_1')$. A similar argument shows that t_2 must be a self-loop, too.

The second part of the lemma can be proved by a symmetric argument. $\square$

This lemma ensures that we do not need to consider the unfoldings of concurrently occurring self-loops. It is sufficient to match one instance of a concurrently occurring self-loop transition, and to make sure it is really matched to a self-loop.

This idea is translated into what we shall call *'no self-loop unfolding'* (short: *nsu*) hp bisimilarity. After giving the definition we will show that for systems with transitive independence relation this new kind of bisimilarity indeed coincides with (hereditary) hp bisimilarity.

Definition A.0.2. A *'no self-loop unfolding'* (short: *nsu*) *hp bisimulation* between two systems S_1 and S_2 consists of a set $\mathcal{H}_{nsu}$ of pairs (r_1, r_2) such that

(i) Whenever $(r_1, r_2) \in \mathcal{H}_{nsu}$, then r_1 is a run of S_1, r_2 is a run of S_2, and r_1 and r_2 are synchronous.

(ii) $(\varepsilon, \varepsilon) \in \mathcal{H}_{nsu}$.

(iii) Whenever $(r_1, r_2) \in \mathcal{H}_{nsu}$ and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, such that t_1 is *not* a concurrently occurring self-loop at r_1, then there exists t_2, such that $r_2 \xrightarrow{t_2} r_2.t_2$ and $(r_1.t_1, r_2.t_2) \in \mathcal{H}_{nsu}$.

(iv) Vice versa.

(v) Whenever $(r_1, r_2) \in \mathcal{H}_{nsu}$ and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, such that t_1 is a concurrently occurring self-loop at r_1, and there exists no x_2 such that $(t_1, x_2) \in BEn(r)$, then there exists t_2, such that t_2 is a self-loop, $r_2 \xrightarrow{t_2} r_2.t_2$, and $(r_1.t_1, r_2.t_2) \in \mathcal{H}_{nsu}$.

(vi) Vice versa.

A nsu hp bisimulation is *hereditary* (short: *h*) when it further satisfies

(vii) Whenever $(r_1, r_2) \in \mathcal{H}_{nsu}$ and $t_1 \in BEn(r_1)$ and $t_2 \in BEn(r_2)$ for some t_1, t_2 such that $last(r_1, t_1) = last(r_2, t_2)$, then $(\delta(r_1, t_1), \delta(r_2, t_2)) \in \mathcal{H}_{nsu}$.

We say two systems are (h) nsu hp bisimilar iff there is a (h) nsu hp bisimulation relating them.

Lemma A.0.2. *Two systems with transitive independence relation are (h) hp bisimilar iff they are (h) nsu hp bisimilar.*

Proof. With Lemma A.0.1 it is easy to check that every (h)hp bisimulation is also a (h) nsu hp bisimulation.

For the non-trivial direction let $\mathcal{H}_{nsu}$ be a (h) nsu hp bisimulation. Define $\mathcal{H}$ by unfolding self-loop matches inductively as follows:

Base Step $\mathcal{H} = \mathcal{H}_{nsu}$,

Inductive Step Whenever $rr' \in \mathcal{H}$ and t_1, t_2 is a pair of concurrently occurring self-loops at r_1, r_2, s. t. $(t_1, t_2) \in BEn(r)$ then $r.t.r' \in \mathcal{H}$.

It is easy to check that $\mathcal{H}$ is a (h)hp bisimulation. □

We can restrict our attention to the special class of *minimal* (h) nsu hp bisimulations, which strictly do not contain any unfoldings of concurrently occurring self-loops.

Definition A.0.3. A (h) nsu hp bisimulation $\mathcal{H}_{nsu}$ is *minimal* iff

- Whenever $r.t.r' \in \mathcal{H}_{nsu}$ and t_1 is a concurrently occurring self-loop at r_1, then there exists *no* x_2 such that $(t_1, x_2) \in BEn(r)$.
- Vice versa.

Lemma A.0.3. *Two systems are (h) nsu hp bisimilar iff there exists a minimal (h) nsu hp bisimulation relating them.*

Proof. We can simply 'collapse' any given (h) nsu hp bisimulation $\mathcal{H}_{nsu}$ to a minimal one: erase all tuples that violate the above conditions from $\mathcal{H}_{nsu}$. Clearly, the result is still a (h) nsu hp bisimulation. □

Minimal (h) nsu hp bisimulations between systems of our subclass look exactly like (h)hp bisimulations of systems with transitive independence relation and no self-loops. They meet all characterisics that made it possible to find a decision procedure for the latter subclass. In particular, the number of joint transitions which one can backtrack over is bounded by the size of the maximal independence clique. So, we get the following result.

Lemma A.0.4. *h nsu hp bisimilarity is decidable for finite-state systems with transitive independence relation.*

Proof. By Lemma A.0.3 it is sufficient to check whether there exists a minimal h nsu hp bisimulation. But this is clearly decidable for our subclass. We only need to adapt the steps of the proof of the decidability of (n)hhp bisimilarity to show that the corresponding notion of (n)h nsu hp bisimilarity is decidable for our subclass. □

With this and Lemma A.0.2 we immediately get decidability for the whole class of finite-state systems with transitive independence relation.

Appendix B

Relating to Chapter 4

B.1 Event Structures

In the following section we assume the standard definitions of (prime) event structures; they can, for example, be found in [Win89]. We also employ the following straightforward facts:

Fact B.1.1 (basic facts about event structures). *Let E be an event structure.*

1. *Let $e \in E$, $\sigma \subseteq E$ with $\forall e^p \in \sigma.\ e^p \prec e$.*

   $x \backslash (\sigma \cup \{e\}) \in \mathit{Confs}(E)$.

2. *Let e_1^p, $e_2^p \in E$ such that $e_1^p \neq e_2^p$.*

   $(\exists e \in E.\ e_1^p \prec e \ \&\ e_2^p \prec e) \implies e_1^p\ co\ e_2^p$.

3. *Let $x \in \mathit{Confs}(E)$, $e_1, e_2 \in E$.*

   $x[e_1\rangle \ \&\ x[e_2\rangle \implies (e_1\ co\ e_2) \vee (e_1\ \#\ e_2)$.

B.2 Proof of Theorem 4.4.2.

It is clear that a given system S is SW-free iff $\mathit{unf}(S)$ is SW-free, and thus Theorem 4.4.2 will follow if we achieve:

Lemma B.2.1. *An occurrence lats is SW-free iff it is cis-decomposable.*

The 'if'-direction is easy to prove: assume a system S such that there is a SW situation at some reachable state s of S. Then, it is easy to check that S cannot be decomposed into a set of IS components at s. Hence, S cannot be cis-decomposable.

To establish the 'only if'-direction it is technically smoother to work with event structures. We carry the definition of SW-free, IS, and cis-decomposable over to event structures in the obvious way. Clearly, we have:

Fact B.2.1. *Let U be an occurrence lats.*

1. *U is SW-free iff $ev(U)$ is SW-free.*

2. *U is cis-decomposable iff $ev(U)$ is cis-decomposable.*

For any SW-free event structure E we exhibit a family of decompositions which will demonstrate that E can be decomposed into IS components at any configuration. We then conclude that any SW-free occurrence lats is cis-decomposable.

