Masaryk University, Brno
Faculty of Informatics

On Extensions of Process Rewrite Systems

Vojtěch Řehák

PhD Thesis

2007

Abstract

The thesis studies properties of Process Rewrite Systems (PRS) and their extensions. Namely, it introduces an extension of PRS, so called weakly extended Process Rewrite Systems (wPRS). This work compares the expressiveness of wPRS with original PRS classes and their known extensions. In addition, it studies decidability of problems related to model checking and other formal verification procedures such as weak and strong bisimulation, the reachability problem, etc. We aim to extend expressive power of known modelling facilities while preserving decidability and maintaining reasonable complexity bounds for problems related to (automatic) verification.

Disertační práce je zaměřena na vlastnosti procesových přepisovacích systémů (PRS) a jejich rozšíření. Práce představuje nové rozšíření hierarchie procesových přepisovacích systémů o slabou konečně stavovou jednotkou (wPRS) a porovnává vyjadřovací sílu těchto tříd s dosud známými rozšířeními. Dále práce studuje hranice rozhodnutelnosti zajímavých problémů vztahujících se k ověřování vlastností modelů, či jiným metodám formální verifikace. Ze zajímavých problémů jmenujme například problém dosažitelnosti, či problémy rozhodování silné a slabé bisimulace. Cílem práce je rozšířit vyjadřovací sílu známých modelů o přirozeně motivované možnosti, které však v rozumné míře zachovají rozhodnutelnost zmiňovaných problémů a složitostní odhady algoritmů řešících tyto problémy.

Acknowledgements

First of all I would like to thank my supervisor Mojmír Křetínský for his help, encouragement, collaboration, and many valuable and wide-ranging discussions during my studies. I am grateful to him also for careful reading a draft of this thesis.

I would like to thank Jan Strejček for his constant support and exemplary collaboration. I thank Laura Bozzelli for her collaboration on some LTL model checking problems. I also very much appreciate the collaboration with David Šafránek and David Antoš.

I would like to express my gratitude to Antonín Kučera, Jiří Srba, Hans Hüttel, and Luca Aceto for valuable suggestions, comments, and pointers. My thanks go to the members of Parallel and Distributed Systems Laboratory (ParaDiSe) and to the members of Institute for Theoretical Computer Science (ITI). I especially thank Jan Obržálek for his consultation regarding my English.

Finally, I would like to thank my wife Kateřina and my parents for their love, support, and patience.

Vojtěch Řehák

Contents

1 Introduction **1**
- 1.1 Subject of This Thesis 2
- 1.2 Thesis Organisation and Contribution 3
- 1.3 Author's Publications 5

2 Process Rewrite Systems (PRS) **9**
- 2.1 Basic Definitions 9
- 2.2 Definition of Process Rewrite Systems 12
- 2.3 Hierarchy of Process Rewrite Systems 14

3 State Extended PRS (sePRS) **19**
- 3.1 Definition of State Extended PRS 19
- 3.2 Hierarchy of State Extended PRS 20
  - 3.2.1 $PN \subseteq sePA$ 22
  - 3.2.2 Strictness and Incomparability 26

4 PRS with Finite Constraint Systems (fcPRS) **31**
- 4.1 Definition of PRS with Finite Constraint Systems 31
- 4.2 Hierarchy of PRS with Finite Constraint Systems 34

5 Weakly Extended PRS (wPRS) **41**
- 5.1 Definition of Weakly Extended PRS 41
- 5.2 Hierarchy of Weakly Extended PRS 43
  - 5.2.1 wBPP Non-bisimilar to any fcPAD 46
  - 5.2.2 PPDA Non-bisimilar to any wPAD 48
  - 5.2.3 wBPA Non-bisimilar to any fcPAN 49
  - 5.2.4 PDA Non-bisimilar to any wPAN 57
  - 5.2.5 Intuition and Conjectures 58

6 Strong Bisimulation Equivalence **61**
- 6.1 Motivation 61
- 6.2 Definition of Strong Bisimilarity 62
- 6.3 Summary 62

7 Weak Bisimulation Equivalence **65**
7.1 Motivation . . . 65
7.2 Definition of Weak Bisimilarity . . . 66
7.3 Undecidability of Weak Bisimilarity . . . 67
7.3.1 wBPA . . . 67
7.3.2 wBPP . . . 68
7.4 Weak Bisimilarity for More Restricted Classes . . . 73
7.4.1 Normed fcBPP . . . 73
7.4.2 Normed fcBPA . . . 74
7.5 Conclusion . . . 77

8 Reachability Problem **81**
8.1 Motivation . . . 81
8.2 Reachability Problem for wPRS . . . 82
8.3 Applications . . . 87
8.3.1 Model Checking Some Safety Properties . . . 87
8.3.2 Semi-decidability of Weak Trace Non-equivalence . . . 88
8.3.3 Other Applications . . . 90
8.4 Conclusion . . . 90

9 Branching Time Logics **93**
9.1 Motivation . . . 93
9.2 Logics and Studied Problems . . . 94
9.3 Reachability HM Property . . . 97
9.4 Corollaries and Remarks . . . 101
9.5 Evitability Simple Property for Deterministic BPP . . . 103
9.6 Summary . . . 104
9.7 Conclusion . . . 105