First of all, we establish some behavioural consequences of excluding SW situations, which will come in useful later on.

Lemma B.2.2 (consequences of excluding SW situations). *Let E be an event structure.*

1. *If E is SW-1 free then for all e_1, $e_2 \in E$ we have:*

$$\exists e_j \in E.\ e_1 < e_j \ \&\ e_2 < e_j \implies \neg(e_1 \ co \ e_2).$$

2. *If E is SW-1 and SW-2 free then for all e_1, $e_2 \in E$ we have:*

$$\exists e_1^c, e_2^c \in E.\ e_1 < e_1^c \ \&\ e_2 \leq e_2^c \ \&\ e_1^c \ \#\ e_2^c \implies \neg(e_1 \ co \ e_2).$$

3. *Let $x \in Confs(E)$. If E is SW-3 free then $\#$ is transitive*[1] *at x, that is for all distinct events $e_1, e_2, e_3 \in E$ with $x[e_1\rangle$, $x[e_2\rangle$ & $x[e_3\rangle$ we have:*

$$e_1 \# e_2 \ \&\ e_2 \# e_3 \implies e_1 \# e_3.$$

Proof. Let E be an event structure.

(1.) Assume $e_1, e_2, e_j \in E$ such that $e_1 < e_j$, $e_2 < e_j$, and $e_1 \ co \ e_2$. Consider the set $\sigma = \{e \in E \mid e_1 < e \ \&\ e_2 < e\}$. Since it at least contains e_j the set σ is non-empty, and hence, we can assume a minimal element e of σ. Let's regard e's prime configuration $x = \downarrow e$. Since $e \in \sigma$ there must be $e_1^p, e_2^p \in x$ with $e_1 \leq e_1^p \prec e$, and $e_2 \leq e_2^p \prec e$ respectively. Now, consider $x' = x \backslash \{e\}$, and $x'' = x \backslash \{e, e_1^p, e_2^p\}$. With Fact B.1.1(1) it is clear that x' and x'' are valid configurations such that $x'[e\rangle$, $x''[e_1^p\rangle$, and $x''[e_2^p\rangle$. Further, by $e_1 \ co \ e_2$ and the minimality of e in σ, e_1^p and e_2^p must be distinct, and hence concurrent by Fact B.1.1(2). But then it is

[1] when regarding distinct events.

easy to see that (x'', e_1^p, e_2^p, e) is a SW-1 situation, and we have thereby proved the property.

(2.) The proof is in principle similar to (1), but it requires a tedious case analysis; it is therefore omitted.

(3.) Let $x \in \mathit{Confs}(E)$, and let $e_1, e_2, e_3 \in E$ be distinct events with $x[e_1\rangle$, $x[e_2\rangle$ & $x[e_3\rangle$. Clearly, if $e_1 \# e_2$, $e_2 \# e_3$ & e_1 *co* e_3 then (x, e_1, e_3, e_2) is a SW-3 situation. The property is then immediate with Fact B.1.1(3). $\square$

Now, we define:

Definition B.2.1 (icfl_E, comp_E, Comps_E). Let E be a SW-free event structure, $x \in \mathit{Confs}(E)$, and $e \in E$ with $x[e\rangle$.

The *inclusive conflict set of e at x*, denoted by $\mathit{icfl}_E(x, e)$, is the set $\{e' \in E \mid x[e'\rangle \;\&\; e \# e'\} \cup \{e\}$.

The *component of x defined by e*, denoted by $\mathit{comp}_E(x, e)$, is the tuple $(E', \leq_E \restriction_{E' \times E'}, \#_E \restriction_{E' \times E'})$, where $E' = \bigcup_{e' \in \mathit{icfl}_E(x,e)} \uparrow e'$.

The *components of x* are then defined by: $\mathit{Comps}_E(x) = \{\mathit{comp}_E(x, e) \mid e \in E \text{ such that } x[e\rangle\}$.

We claim that Comps_E gives us a family of decompositions for each SW-free event structure as we have promised to exhibit. To prove this claim we first establish that comp_E satisfies the following essential properties:

Lemma B.2.3 (facts about comp_E). *Let E be an SW-free event structure, $x \in \mathit{Confs}(E)$, and $e \in E$ with $x[e\rangle$.*

1. *$\mathit{comp}_E(x, e)$ is a sub-structure of (E, x).*
2. *$\mathit{comp}_E(x, e)$ is IS.*
3. *Let $e' \in E$ with $x[e'\rangle$.*
   (a) *$e' \# e \implies \mathit{comp}_E(x, e') = \mathit{comp}_E(x, e)$.*
   (b) *e' co $e \implies \mathit{comp}_E(x, e')$ co $\mathit{comp}_E(x, e)$.*
4. *Let $e' \in \mathit{comp}_E(x, e)$.*
   (a) *$\downarrow_{\mathit{comp}_E(x,e)} e' = \downarrow_{(E,x)} e'$.*
   (b) *$\mathit{cfl}_{\mathit{comp}_E(x,e)}(e') = \mathit{cfl}_{(E,x)}(e')$.*

Proof. Let E, x, and e be given as above.

(1.) Obvious by definition.

(2.) This is straightforward to prove with Lemma B.2.2(3).

(3.) Let $e' \in E$ with $x[e'\rangle$.

(a) If $e' \# e$ then by Lemma B.2.2(3) we have $\mathit{icfl}_E(x, e') = \mathit{icfl}_E(x, e)$, and thus clearly $\mathit{comp}_E(x, e') = \mathit{comp}_E(x, e)$.

(b) Let e' *co* e, and to the contrary assume there are $e_1 \in \mathit{comp}_E(x, e)$ and $e_2 \in \mathit{comp}_E(x, e')$ such that $\neg(e_1 \mathit{\ co\ } e_2)$. This means we either have (1) $e_1 = e_2$, (2) $e_1 < e_2$, (3) $e_1 > e_2$, or (4) $e_1 \# e_2$. We show that all of these cases lead to a contradiction with SW-freeness, and thereby establish the property. By definition of comp_E there must be $e_1^i, e_2^i \in E$ such that $e_1^i \in \mathit{icfl}_E(x, e)$ & $e_1^i \leq e_1$, and $e_2^i \in \mathit{icfl}_E(x, e')$ & $e_2^i \leq e_2$ respectively. Note that with Lemma B.2.2(3) and Fact B.1.1(3) we obtain $e_1^i \mathit{\ co\ } e_2^i$; in the following we refer to this fact by (A).

Suppose (1) $e_1 = e_2$; use e_1. Because of (A) we have $e_1^i \neq e_1$, $e_2^i \neq e_1$, and hence $e_1^i < e_1$, $e_2^i < e_1$. But by Lemma B.2.2(1) and (A) this certainly is a contradiction to SW-freeness.