10 Linear Time Logic **109**
10.1 Motivation . . . 109
10.2 Definitions of the Studied Problems . . . 111
10.3 Fragment $\mathcal{A}$ and Translation of LTL($\mathsf{F}_s, \mathsf{G}_s$) into $\mathcal{A}$ . . . 117
10.4 Model Checking of wPRS against Negated $\mathcal{A}$ . . . 120
10.5 Model Checking LTL($\mathsf{F}_s, \mathsf{P}_s$) . . . 131
10.6 Undecidability Results . . . 140
10.7 Summary . . . 143
10.8 Conclusion . . . 146

11 LTL$^{\mathbf{det}}$ Model Checking **149**
11.1 Motivation . . . 149
11.2 Definition of the Studied Problem . . . 150
11.3 Proof Construction . . . 151
11.4 Summary . . . 155

12 Conclusion and Future Work **157**

Bibliography **159**

Chapter 1

Introduction

The need for formal modelling of systems and subsequent more or less automatic verification is indisputable. There have been many words written about model checking [CGP99] being a suitable automatic method for model verification.

When choosing a modelling formalism we need to have in mind two features going against each other. On the one hand, we want such a formalism to be expressive enough to model as much system behaviour as possible. This is the most important parameter for users of a modelling/verification tool. On the other hand, the more powerful the formalism is, the more complex verification algorithms are.

Basic model checking algorithms are based on exhaustive searching of a system state space. As the state space is usually huge, this approach leads to large computational requirements. A significant advancement was achieved by so called symbolic approach [McM93] that was successfully applied to hardware design verification. Unfortunately, the symbolic model checking did not achieve such great success in case of software verification. Currently, several interesting approaches are investigated in order to improve the applicability of model checking. The most useful technique is an application of abstraction to reduce the size of state space of the model. There are two basic approaches to abstraction, over-approximation and under-approximation. Over-approximation can possibly introduce new behaviour to the model, while under-approximation can possibly remove some valid behaviour of the modelled system. Hence, an over-approximation approach is suitable for proving correctness, since correctness of the abstract model implies correctness of the original system. On the other hand, an under-approximation approach works very well for bug hunting, as every incorrect behaviour in the abstract model has some, also incorrect, counterpart in the original system.

Even though software systems are usually of a finite size, they can be parametrised. The parameter can be maximal stack size, number of pro-

cesses running in parallel, etc. This quite naturally suggests to abstract away from these parameters and instead consider an infinite state system that is an over-approximation of the original one. In that case the state space increases its size, but the system can get more regular structure. Therefore focusing on the structure, rather than state by state exploration of the whole state space, can bring considerable improvement. The crucial difference follows from the fact that in such an abstract system the complexity of model checking depends on the size of the specification rather than on the size of the state space.

1.1 Subject of This Thesis

PRS In this thesis we focus on infinite state structures defined by a finite number of rules, which in turn induce an infinite transition relation of the system. We use the concept of Process Rewrite Systems (PRS) introduced by Mayr in his PhD thesis [May98]. Process rewrite systems are an elegant and unified approach that subsumes many of the important infinite state formalisms studied before, for example Petri nets (PN), pushdown automata (PDA), and various process algebras (BPA, BPP, PA). Process rewrite systems form a strict hierarchy with respect to strong bisimulation equivalence. This hierarchy is called the *PRS-hierarchy*. It is worth mentioning that the PRS-hierarchy subsumes hierarchies of infinite systems defined by Stirling, Caucal, and Moller [Cau92, Mol98].

sePRS It is well known that PA can model both parallel and sequential processes but without any communication among them. Let us mention that even the most general class of the PRS-hierarchy does not support an arbitrary kind of communication between processes. Some kind of global control is a very natural and useful instrument from the model designer point of view. Let us mention the success of PDA in modelling recursive programs or of PN in modelling dynamic creation of concurrent processes. In both these formalisms we have a notion of a *finite-state control unit (FSU)* which can keep global information and thus provide means of communication between modelled recursive programs or concurrent processes. Process rewrite systems extended with an FSU, called state-extended PRS, first appeared in [JKM01]. Unfortunately, using an FSU to extend the PRS rewriting mechanism is too powerful, since the reachability problem becomes undecidable even for a *state extended* version of *PA processes (sePA)* [BEH95].

fcPRS In [Str02], Strejček transferred some principles of Concurrent Constraint Programming (CCP) [SR90] to the formalism of process rewrite systems. The mechanism of rewrite systems is extended with a very restricted form of finite state unit. The unit behaves as a shared CCP *store* which is seen as a constraint on the values that common variables can represent. Strejček

has also shown that his definition of *PRS with finite constraint systems* (fcPRS) brings new classes lying strictly (with respect to strong bisimulation) between each original PRS class and its non-equivalent state extended counterpart – with one exception of PRS/sePRS, where the situation is not clear. Unfortunately, the finite constraint specification is quite complicated and so it is unhandy for modelling systems of processes. More precisely, it remains "too close" to its initial motivation (coming from CCP) to be useful for modelling different kind of FSU.

wPRS

In this thesis we introduce yet another extension lying between the finite constraint extension and the state extension. A definition of our extension is very close to the state extension and at the same time it keeps advantages of the less expressive classes (esp. decidability of the reachability problem). We define *weakly extended PRS* (wPRS) classes where the FSU is subject to some restrictions inspired by weak finite automata introduced in [MSS92]. (Here we use a nondeterministic automata rather than alternating ones.) Roughly speaking, weakly extended PRS formalism is a state extended PRS where the FSU contains no cycles except self loops; in other words, the state of FSU changes only finitely many times during an execution.