Let's try (2) $e_1 < e_2$. Firstly, this assumption immediately gives us $e_1^i < e_2$. With (A) we further obtain $e_2 \neq e_2^i$, and hence $e_2^i < e_2$. But then again by Lemma B.2.2(1) and (A) we have reached a contradiction to SW-freeness.

Case (3) can be disproved by the symmetric argument.

Finally, assume (4) $e_1 \# e_2$. By (A) we obtain that at least one of the two statements must hold: $e_1^i \neq e_1$ or $e_2^i \neq e_2$. W.l.o.g. assume the first, and with it $e_1^i < e_1$. Then by Lemma B.2.2(2) we have clearly a contradiction to SW-freeness since altogether: $e_1^i < e_1$, $e_2^i \leq e_2$, $e_1 \# e_2$, but $e_1^i \mathit{\ co\ } e_2^i$.

(4.) Let $e' \in \mathit{comp}_E(x, e)$, and to abbreviate set $c = \mathit{comp}_E(x, e)$, and $E_x = (E, x)$.

(a) By definition of c, $(\downarrow_c e') \subseteq (\downarrow_{E_x} e')$ is clearly given, and $(\downarrow_c e') \supseteq (\downarrow_{E_x} e')$ follows from $(\downarrow_{E_x} e') \subseteq c$.

To prove the latter assume $e^* \in E_x$ with $e^* \leq e'$ but $e^* \notin c$; we will see that this assumption leads to a contradiction. Since $e' \in c$, we clearly have $e^* < e'$. Then e' is not minimal in E_x, and by definition of comp_E we further obtain $e_i' \in c$ such that e_i' is minimal in E_x, and $e_i < e'$. On the other hand, we can also assume $e_i^* \in E_x$ such that e_i^* is minimal in E_x and $e_i^* \leq e^*$. By $e^* \notin c$ we must also have $e_i^* \notin c$. Then we have $\neg(e_i^* \# e_i')$, and by Fact B.1.1(3) $e_i' \mathit{\ co\ } e_i^*$. But with Lemma B.2.2(1) we have now arrived at a contradiction with SW-freeness.

(b) By definition of c, $\mathit{cfl}_c(e') \subseteq \mathit{cfl}_{E_x}(e')$ is obvious, and for $\mathit{cfl}_c(e') \supseteq \mathit{cfl}_{E_x}(e')$ it is sufficient to prove $\mathit{cfl}_{E_x}(e') \subseteq c$.

To the contrary assume $e^* \in E_x$ with $e^* \# e'$ but $e^* \notin c$. Clearly, there must be e'_i, $e^*_i \in E_x$ such that e'_i, e^*_i are minimal in E_x, and $e'_i \leq e'$, $e^*_i \leq e^*$. Because $e^* \notin c$, we also have $e^*_i \notin c$, and further $\neg(e^*_i \# e'_i)$. Then, by Fact B.1.1(3) we obtain $e'_i \; co \; e^*_i$. With Lemma B.2.2(3) this in turn means we have $e'_i \neq e'$ or $e^*_i \neq e^*$. But note that in either case by Lemma B.2.2(2) we have reached a contradiction to SW-freeness. $\square$

Now, it is not difficult to obtain:

Lemma B.2.4 (crucial fact about Comps_E). *Let E be an SW-free event structure, and let $x \in \mathit{Confs}(E)$. Then $\mathit{Comps}_E(x)$ is a decomposition of (E, x) into IS systems.*

Proof. Suppose E and x are given as above, and set $\mathcal{D} = \mathit{Comps}_E(x)$. We need to show: (a) for all $c \in \mathcal{D}$ c is an IS sub-structure of (E, x), (b) for all $c, c' \in \mathcal{D}$ $c \neq c' \Rightarrow c \; co \; c'$, and (c) $\mathit{Confs}(E, x) = \{\bigcup_{c \in \mathcal{D}} x_c \mid \forall c \in \mathcal{D}.\ x_c \in \mathit{Confs}(c)\}$.

(a) is obvious with Lemma B.2.3(1) and (2); (b) follows from Fact B.1.1(3) and Lemma B.2.3(3a,b). To establish (c) we have to prove that for all $x' \subseteq (E, x)$: $x' \in \mathit{Confs}(E, x)$ iff there is a family $\{x_c\}_{c \in \mathcal{D}}$ such that $\forall c \in \mathcal{D}.\ x_c \in \mathit{Confs}(c)$ and $x' = \bigcup_{c \in \mathcal{D}} x_c$. This can be done by induction on the size of x' with the help of Lemma B.2.3(4a,b). $\square$

It is clear that together with Fact B.2.1, Lemma B.2.4 immediately proves Lemma B.2.1, and hence Theorem 4.4.2.

Appendix C

Relating to Chapter 6

C.1 Some Intuition

A Decomposition View for Live fc Systems. The crucial feature of free choice systems is that conflict and synchronization are both allowed but separated out from each other: conflict is only permitted in the 'S-system way', and synchronization only in the 'T-system way' (cf. Figure 6.3). As a result, the unfolding of a free choice system has good decomposition properties: it can be understood as an interconnection of initially sequential components. In the following, we derive and elaborate this view slightly more specialized as a decomposition aspect of *live* fc systems, our class of focus in this chapter. Naturally, we work in the context of a live fc system $\mathcal{N}$.

The Structure of States. Assume $M \in \mathit{Reach}(\mathcal{N})$, and let $\mathit{APlaces}(M)$ be the places that are active at M in that they take part in enabling a transition at M (formally: $\mathit{APlaces}(M) = \{{}^\bullet t \mid t \in T_N \text{ with } M[t\rangle\}$). Due to the fc restriction, $\mathit{APlaces}(M)$ can uniquely be partitioned into blocks such that the places of a block have identical sets of output transitions, say T_{out}, and together they enable all the transitions of T_{out}. In more detail, we have: either (1) a block contains exactly one place which enables a non-empty set of transitions in the S-system way, or (2) a block consists of several places which together enable exactly one transition in the T-system way. Accordingly, we classify the blocks into *switch blocks* and *synch blocks*; the default case (one place enables one transition) is set to type switch. The places of M not contained in $\mathit{APlaces}(M)$ are synch places that are waiting for their synch partners to get ready (formally: $p \in M$ such that $\exists p' \in {}^\bullet(p^\bullet).\ p' \notin M$); we will refer to them as *pending synch places*. In summary, we have:

> Each state of a live fc system can uniquely be structured into a set of blocks, which classify into type switch and synch, and a set of pending

synch places.

Units. The blocks of M give rise to special components of $\mathcal{N}$: each block b defines a *unit* (N, b), which according to the type of b can be classified as either a switch or a synch unit. Four things are important about the behaviour of units: (1) The units of M compute independently of each other. (2) Each unit is initially sequential. (3) Each unit is capable of external synchronization at and only at its behavioural end points. (4) In the unfolding of each unit, the place occurrences that can act as end points are related by a typical pattern of concurrency and conflict.

Global Behaviour. The behaviour of $\mathcal{N}$ can then be described as follows: the system starts out as the parallel composition of the initially sequential units induced by M_0. Due to the units' capability of external synchronization the behaviour can go beyond this initial stage: if a unit has reached a local end point then together with end points of other units and/or initially pending synch places it may be able to form a new unit, say U, of type synch. U will be causally related to its 'parent units' and concurrent to the remaining ones. It in turn may create further units by synchronizing with existing units and/or initially pending synch places. The synchronization partners of U can be of any 'level of creation', and may even include U's parent units, which could have several concurrent end points. The behaviour of $\mathcal{N}$ further evolves in this fashion. Accordingly, the unfolding of $\mathcal{N}$ can be seen as a complex interconnection of unfoldings of initially sequential units, where the interconnection consists of causality, concurrency, and conflict; the conflict relation will be induced by the conflict relation that connects the end points of each unit.

A Hierarchy of Units. The same view carries on within each unit in a hierarchical fashion: by executing a transition the unit will evolve into a new local state, which gives rise to a set of independently computing sub-units and a set of locally pending synch places. The sub-units have corresponding synchronization capability, which can now occur at the different levels of the hierarchy: let U be a unit, and U_s be a sub-unit of U; at the level of U an end point of U_s is either an end point as well, or an internal synchronization place.

Conclusion. Altogether, this gives us our *decomposition view for live fc systems* (short: *DV-lfcs*):

> The unfolding of a live fc system can be understood as a complex ccc-interconnection of unfoldings of initially sequential units, and the same view applies within each unit in a hierarchical way.

(We use ‘ccc-interconnection’ short for ‘interconnection which consists of causality, concurrency, and conflict’.)

General Idea. As already mentioned, fc systems generalize comm-free systems by admitting synchronization in the T-system way. Accordingly, we can understand DV-lfcs as a generalization of our decomposition view for comm-free systems, which can be formulated as follows: each comm-free system is a parallel composition of initially sequential units, and the same view applies within each unit in a hierarchical way (cf. Section 4.4 and 5.3). DV-lfcs adds synchronization capability to this view: now units can jointly evolve into new units at their behavioural end points.

This analogy immediately suggests a general way of tackling the coincidence problem for live fc systems: we obtained our coincidence results for comm-free systems (in the disguise of SW-free systems and SBPP; cf. Section 4.4 and 5.3) with the help of the key insight that their decomposition properties translate into corresponding composition and decomposition results for hp and hhp bisimilarity. We could now try to prove that hp and hhp bisimilarity coincide for live fc systems by showing that DV-lfcs translates into hhp and hp bisimilarity in an analogous way.

Informally, DV-lfcs can be translated into a composition and decomposition property for notions of bisimilarity as follows:

Definition C.1.1 (informal). Let x be a notion of bisimilarity.

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two live fc systems. We say $\mathcal{W}$ is a *x DV-lfcs match for $\mathcal{N}_1$ and $\mathcal{N}_2$* iff $\mathcal{W}$ is a ccc-interconnection of x bisimulations such that

1. each x bisimulation of $\mathcal{W}$ relates a unit of $\mathcal{N}_1$ with a unit of $\mathcal{N}_2$, and
2. for $i = 1, 2$, the projection of $\mathcal{W}$ onto $\mathcal{N}_i$ is ‘isomorphic up to conflict’ to the unfolding of $\mathcal{N}_{\bar{i}}$.

We say x bisimilarity is *DV-lfcs composable* iff for any two live fc systems $\mathcal{N}_1$ and $\mathcal{N}_2$ we have: if there exists a x DV-lfcs match for $\mathcal{N}_1$ and $\mathcal{N}_2$ then $\mathcal{N}_1$ and $\mathcal{N}_2$ are x bisimilar.

We say x bisimilarity is *DV-lfcs decomposable* iff for any two live fc systems $\mathcal{N}_1$ and $\mathcal{N}_2$ we have: if $\mathcal{N}_1$ and $\mathcal{N}_2$ are x bisimilar then there exists a x DV-lfcs match for $\mathcal{N}_1$ and $\mathcal{N}_2$.

If we achieve

1. hhp bisimilarity is DV-lfcs composable, and
2. hp bisimilarity is DV-lfcs decomposable,

then it should be possible to prove coincidence for live fc systems analogously to our coincidence proof for SW-free systems (cf. Section 4.4.3): let $\mathcal{N}_1 \sim_{hp} \mathcal{N}_2$; roughly speaking we can obtain $\mathcal{N}_1 \sim_{hhp} \mathcal{N}_2$ in the following three steps: (a) By (2) we can assume a hp DV-lfcs match for $\mathcal{N}_1$ and $\mathcal{N}_2$, say $\mathcal{W}$. (b) By induction on the smallest upper bound on the number of transitions that can be executed concurrently we can transform each hp bisimulation of $\mathcal{W}$ into an hhp bisimulation, and thereby obtain an hhp DV-lfcs match. (c) By (1) we can then indeed conclude $\mathcal{N}_1 \sim_{hhp} \mathcal{N}_2$.

We expect that (1) can be proved without any difficulty; it is very intuitive that hhp (and also hp) bisimilarity is DV-lfcs composable. The real challenge lies in showing that hp bisimilarity is DV-lfcs decomposable. This is far from obvious, and the degree of difficulty involved is considerable: a ccc-interconnection can be a very complicated structure (as opposed to a parallel composition as for comm-free systems!). Therefore, we cannot hope to solve the problem in one go, but we will present an approach that divides it up into several subproblems. Our approach will have the merit that decidablity of (c)hhp bisimilarity can be achieved as a partial result.

C.2 Diamond Interrelations

A *trace alphabet* is a pair (Σ, I), where the alphabet Σ is a finite set, and $I \subseteq \Sigma \times \Sigma$ is an irreflexive and symmetric independence relation. Let t, t' range over Σ. Let Σ^* be the set of finite words over Σ, and let r, r', w, v range over Σ^*. The independence relation I induces a relation $\sim_I \subseteq \Sigma^* \times \Sigma^*$ defined by $r \sim_I r'$ iff $r \uparrow \{t, t'\} = r' \uparrow \{t, t'\}$ for all $t, t' \in \Sigma$ such that $\neg(t \; I \; t')$. Clearly, $\sim_I$ is an equivalence relation.