Weakly extended PRS formalism proves to be a very good compromise standing in between very expressible, but Turing powerful, sePRS and non-extended PRS lacking the advantage of global communication. Hence, this formalism provides the best comfort to system designers while still preserving some algorithmic verification abilities essential for developers of a potential verification tool. Moreover, the weak extension makes it easy to come up with some of the constructive proofs. It is worth mentioning that some of these proofs bring new results also for the non-extended PRS classes.

1.2 Thesis Organisation and Contribution

Besides introducing weakly extended PRS itself, we have proved the strictness of an *extended PRS-hierarchy* with respect to strong bisimulation equivalence. The extended PRS-hierarchy consists of all PRS, fcPRS, wPRS, and sePRS classes. The following four chapters are devoted to these (extended) PRS formalisms. Each chapter defines one of the formalisms and includes results proving strictness of the relations between the classes introduced so far. The results are published in [May98], [Str02], and our papers [KŘS04b] and [KŘS04a]. Some of the strictness problems are still open. In these cases we provide our conjectures supported by intuitive explanations only — see "Intuition and Conjectures" parts of Chapter 3, 4, and 5.

In the remaining chapters we focus on some interesting problems related to model checking and other formal verification procedures such

as weak and strong bisimulation checking. Chapter 6 reviews (un)decidability as well as complexity boundaries of strong bisimilarity between two states of a given extended PRS. As we have not reached any new results, this chapter only summarises all known results in this area.

Chapter 7 studies a weak bisimilarity problem. After a listing of all known results, we show our results of [KŘS06] stating that the problem is undecidable for weakly extended versions of BPA and BPP. We also strengthen the result by proving that the weak bisimilarity problem is undecidable even for normed subclasses of BPA and BPP extended with finite constraint systems.

A reachability problem (i.e. given two states α, β, the problem is whether the state β is reachable by some computation starting in the state α) can be considered as a basis for all model checking problems. In Chapter 8 we show that the reachability problem remains decidable for the class of wPRS and thus we determine the decidability borderline of the reachability problem in whole extended PRS-hierarchy. This result was published in [KŘS04a].

Chapter 9 is concentrated on the model checking problem for branching time logics. We deal with model checking for some simple branching time logics, namely EF and EG, and their fragments. We examine the problem whether a given weakly extended process rewrite system (wPRS) contains a reachable state satisfying a given formula of Hennessy–Milner logic. The main result of this chapter is that this problem is decidable. As a corollary we observe that also the problem of strong bisimilarity between wPRS and finite-state systems is decidable. Decidability of the same problem for wPRS subclasses, namely PAN and PRS, has been formulated as an open question. This chapter incorporates our contribution published in [KŘS05].

The model checking problem for Lineal Time Logic (LTL) is studied in Chapter 10. Most of the contents of this chapter was published in [BKŘS06]. We show that the LTL model checking problem is decidable for wPRS if we consider properties defined by formulae only with the *strict eventually* and *strict always* modalities. In the following section, we show that the model checking problem is decidable even for wPRS and LTL fragment based on modalities *strict eventually*, *strict always*, *eventually in the strict past* and *always in the strict past*. We establish the exact decidability border of model checking of wPRS classes and all basic modality fragments of LTL by showing that the model checking problems for PA and LTL(U), and for PA and LTL$(\overset{\infty}{\mathsf{F}}, \mathsf{X})$, are undecidable.

Chapter 11 is devoted to model checking problems for LTL$^{\text{det}}$ and extended process rewrite systems. LTL$^{\text{det}}$ (introduced in [Mai00]) is a common fragment of CTL and LTL. Using some results of the previous chapter we show that the model checking problem for wPRS and LTL$^{\text{det}}$ is decidable. Our results of this chapter have not been published yet.

1.3 Author's Publications

- Publications related to this PhD thesis

  - Journal papers

    [1] M. Křetínský, V. Řehák, and J. Strejček. Decidability of Reachability in Extended Process Rewrite Systems. *Information and Computation*, 2005. Submitted for publication.

    [2] M. Křetínský, V. Řehák, and J. Strejček. Refining the Undecidability Border of Weak Bisimilarity. In *Proceedings of the 7th International Workshop on Verification of Infinite-State Systems (INFINITY'05)*, volume 149 of *Electronic Notes in Theoretical Computer Science*, pages 17–36. Elsevier Science Publishers, 2006.

    [3] M. Křetínský, V. Řehák, and J. Strejček. On Extensions of Process Rewrite Systems: Rewrite Systems with Weak Finite-State Unit. In Philippe Schnoebelen, editor, *Proceedings of INFINITY 2003, the 5th International Workshop on Verification of Infinite-State Systems, a satellite workshop of CONCUR 2003*, volume 98 of *Electronic Notes in Theoretical Computer Science*, pages 75–88. Elsevier Science Publishers, 2004.

  - Conference and workshop papers

    [4] L. Bozzelli, M. Křetínský, V. Řehák, and J. Strejček. On Decidability of LTL Model Checking for Process Rewrite Systems. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, volume 4337 of *Lecture Notes in Computer Science*, pages 248–259. Springer-Verlag, 2006.

    [5] V. Řehák. Weakly Extended Process Rewrite Systems. In *MOVEP'06: 7th school on MOdeling and VErifying parallel Processes*, pages 360–364. Université de Bordeaux, 2006. Student's paper.

    [6] M. Křetínský, V. Řehák, and J. Strejček. Reachability of Hennessy–Milner Properties for Weakly Extended PRS. In R. Ramanujam and Sandeep Sen, editors, *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science*, volume 3821 of *Lecture Notes in Computer Science*, pages 213–224. Springer-Verlag, 2005.

    [7] M. Křetínský, V. Řehák, and J. Strejček. Refining the Undecidability Border of Weak Bisimilarity. In *Preliminary Proceedings of the 7th International Workshop on Verification of*

Infinite-State Systems (INFINITY'05), NS-05-4, BRICS Notes Series, pages 3–14. BRICS, 2005.

[8] V. Řehák. Reachability for Extended Process Rewrite Systems. In *MOVEP'04: 6th school on MOdeling and VErifying parallel Processes*, pages 77–82. Université Libre de Bruxelles, 2004. Student's paper.

[9] M. Křetínský, V. Řehák, and J. Strejček. Extended Process Rewrite Systems: Expressiveness and Reachability. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 – Concurrency Theory: 15th International Conference*, volume 3170 of *Lecture Notes in Computer Science*, pages 355–370. Springer-Verlag, 2004.

[10] M. Křetínský, V. Řehák, and J. Strejček. On Extensions of Process Rewrite Systems: Rewrite Systems with Weak Finite-State Unit. In *Preliminary Proceedings of the 5th International Workshop on Verification of Infinite-State Systems INFINTIY'2003*, pages 73–85. Les Universités à Marseille, 2003.

– Technical reports

[11] L. Bozzelli, M. Křetínský, V. Řehák, and J. Strejček. On Decidability of LTL Model Checking for Weakly Extended Process Rewrite Systems. Technical Report FIMU-RS-2006-05, 27pp, Faculty of Informatics, Masaryk University, Brno, 2006. Full version of FSTTCS 2006 paper.

[12] M. Křetínský, V. Řehák, and J. Strejček. Refining the Undecidability Border of Weak Bisimilarity. Technical Report FIMU-RS-2005-06, 20pp, Faculty of Informatics, Masaryk University, Brno, 2005. Full version of INFINITY 2005 paper.

[13] M. Křetínský, V. Řehák, and J. Strejček. On the expressive power of extended process rewrite systems. Technical Report RS-04-07, 12pp, Basic Research in Computer Science, Aarhus, Denmark, 2004.

[14] M. Křetínský, V. Řehák, and J. Strejček. Process Rewrite Systems with Weak Finite-State Unit. Technical Report FIMU-RS-2003-05, 23pp, Faculty of Informatics, Masaryk University, Brno, 2003. Full version of INFINITY 2003 paper.

• Publications related to Liberouter project

– Conference and workshop papers

[15] D. Antoš and V. Řehák. Routing, L2 Addressing, and Packet Filtering in a Hardware Engine. In *Proceedings of the 2nd Doctoral Wokrshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2006)*, Mikulov. pages 1–8. FIT BUT, 2006.

[16] P. Hlávka, V. Řehák, A. Smrčka, P. Šimeček, D. Šafránek, T. Vojnar. Formal Verification of the CRC Algorithm Properties. In *Proceedings of the 2nd Doctoral Wokrshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2006)*, Mikulov. pages 55–62. FIT BUT, 2006.

[17] A. Smrčka, V. Řehák, T. Vojnar, D. Šafránek, P. Matoušek, and Z. Řehák. Verifying VHDL Designs with Multiple Clocks in SMV. In *11th International Workshop on Formal Methods for Industrial Critical Systems FMICS 2006*, 2007. To appear in LNCS series.

[18] D. Antoš and V. Řehák. Routing and Level 2 Addressing in a Hardware Accelerator for Network Applications. In *ICT 2006, 13th International Conference on Telecommunications. Funchal, Portugal*, pages 1–4. Universidade da Madeira, 2006.

[19] D. Antoš, V. Řehák, and P. Holub. Packet Filtering for FPGA-Based Routing Accelerator. In *CESNET Conference 2006*, pages 161–173. CESNET, 2006.

[20] T. Kratochvíla, V. Řehák, and D. Šafránek. Formal Verification of a FIFO Component in Design of Network Monitoring Hardware. In *CESNET Conference 2006*, pages 151–160. CESNET, 2006.