We define the following properties:

Definition C.2.1 (properties). Let (Σ, I) be a trace alphabet. A language $\mathcal{H} \subseteq \Sigma^*$ can satisfy the following properties w.r.t. I:

1. **1-backtrack** (short: **1-bt**): $rtt' \in \mathcal{H} \;\&\; t \; I \; t' \implies rt' \in \mathcal{H}$.
2. **backtrack** (short: **bt**): $rtw \in \mathcal{H} \;\&\; t \; I \; w \implies rw \in \mathcal{H}$.
3. **1-coherent** (short: **1-coh**): $rt \in \mathcal{H} \;\&\; rt' \in \mathcal{H} \;\&\; t \; I \; t' \implies rtt' \in \mathcal{H}$.
4. **(wt)-coherent** (short: **(wt)-coh**):
   $rw \in \mathcal{H} \;\&\; rt \in \mathcal{H} \;\&\; t \; I \; w \implies rwt \in \mathcal{H}$.
5. **(tw)-coherent** (short: **(tw)-coh**):
   $rw \in \mathcal{H} \;\&\; rt \in \mathcal{H} \;\&\; t \; I \; w \implies rtw \in \mathcal{H}$.
6. **continuation** (short: **cont**): $r, rw, r' \in \mathcal{H} \;\&\; r \sim_I r' \implies r'w \in \mathcal{H}$.
7. **(tw, w)-closure** (short: **(tw, w)-cl**):
   $rtw \in \mathcal{H} \;\&\; rw \in \mathcal{H} \;\&\; t \; I \; w \implies rwt \in \mathcal{H}$.
8. **(tw, w)-closure continuation** (short: **(tw, w)-clcont**):
   $rtwv \in \mathcal{H} \;\&\; rw \in \mathcal{H} \;\&\; t \; I \; w \implies rwtv \in \mathcal{H}$.

Let X range over the set of properties $\{$**1-bt**, **bt**, **1-coh**, **(wt)-coh**, **(tw)-coh**, **cont**, **(tw, w)-cl**, **(tw, w)-clcont**$\}$. We write $\mathcal{H} \models_I X$ to denote that $\mathcal{H}$ satisfies property X w.r.t. I. To express that $\mathcal{H}$ satisfies several properties we allow a list of properties behind the '$\models_I$'.

We first state some obvious interrelations:

Proposition C.2.1 (obvious interrelations). *Let (Σ, I) be a trace alphabet, and let $\mathcal{H} \subseteq \Sigma^*$ be a* prefix-closed *language over Σ. We obviously have:*

1. $\mathcal{H} \models_I$ **bt** $\implies \mathcal{H} \models_I$ **1-bt**,

2. $\mathcal{H} \models_I$ **(wt)-coh** $\implies \mathcal{H} \models_I$ **1-coh***,*

3. $\mathcal{H} \models_I$ **(tw)-coh** $\implies \mathcal{H} \models_I$ **1-coh***, and*

4. $\mathcal{H} \models_I$ **(tw, w)-clcont** $\implies \mathcal{H} \models_I$ **(tw, w)-cl***.*

For *prefix-closed* languages there are the following connections:

Proposition C.2.2 (interrelations between the properties). *Let* (Σ, I) *be a trace alphabet, and let* $\mathcal{H} \subseteq \Sigma^*$ *be a* prefix-closed *language over* Σ*. Then the following holds:*

1. $\mathcal{H} \models_I$ **1-coh** $\iff \mathcal{H} \models_I$ **(wt)-coh***,*

2. $\mathcal{H} \models_I$ **1-coh** & **cont** $\implies \mathcal{H} \models_I$ **(tw)-coh***,*

3. $\mathcal{H} \models_I$ **1-coh***,* **1-bt** & **cont** $\implies \mathcal{H} \models_I$ **bt***, and*

4. $\mathcal{H} \models_I$ **1-bt** & **(tw, w)-clcont** $\implies \mathcal{H} \models_I$ **bt***.*

Proof. (1.) The '$\Leftarrow$'-direction is immediate. For the '$\Rightarrow$' direction we presuppose $\mathcal{H} \models_I$ **1-coh**, and let $rw, rt \in \mathcal{H}$ with $t\ I\ w$. We show $rwt \in \mathcal{H}$ by induction on the length of w.

If $w \equiv \varepsilon$ there is nothing to prove: $rt \in \mathcal{H}$ by assumption. We move on to the inductive case and assume $w \equiv vt'$. By prefix-closure of $\mathcal{H}$ we then have $rv \in \mathcal{H}$; together with the induction hypothesis, $rt \in \mathcal{H}$, and $t\ I\ v$ this gives us $rvt \in \mathcal{H}$. Now, with $rvt', rvt \in \mathcal{H}$ and $t'\ I\ t$ we can apply the **1-coh**-property, and thereby obtain $rvt't \in \mathcal{H}$ as required.

(2.) We assume the **1-coh**- and the **cont**-property for $\mathcal{H}$, and let $rw, rt \in \mathcal{H}$ with $t\ I\ w$. Again we will proceed by induction on the length of w; this time to establish $rtw \in \mathcal{H}$.

As before, in the case $w \equiv \varepsilon$, $rtw \equiv rt \in \mathcal{H}$ is given by assumption.

For the inductive case let $w \equiv vt'$. By prefix-closure of $\mathcal{H}$ we know that $rv \in \mathcal{H}$. $rv \in \mathcal{H}$ together with $rt \in \mathcal{H}$ and $t\ I\ v$ gives us $rvt \in \mathcal{H}$ by the **(wt)-coh**-property. We are allowed to employ the **(wt)-coh**-property, because as we know from (1.) it follows from the **1-coh**-property. With $rvt, rvt' \in \mathcal{H}$ and $t\ I\ t'$ we then apply the **1-coh**-property to obtain $rvtt' \in \mathcal{H}$.

On the other hand from $rv \in \mathcal{H}$, $rt \in \mathcal{H}$, and $t\ I\ v$ we deduce $rtv \in \mathcal{H}$ by induction hypothesis. Now, together with $rvt, rvtt' \in \mathcal{H}$ this makes a case for the **cont**-property, since clearly $rvt \sim_I rtv$. But this gives us $rtvt' \in \mathcal{H}$, which is exactly what we have set out to prove.

(3.) Presuppose $\mathcal{H} \models_I$ **1-coh**, **1-bt** & **cont**, and let $rtw \in \mathcal{H}$ with $t\ I\ w$. We prove $rw \in \mathcal{H}$ by induction on the length of w. If $w \equiv \varepsilon$, then $rw \equiv r \in \mathcal{H}$ follows from $\mathcal{H}$ being prefix-closed.

To tackle the inductive case we assume $w \equiv vt'$. By prefix-closure of $\mathcal{H}$ we have $rtv \in \mathcal{H}$ and also $rt \in \mathcal{H}$. From $rtv \in \mathcal{H}$ and $t\ I\ v$ together with the induction hypothesis we obtain $rv \in \mathcal{H}$. This, $rt \in \mathcal{H}$, and $t\ I\ v$ in turn give us $rvt \in \mathcal{H}$ by the **1-coh**-property and clause (1.) of the proposition.

We have $rtv, rtvt', rvt \in \mathcal{H}$, and clearly $rvt \sim_I rtv$; so by the **cont**-property we know that $rvtt' \in \mathcal{H}$. But now we can apply the **1-bt**-property, and easily obtain $rvt' \in \mathcal{H}$.

(4.) To prove the last clause we assume that the **1-bt**- and the **(tw, w)-clcont**-property hold for $\mathcal{H}$, and let $rtw \in \mathcal{H}$ with $t\ I\ w$. We show $rw \in \mathcal{H}$ by induction on the length of w.

Case $w \equiv \varepsilon$: We need to prove $r \in \mathcal{H}$. But this is clearly given because $rtw \in \mathcal{H}$ and $\mathcal{H}$ is prefix-closed.