[21] D. Antoš, V. Řehák, and J. Kořenek. Hardware Router's Lookup Machine and its Formal Verification. In *3rd International Conference on Networking ICN'2004 Conference Proceedings*. Gosier, Guadeloupe, French Caribbean, pages 1002–1007. University of Haute Alsace, Colmar, France, 2004.

[22] D. Antoš, J. Kořenek, and V. Řehák. Lookups in IPv6 router implemented in an FPGA. In *EurOpen, Proceedings of the 23rd conference*, Strážnice, Czech Republic, pages 91–102, 2003. (in Czech).

– Technical reports

[23] P. Hlávka, T. Kratochvíla, V. Řehák, D. Šafránek, P. Šimeček, and T. Vojnar. CRC64 Algorithm Analysis and Verification. Technical Report 27, 7pp, CESNET, December 2005.

[24] J. Holeček, T. Kratochvíla, V. Řehák, D. Šafránek, and P. Šimeček. Verification Process of Hardware Design in Liberouter Project. Technical Report 05, 7pp, CESNET, November 2004.

[25] J. Holeček, T. Kratochvíla, V. Řehák, D. Šafránek, and P. Šimeček. How to Formalize FPGA Hardware Design. Technical Report 04, 11pp, CESNET, October 2004.

[26] J. Holeček, T. Kratochvíla, V. Řehák, D. Šafránek, and P. Šimeček. Verification Results in Liberouter Project. Technical Report 03, 32pp, CESNET, September 2004.

[27] T. Kratochvíla, V. Řehák, and P. Šimeček. Verification of COMBO6 VHDL Design. Technical Report 17, 17pp, CESNET, November 2003.

[28] D. Antoš, J. Kořenek, K. Minaříková, and V. Řehák. Packet header matching in Combo6 IPv6 router. Technical Report 01, 15pp, CESNET, January 2003.

[29] J. Barnat, T. Brázdil, P. Krčál, V. Řehák, and D. Šafránek. Model Checking in IPv6 Hardware Router Design. Technical Report 07, 8pp, CESNET, July 2002.

- Publication related to the author master thesis

[30] V. Řehák. $\oplus$-OBDD in Symbolic Model Checking. In M. Bielikova, editor, *Proceedings of SOFSEM'02 Student Research Forum*, pages 41–46, 2002.

Chapter 2

Process Rewrite Systems (PRS)

In this chapter we recall a definition of Process Rewrite Systems (PRS). PRS were introduced in [May97b] and a compact summary of the topic is nicely presented in [May98]. In Section 2.1 we introduce some basic notions behind are needed to define PRS in Section 2.2. In the next section we compare an expressive power of the defined PRS formalisms with respect to the strong bisimulation equivalence.

2.1 Basic Definitions

Labelled Transition System

We use a common notion of Labelled Transition System (LTS) for semantic definitions of infinite state systems. LTS is a well known formalism defined as follows.

Definition 2.1. *A* labelled transition system (LTS) $\mathcal{L}$ *is defined as a tuple* $(S, Act, \longrightarrow, \alpha_0)$*, where*

- S *is a set of* states *or* processes*,*
- Act *is a set of* atomic actions *or* labels*,*
- $\longrightarrow \subseteq (S \times Act \times S)$ *is a* transition relation*,*
- $\alpha_0 \in S$ *is a distinguished* initial state*.*

We write $\alpha \xrightarrow{a} \beta$ *instead of* $(\alpha, a, \beta) \in \longrightarrow$*. A state* $\alpha \in S$ *is* terminal *(or* deadlocked*, written* $\alpha \not\longrightarrow$*) if there is no* $a \in Act$ *and* $\beta \in S$ *such that* $\alpha \xrightarrow{a} \beta$*.*

The transition relation $\longrightarrow$ can be in a homomorphic way extended to finite sequences of actions $\sigma \in Act^*$ so as to write $\alpha \xrightarrow{\varepsilon}{}^* \alpha$ and $\alpha \xrightarrow{a\sigma}{}^* \beta$ whenever $\alpha \xrightarrow{a} \gamma \xrightarrow{\sigma}{}^* \beta$ for some state γ. A state β is *reachable from a state* α, written $\alpha \longrightarrow^* \beta$, if there is $\sigma \in Act^*$ such that $\alpha \xrightarrow{\sigma}{}^* \beta$. We say that a state is *reachable* if it is reachable from the initial state.

Equivalences on LTS

In this thesis we define some classes of infinite state systems and naturally we would like to compare their expressive power. For the sake of the comparison an equivalence on the systems has to be chosen. We take into account isomorphism, strong bisimulation, and trace equivalence (also known as language equivalence).

An *isomorphism* is the strongest relation of those mentioned above.

Definition 2.2. *A binary relation R on set of states S is an* isomorphism *if and only if it is a bijection and for each $(\alpha, \beta) \in R$ the following conditions hold:*

- $\forall\, \alpha' \in S, a \in Act : \alpha \xrightarrow{a} \alpha'$ *implies* $(\exists \beta' \in S : \beta \xrightarrow{a} \beta' \wedge (\alpha', \beta') \in R)$
- $\forall\, \beta' \in S, a \in Act : \beta \xrightarrow{a} \beta'$ *implies* $(\exists \alpha' \in S : \alpha \xrightarrow{a} \alpha' \wedge (\alpha', \beta') \in R)$

If an isomorphism can be constructed between the initial states of two labelled transition systems then we say those labelled transition systems are isomorphic.