Case $w \equiv vt'$: By prefix-closure of $\mathcal{H}$ and $rtvt' \in \mathcal{H}$ we have $rtv \in \mathcal{H}$. With the induction hypothesis we further obtain $rv \in \mathcal{H}$. $rtvt', rv \in \mathcal{H}$, $t\ I\ v$ and the **(tw, w)-clcont**-property then give us $rvtt' \in \mathcal{H}$. Now we can apply the **1-bt**-property and obtain $rvt' \in \mathcal{H}$ as required. □

We now apply our knowledge about diamond relations to (h)hp bisimilarity. Given two systems S_1 and S_2, we can regard (h)hp bisimulations as languages over the trace alphabet $(\Sigma, I) := (T_{S_1} \times T_{S_2}, I_{(S_1,S_2)})$. First of all, we have that **cont** and **(tw, w)-clcont** hold for $\sim_{hp}$ and $\sim'_{hp}$, where the latter denotes the largest *prefix-closed* hp bisimulation.

Proposition C.2.3. *We have:*

1. *(a)* $\sim_{hp} \models_I$ **cont**, *and (b)* $\sim'_{hp} \models_I$ **cont**.
2. *(a)* $\sim_{hp} \models_I$ **(tw, w)-clcont**, *and (b)* $\sim'_{hp} \models_I$ **(tw, w)-clcont**.

Proof. (1.(a)) Define a relation R inductively by:

base case. $R = \sim_{hp}$,

inductive case. if $r, rw, r' \in R$ & $r \sim_I r'$ then $r'w \in R$.

It is easy to check that R is a hp bisimulation. Since $\sim_{hp}$ is the largest hp bisimulation and $\sim_{hp} \subseteq R$ by definition of R we clearly have $R = \sim_{hp}$. R obviously satisfies the **cont**-property, and so does $\sim_{hp}$.

(1.(b)) follows by a similar argument, when we base R on $\sim'_{hp}$ instead of $\sim_{hp}$.

(2.(a)+(b)) We proceed similar as in (1.) and define a relation R that obviously satisfies the **(tw, w)-clcont**-property. This R will also be a hp bisimulation and so we can argue as above. □

We now exploit the interrelations to show that the **1-coh**- and the **(tw)-coh**-property coincide for $\sim'_{hp}$, and similarly that the **1-bt**- and **bt**-property coincide.

Proposition C.2.4.

1. $\sim'_{hp} \models_I$ **1-coh** $\iff \sim'_{hp} \models_I$ **(tw)-coh**.
2. $\sim'_{hp} \models_I$ **1-bt** $\iff \sim'_{hp} \models_I$ **bt**.

Proof. (1.) The '⇐'-direction is obvious. On the other hand, the '⇒'-direction follows from Prop. C.2.2(2.) since we know from above that $\sim'_{hp}$ satisfies the **cont**-property.

(2.) Again the '⇐'-direction is obvious, and the '⇒'-direction follows from a connection between the language properties: $\sim'_{hp}$ satisfies the **(tw, w)-clcont**-property, and so Prop. C.2.2(4.) applies. □

Note how this can help us to establish coincidence of hp and hhp bisimilarity for a system class (recalling that we can restrict our attention to prefix-closed hp bisimulations). If we can show that the **1-bt**-property holds for $\sim'_{hp}$ then coincidence follows immediately, and similarly for **1-coh**.

Corollary C.2.1.

1. $\sim'_{hp} \models_I$ **1-bt** $\implies \sim'_{hp} = \sim_{hhp}$, *and*
2. $\sim'_{hp} \models_I$ **1-bt** & **1-coh** $\implies \sim'_{hp} = \sim_{chhp}$.

Proof. (1.) If $\sim'_{hp}$ satisfies **1-bt**, and thus **bt**, then $\sim'_{hp}$ is a hhp bisimulation, and we have $\sim'_{hp} \subseteq \sim_{hhp}$. Of course we also have $\sim_{hhp} \subseteq \sim'_{hp}$, and so $\sim'_{hp} = \sim_{hhp}$.

(2.) follows from a similar argument. □

C.3 Appendix to Section 6.6

Proof of Prop. 6.6.7. Assume entities as specified above. Either we have: (a) $p' \in M$ or (b) $p' \notin M$. If (b) holds then we have $K'(p) \prec_M K$, and so clearly $K \in \mathit{frozen}(M, \mathcal{C})$.

If (a) holds then consider the following: $K' \in \mathit{frozen}(M, \mathcal{C})$ means there must be $K'' \in \mathit{Cover}$, $p'' \in P_{K''}$ such that (W) $K''(p'') \prec_M K'$ and either we have $K'' \in \mathcal{K}$ & $\not\exists t \in \gamma^{\mathcal{C}}_{K''}.\ p \in t^\bullet$, or $K'' \in \mathit{frozen}(M, \mathcal{C})$. Note that to show $K \in \mathit{frozen}(M, \mathcal{C})$ we only need $K''(p'') \prec_M K$, and this in turn will follow from $p'' \in \mathit{SPartners}(M(K))$. This is clearly the case with the following three facts: (1) $p'' \in \mathit{SPartners}(p')$ by (W), (2) $p'' \neq p$ by $p \in M$ and $p'' \notin M$ (follows from (W)), and (3) $\mathit{SPartners}(M(K)) \backslash p' = \mathit{SPartners}(p') \backslash M(K)$ by $p' \in \mathit{SPartners}(M(K))$ □

C.4 Appendix to Section 6.14

C.4.1 Relating to Section 6.14.2

First, we formally define $syass(r)$, and verify that it behaves as expected.

Definition C.4.1. For all $r \in \sim_{hp}$ we inductively define a map $syass$ to obtain a match between $synchP(r_1)$ and $synchP(r_2)$:

$$
\begin{array}{lcl}
syass(\varepsilon) & = & \emptyset, \\
syass(r.t) & = & syass(r) \backslash \beta^{sy}_{pre} \cup \beta^{sy}_{post},
\end{array}
$$

where

$$\beta^{sy}_{pre} = \begin{cases} \emptyset & \text{if } t \text{ is of type switch,} \\ \{(p_1, p_2) \in syass(r) \mid p_i \in {}^\bullet t_i \text{ for } i = 1, \text{ or } 2\} & \text{if } t \text{ is of type synch,} \end{cases}$$

and

$$\beta^{sy}_{post} = \begin{cases} \emptyset & \text{if } t \text{ is of type A,} \\ \{({t_1}^\circ, {t_2}^\circ)\} & \text{if } t \text{ is of type B.} \end{cases}$$

Proposition C.4.1. *Let $r \in \sim_{hp}$.*

1. *$syass(r)$ is a bijection between $synchP(r_1)$ and $synchP(r_2)$.*

2. *$\forall p_1 \in synchP(r_1).\ syass(r)(p_1) = (t_2^e)^\circ$, where $e = gen(r_1, p_1)$.*

3. *Let γ be such that $(r, \gamma) \in \sim_{bp}$, and $b_1 \in blocks(r_1)$, $b_2 \in blocks(r_2)$ such that b_1 or b_2 is of type synch.*

$$\gamma(b_1) = b_2 \implies syass(b_1) = b_2.$$

4. *Let $t \in JT$ such that t_1 or t_2 is of type synch.*

$$r \xrightarrow{t} \implies syass(r)({}^\bullet t_1) = {}^\bullet t_2.$$

Proof. Let $r \in \sim_{hp}$. We prove the four clauses by induction on the length of r 'in one go'.

Base case $r \equiv \varepsilon$: (1)-(4) are immediate when considering that by definition of the buffered restriction we have $synchP(M_0) = \emptyset$.

Inductive case $r \equiv r'.t^n$: (1) easily follows by induction hypothesis of (1) and the following considerations. With the induction hypothesis of (4) it is easy to see that the pairs of places deleted from the assignment exactly correspond to the synch places consumed by t^n. Furthermore, the pairs of places added to the assignment exactly cover the synch places produced by t^n, and by safeness it is clear that the assignment stays a bijection.

(2) Assume $p_1 \in synchP(r_1)$, and set $e = gen(r_1, p_1)$. If $e < |r.t^n|$ then (2) easily follows with the induction hypothesis of (2). If, on the other hand, $e = |r.t^n|$ then t^n must be of type B (Prop. 6.14.1(1)), and it can directly be read from the definition that $syass(r)(p_1) = (t_2^e)^\circ$ as required.

(3) is a consequence of (2) and Prop. 6.14.4(2), and (4) follows from (2) and Prop. 6.14.4(1b). $\square$

We are now ready to define our map *jproc*, and prove that it provides a translation as required to transform any bp bisimulation into a cp bisimulation:

Definition C.4.2. For all $\rho \equiv (r, \gamma) \in \sim_{bp}$ we define a map *jproc* to translate ρ into a corresponding joint process:

$$jproc(\rho) = \{(p_1, p_2) \mid (\{p_1\}, \{p_2\}) \in \gamma\} \cup syass(r).$$

Proposition C.4.2. *Let* $\rho \equiv (r, \gamma) \in \sim_{bp}$.

1. $jproc(\rho) \in JProc$.
2. (a) *For all* $b_1 \in blocks(r_1)$, $b_2 \in blocks(r_2)$ *we have:*

$$\gamma(b_1) = b_2 \implies jproc(\rho)(b_1) = b_2.$$

   (b) *For all* $t \in JT$, $\rho' \in BP$ *we have:*

$$\rho \xrightarrow{t}_{bp} \rho' \ \&\ \rho' \in \sim_{bp} \implies jproc(\rho) \xrightarrow{t}_{cp} jproc(\rho').$$

Proof. Let $\rho \equiv (r, \gamma) \in \sim_{bp}$.

(1) By Prop. C.4.1(1) the second component of $jproc(\rho)$ gives a bijection for places of type synch. On the other hand, the first component of $jproc(\rho)$ provides a bijection for places of type switch: each switch place is represented by exactly one switch block, where switch blocks are exactly the blocks of cardinality one; furthermore, by Prop. 6.14.3(3) γ matches switch blocks against switch blocks.

(2a) Assume $b_1 \in blocks(r_1)$, $b_2 \in blocks(r_2)$ such that $\gamma(b_1) = b_2$. Bearing in mind the argumentation of (1) we obtain: if b_1 is of type switch $jproc(\rho)(b_1) = b_2$ follows directly from the definition; if b_1 is of type synch then this is immediate with Prop. C.4.1(3).

(2b) easily follows with (2a) and the definition of $syass(r.t)$. (cf. the definition of $\rightarrow_{bp}$ in Def. 6.14.2 and $\rightarrow_{cp}$ in Def. 6.4.2). $\square$

Now, it is straightforward to prove Lemma 6.14.1:

Proof of Lemma 6.14.1. Let $\rho \in \sim_{bp}$, and assume a bp bisimulation $\mathcal{B}$ such that $\rho \in \mathcal{B}$. With Prop C.4.2(1) and (2b) it is easy to check that $jproc(\mathcal{B})$ is a cp bisimulation. Since obviously $jproc(\rho) \in jproc(\mathcal{B})$, this implies $jproc(\rho) \in \sim_{cp}$. $\square$

C.4.2 Relating to Section 6.14.3

Part (1).

Proof of Prop. 6.14.7. Let t be given as above. If $t = \mathit{init}$ then we can assume r'' such that $r'' \in \sim_{hp}$, and otherwise we have r', r'' such that $r'.t.r'' \in \sim_{hp}$. In the first case set $r = \varepsilon$, and otherwise set $r = r'.t$. By prefix-closure of $\sim_{hp}$ it is clear that $r \in \sim_{hp}$.

Consider the following two statements:

1. $|t_1{}^\bullet| = |t_2{}^\bullet|$.

2. $\forall p_1 \in t_1{}^\bullet .\ \exists!\, p_2 \in t_2{}^\bullet .\ (p_1, p_2)$ satisfies the two conditions of $(*)$.

With (1) and (2) it is straightforward to exhibit β_{post} as required: (2) establishes that there exists exactly one function leading from $t_1{}^\bullet$ to $t_2{}^\bullet$ such that $(*)$ is satisfied. Moreover, when considering that $\mathcal{N}_1$ is spsd, it is easy to verify that this function must be injective. With (1) it is then clear that the unique function is indeed a unique bijection. We shall now prove that the statements (1) and (2) do hold.

(1.) Choose a set $T_1 \subseteq (t_1{}^\bullet)^\bullet$ such that $\forall p_1 \in t_1{}^\bullet .\ \exists! t_1^x \in T_1 \,.\ {}^\circ t_1^x = p_1$; this is possible due to Fact 6.2.2. Clearly, we have (A) $|T_1| = |t_1{}^\bullet|$. Let w_1 be a linearization of the transitions in T_1. It is easy to see that w_1 is a concurrent step at r_1. Then, since $r \in \sim_{hp}$ there must be a match for w_1 at r; that is we have w such that $r.w \in \sim_{hp}$ and $\mathit{proj}_1(w) = w_1$. Set $T_2 = \mathit{set}(\mathit{proj}_2(w))$. Clearly, we have (B) $|T_2| = |T_1|$. Furthermore, we can apply the following two insights to obtain more information about T_2: (1) from Prop. 6.14.5 we can derive that w_1 must be matched against transitions of $(t_2{}^\bullet)^\bullet$; (2) by Prop. 4.3.2(1) concurrent steps have to be matched against concurrent steps. (1) entails $T_2 \subseteq (t_2{}^\bullet)^\bullet$, which includes that each $t_2^x \in T_2$ must be of type switch; (2) implies $\forall t_2^a, t_2^b \in T_2 .\ t_2^a \neq t_2^b \implies {}^\bullet(t_2^a) \cap {}^\bullet(t_2^b) = \emptyset$. Together this means the preplace function gives an injective match from T_2 to $t_2{}^\bullet$; we have: (C) $|{}^\bullet T_2| = |T_2|$, and $(\star)$ ${}^\bullet T_2 \subseteq t_2{}^\bullet$. On the other hand, we can infer $(\ddagger)$ $t_2{}^\bullet \subseteq {}^\bullet T_2$ because otherwise we could exhibit a transition $t_2^x \in t_2{}^\bullet$ that is enabled at $r_2.w_2$ but cannot be matched at $r.w$ by $\mathcal{N}_1$: by Prop. 6.14.5 any $t_2^x \in t_2{}^\bullet$ has to be matched against a transition of $(t_1{}^\bullet)^\bullet$, but by choice of T_1 no such transition is available at $r.w$. Clearly, from $(\star)$ and $(\ddagger)$ we obtain (D) $|{}^\bullet T_2| = |t_2{}^\bullet|$. But altogether (A)-(D) proves $|t_2{}^\bullet| = |t_1{}^\bullet|$ as required.