The strong bisimulation is the second strongest relation of those we consider here. In context of studying process behaviour we are not interested in whether the relation between the systems is a bijection or not. Therefore the strong bisimulation does not have to be a bijection.

Definition 2.3. *A binary relation R on set of states S is a* (strong) bisimulation *[Mil89] if and only if for each $(\alpha, \beta) \in R$ the following conditions hold:*

- $\forall\, \alpha' \in S, a \in Act : \alpha \xrightarrow{a} \alpha'$ *implies* $(\exists \beta' \in S : \beta \xrightarrow{a} \beta' \wedge (\alpha', \beta') \in R)$
- $\forall\, \beta' \in S, a \in Act : \beta \xrightarrow{a} \beta'$ *implies* $(\exists \alpha' \in S : \alpha \xrightarrow{a} \alpha' \wedge (\alpha', \beta') \in R)$

(Strong) bisimulation equivalence *on an assumed LTS is the maximal bisimulation (i.e. union of all bisimulations). If a bisimulation can be constructed between the initial states of two labelled transition systems then we say those labelled transition systems are* (strongly) bisimilar.

In the case of trace equivalence we abstract away from the branching structure of a system and compare all possible executions only.

Definition 2.4. *A binary relation R on set of states S is a* trace equivalence *if and only if for each $(\alpha, \beta) \in R$ the following conditions hold:*

- $\forall\, \alpha' \in S, w \in Act^* : \alpha \xrightarrow{w}{}^* \alpha'$ *implies* $(\exists \beta' \in S : \beta \xrightarrow{w}{}^* \beta')$
- $\forall\, \beta' \in S, w \in Act^* : \beta \xrightarrow{w}{}^* \beta'$ *implies* $(\exists \alpha' \in S : \alpha \xrightarrow{w}{}^* \alpha')$

If a trace equivalence can be constructed between the initial states of two labelled transition systems then we say those labelled transition systems are trace equivalent.

As we work with systems describing processes in the thesis, we put the emphasise on strong bisimulation equivalence. We refer to [vG93] for more equivalences on processes.

Process Terms

The concept of Process Rewrite Systems (PRS) formalism is based on rewriting of process terms. Therefore, being able to define PRS, we have to define process terms at fist.

Definition 2.5. *Let* $Const = \{X, \ldots\}$ *be a countably infinite set of* process constants. *The set* $\mathcal{T}$ *of* process terms *(ranged over by* $t, \ldots$*) is defined by the abstract syntax*

$$t ::= \varepsilon \mid X \mid t_1.t_2 \mid t_1 \| t_2,$$

where

- ε *is the empty term,*
- $X \in Const$ *is a process constant (used as an atomic process),*
- *'*$\|$*' mean a parallel composition, and*
- *'.' mean a sequential composition.*

We always work with equivalence classes of terms modulo commutativity and associativity of '$\|$*' and modulo associativity of '.' We also define* $\varepsilon.t = t = t.\varepsilon$ *and* $t \| \varepsilon = t$.

For every process term t, we define a set $Const(t)$ to be the set of all constants occurring in the process term t.

For the sake of the following definition we define four classes of process terms depending on whether the parallel composition and/or the sequential composition has been used in it.

Definition 2.6. *We distinguish four* classes of process terms *as:*

"1" *terms consisting of a single process constant only (i.e.* $\varepsilon \notin$ **1***),*

$$t ::= X$$

"S" sequential *terms without parallel composition, e.g.* $X.Y.Z$*,*

$$t ::= \varepsilon \mid X \mid t_1.t_2$$

"P" parallel *terms without sequential composition. e.g.* $X \| Y \| Z$*,*

$$t ::= \varepsilon \mid X \mid t_1 \| t_2$$

"G" general *terms with arbitrarily nested sequential and parallel compositions.*

$$t ::= \varepsilon \mid X \mid t_1.t_2 \mid t_1 \| t_2$$

2.2 Definition of Process Rewrite Systems

In this section, we recall the definition of Process Rewrite Systems (PRS) as introduced by Mayr in his Ph.D. thesis [May98].

Definition 2.7. *Let $Act = \{a, b, \cdots\}$ be a countably infinite set of* atomic actions *and $\alpha, \beta \in \{1, S, P, G\}$ such that $\alpha \subseteq \beta$. An* (α, β)-PRS (process rewrite system) *is a pair $\Delta = (R, t_0)$, where*

- *$R \subseteq ((\alpha \setminus \{\varepsilon\}) \times Act \times \beta)$ is a finite set of* rewrite rules*, and*
- *$t_0 \in \beta$ is an* initial term*.*

We write $(t_1 \overset{a}{\hookrightarrow} t_2) \in R$ instead of $(t_1, a, t_2) \in R$.

Let Δ be a PRS as defined in Definition 2.7. We define $Const(\Delta)$ as the set of all constants occurring in the rewrite rules of Δ or in its initial state, and $Act(\Delta)$ as the set of all actions occurring in the rewrite rules of Δ. We sometimes write $(t_1 \overset{a}{\hookrightarrow} t_2) \in \Delta$ instead of $(t_1 \overset{a}{\hookrightarrow} t_2) \in R$.

Definition 2.8. *The semantics of Δ is given by the LTS $(S, Act(\Delta), \longrightarrow_\Delta, t_0)$, where*

- *the set of states $S = \{t \in \beta \mid Const(t) \subseteq Const(\Delta)\}$,*
- *the initial state of LTS t_0 is formed by the initial term of Δ, and*
- *the transition relation $\longrightarrow_\Delta$ is the least relation satisfying the following inference rules for all $t_1, t_2, t \in S$ and $a \in Act(\Delta)$.*

$$\frac{(t_1 \overset{a}{\hookrightarrow} t_2) \in \Delta}{t_1 \overset{a}{\longrightarrow}_\Delta t_2} \qquad \frac{t_1 \overset{a}{\longrightarrow}_\Delta t_2}{t_1 \| t \overset{a}{\longrightarrow}_\Delta t_2 \| t} \qquad \frac{t_1 \overset{a}{\longrightarrow}_\Delta t_2}{t_1 . t \overset{a}{\longrightarrow}_\Delta t_2 . t}$$

Note that parallel composition is commutative and, thus, the inference rule for parallel composition also holds with t_1 and t exchanged.

If no confusion arises, we sometimes speak about a "process rewrite system" meaning the "labelled transition system generated by a process rewrite system".

We use $\longrightarrow_\Delta$ or $\longrightarrow_R$ to emphasise that we mean the transition relation generated by a set of rules R and we also write $\longrightarrow$ instead of $\longrightarrow_\Delta$ if Δ is clear from the context.

Remark 2.9. *As there are only finitely many terms t_i and atomic actions a_i such that $t_0 \overset{a_i}{\longrightarrow}_\Delta t_i$, it can be assumed without loss of generality that the initial state t_0 of an (α, β)-PRS is a single constant, say X_0.*

Every pair $\alpha, \beta \in \{1, S, P, G\}$ induces a class of all labelled transition systems that can be expressed by an (α, β)-PRS. Some of the classes correspond to LTS classes of widely known models. In what follows we quote an apt descriptions of PRS classes as it was published in [Str02]:

- $(1, 1)$-PRS are equivalent to finite-state systems (FS). Every process constant corresponds to a state and the state space is bounded by $|Const(\Delta)|$. Every finite-state system can be encoded as a $(1, 1)$-PRS.

- $(1, S)$-PRS are equivalent to Basic Process Algebra processes (BPA) defined in [BK85], which are the transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are allowed.

- $(1, P)$-PRS are equivalent to communication-free nets, the subclass of Petri nets where every transition has exactly one place in its preset [BE97]. This class of Petri nets is equivalent to Basic Parallel Processes (BPP) [CHM93].

- $(1, G)$-PRS are equivalent to PA-processes, Process Algebras with sequential and parallel composition, but no communication (see [BK85] for details).

- It is easy to see that pushdown automata can be encoded as a subclass of (S, S)-PRS (with at most two constants on the left-hand side of rules). Caucal [Cau92] showed that any unrestricted (S, S)-PRS can be presented as a pushdown automaton (PDA), in the sense that the transition systems are isomorphic up to the labelling of states. Thus (S, S)-PRS are equivalent to pushdown processes (which are the processes described by pushdown automata).

- (P, P)-PRS are equivalent to Petri nets (PN). Every constant corresponds to a place in the net and the number of occurrences of a constant in a term corresponds to the number of tokens in this place. This is because we work with classes of terms modulo commutativity of parallel composition. Every rule in Δ corresponds to a transition in the net.

- (S, G)-PRS is the smallest common generalisation of pushdown processes and PA-processes. They are called *PAD* (PA + PDA) in [May97b].

- (P, G)-PRS are called PAN-processes in [May97a]. It is the smallest common generalisation of Petri nets and PA-processes and it strictly subsumes both of them (e.g. PAN can describe all Chomsky-2 languages while Petri nets cannot).

- The most general case is (G, G)-PRS (here simply called PRS). PRS have been introduced in [May97b]. They subsume all of the previously mentioned classes.

For some other intuition into the topic of PRS we refer to a very comprehensible introduction published by Esparza [Esp02] or Mayr's thesis [May98].

2.3 Hierarchy of Process Rewrite Systems

Figure 2.1 describes a hierarchy of (α, β)-PRS classes with respect to strong bisimulation equivalence (bisimilarity). We call this hierarchy the *PRS-hierarchy*. More precisely, the classes of PRS systems are interpreted as the sets of their underlying labelled transition systems. The depicted hierarchy is then the upward oriented Hasse diagram of a partial order relation '$\subseteq$' between these sets of labelled transition systems modulo bisimulation equivalence. In other words, a line connecting X and Y with Y placed higher than X means that every transition systems definable in X can be (up to bisimulation equivalence) defined in Y while the reverse does not hold – we write $X \subsetneq Y$. Moreover, the classes that are not connected by any sequence of upward going lines are incomparable.