(2.) Let $p_1 \in t_1{}^\bullet$. With the help of (1) we shall exhibit p_2 as required. Choose a set $T_1 \subseteq (t_1{}^\bullet \backslash \{p_1\})^\bullet$ such that $\forall p_1' \in (t_1{}^\bullet \backslash \{p_1\}) .\ \exists! t_1^x \in T_1 \,.\ {}^\circ t_1^x = p_1'$; again this is possible due to Fact 6.2.2. Clearly, we have (A) $|T_1| = |t_1{}^\bullet| - 1$. Let

w_1 be a linearization of T_1. Again, w_1 constitutes a concurrent step at r_1, and consequently there must be a match for w_1 at r in $\sim_{hp}$; that is there must be w such that $r.w \in \sim_{hp}$ and $proj_1(w) = w_1$. Set $T_2 = set(proj_2(w))$. Clearly, we have (B) $|T_1| = |T_2|$. By similar argumentation as above we further obtain (C) $|{}^\bullet T_2| = |T_2|$, and ($\star$) ${}^\bullet T_2 \subseteq t_2{}^\bullet$. From (1), (A), (B), and (C) we infer $|{}^\bullet T_2| = |t_2{}^\bullet| - 1$. Considering ($\star$) we then set p_2 such that $t_2{}^\bullet = {}^\bullet T_2 \cup \{p_2\}$; we will show that p_2 provides a place as required.

Consider the transitions of $p_1{}^\bullet$, and $p_2{}^\bullet$ respectively. It is easy to see that each of them will be enabled at $r.w$, and consequently each of them will have a match at $r.w$ in $\sim_{hp}$. Since p_1 is the only remaining postplace of t_1 at $r.w$ and the analogue is valid for p_2, with Prop. 6.14.5 it is clear that a transition of $p_1{}^\bullet$ must be matched by a transition of $p_2{}^\bullet$, and vice versa. But then, since matching is label-preserving, it is immediate that the pair (p_1, p_2) indeed satisfies the two conditions of $(*)$. Thus, it only remains to prove that p_2 is unique. But this immediately follows when considering that $\mathcal{N}_2$ is spsd. $\square$

Part (2). We proceed analogously to Appendix C.4.1.

Definition C.4.3. For all $r \in \sim_{hp}$ we inductively define a map *swass* to obtain a match between $switchP(r_1)$ and $switchP(r_2)$:

$$\begin{array}{lcl} swass(\varepsilon) & = & psd(init), \\ swass(r.t) & = & swass(r) \backslash \beta^{sw}_{pre} \cup \beta^{sw}_{post}, \end{array}$$

where

$$\beta^{sw}_{pre} = \begin{cases} \{(p_1, p_2) \in swass(r) \mid p_i \in {}^\bullet t_i \text{ for } i = 1, \text{ or } 2\} & \text{if } t \text{ is of type switch,} \\ \emptyset & \text{if } t \text{ is of type synch,} \end{cases}$$

and

$$\beta^{sw}_{post} = \begin{cases} psd(t) & \text{if } t \text{ is of type A,} \\ \emptyset & \text{if } t \text{ is of type B.} \end{cases}$$

Proposition C.4.3. *Let $r \in \sim_{hp}$.*

1. *$swass(r)$ is a bijection between $switchP(r_1)$ and $switchP(r_2)$.*

2. *$\forall p_1 \in switchP(r_1).\ swass(r)(p_1) = psd(t^e)(p_1),$ where $e = gen(r_1, p_1)$.*

3. *Let $t \in JT$ such that t_i is of type switch for $i = 1$, or 2.*

$$r \xrightarrow{t} \implies swass(r)({}^\bullet t_1) = {}^\bullet t_2.$$

Proof. Let $r \in \sim_{hp}$. We prove the three clauses by induction on the length of r 'in one go'.

Base case $r \equiv \varepsilon$*:* we have $swass(r) = psd(init)$. (1) is trivial since by definition $psd(init)$ is a bijection between $M_0^1 = switchP_1(\varepsilon)$ and $M_0^2 = switchP_2(\varepsilon)$. To see (2) consider that by definition we have: $\forall p_1 \in M_0^1.\ gen(\varepsilon, p_1) = init$, and $t^{init} = init$. (3) follows from the same facts and Prop. 6.14.9 when considering that ${}^\bullet t_1 \subseteq M_0^1$.

Inductive case $r \equiv r'.t^n$*:* By prefix-closure of $\sim_{hp}$ we have $r' \in \sim_{hp}$; hence, by induction hypothesis we obtain that the clauses (1) to (3) hold for r'. (1) is similar to the corresponding part in the proof of Prop. C.4.1; e.g. one employs the induction hypothesis of (3). In addition, here we consider that if t^n is of type A then $psd(t^n)$ will be defined and provide a bijection between $(t_1^n)^\bullet$ and $(t_2^n)^\bullet$. (2): again, this is analogous to Prop. C.4.1. Finally, (3) follows from (2) and Prop. 6.14.9. □

Definition C.4.4. For all $r \in \sim_{hp}$ we define a map *jproc* to translate r into a corresponding joint process: $jproc(r) = swass(r) \cup syass(r)$.

Proposition C.4.4. *Let* $r \in \sim_{hp}$*.*

1. $jproc(r) \in JProc$*.*
2. *For all* $t \in JT$ *we have:*
   (a) $r \xrightarrow{t} \implies jproc(r)({}^\bullet t_1) = {}^\bullet t_2$.
   (b) $r \xrightarrow{t} r.t \ \&\ r.t \in \sim_{hp} \implies jproc(r) \xrightarrow{t}_{cp} jproc(r.t)$.

Proof. (1) follows from Prop. C.4.1(1) and Prop. C.4.3(1). In turn, (2a) is a consequence of Prop. C.4.1(4) and Prop. C.4.3(3). Finally, (2b) is immediate with (2a) and the definition of $swass(r.t)$ and $syass(r.t)$ (cf. the definition of $\rightarrow_{cp}$ in Def. 6.4.2). □

Proof of Lemma 6.14.2. Analogously to Lemma 6.14.1 this is straightforward with Prop. C.4.4. □