The strictness of the hierarchy with respect to strong bisimulation equivalence follows from the results presented (or cited) in [BCS96, Mol96, May00], where the following systems are presented:

- **a BPP system which is not bisimilar to any PDA system**,
- **a PN system which is not bisimilar to any PAD system**,
- **a BPA system which is not bisimilar to any PN system**, and
- **a PDA system which is not bisimilar to any PAN system**.

These systems imply strictness of all relations of the PRS-hierarchy. Moreover, they also show incomparability of all non-related classes of the PRS-hierarchy. In what follows we exemplify the four aforementioned systems. To make references to these systems human readable, we assign intuitive acronyms to the systems; e.g. the system of Example 2.10 is called **BPP non-PDA** as it represents a **BPP** system which is **not** bisimilar to any **PDA** system.

Example 2.10. (BPP non-PDA) *A BPP system with the initial state X that is not bisimilar to any PDA system (for the proof see [Mol96]).*

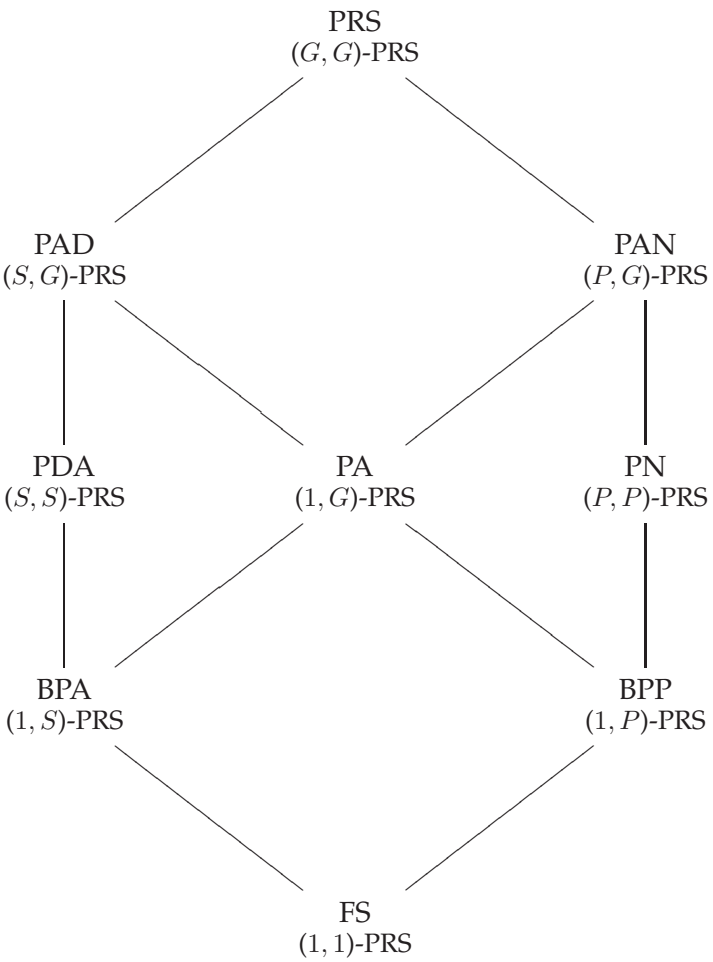


Figure 2.1: The PRS-hierarchy

$$X \overset{a}{\hookrightarrow} X\|B \qquad X \overset{c}{\hookrightarrow} X\|D \qquad X \overset{e}{\hookrightarrow} \varepsilon \qquad B \overset{b}{\hookrightarrow} \varepsilon \qquad D \overset{d}{\hookrightarrow} \varepsilon$$

It follows from Example 2.10 that classes BPP, PA, and PAD are not contained ($\not\subseteq$) in any of classes FS, BPA, and PDA. We write

$$\text{BPP}, \text{PA}, \text{PAD} \not\subseteq \text{FS}, \text{BPA}, \text{PDA}.$$

Example 2.11. (PN non-PAD) *A Petri net given as (P, P)-PRS with the initial state $X\|A\|B$ that is not bisimilar to any PAD process (for the proof see [May00]).*

$$\begin{array}{llll} X \overset{g}{\hookrightarrow} X\|A\|B & Y\|A \overset{a}{\hookrightarrow} Y & X\|A \overset{d}{\hookrightarrow} Z & Y\|A \overset{d}{\hookrightarrow} Z \\ X \overset{c}{\hookrightarrow} Y & Y\|B \overset{b}{\hookrightarrow} Y & X\|B \overset{d}{\hookrightarrow} Z & Y\|B \overset{d}{\hookrightarrow} Z \end{array}$$

Intuitively, the system models two counters running in parallel with a synchronised increasing (action g) and a synchronised switching to decreasing phase (action c) and deadlock state (action d).

It follows from Example 2.11 that classes PN, PAN, and PRS are not contained ($\not\subseteq$) in any of classes BPP, PA, and PAD. We write

$$\text{PN}, \text{PAN}, \text{PRS} \not\subseteq \text{BPP}, \text{PA}, \text{PAD}.$$

Example 2.12. (BPA non-PN) *A BPA system with the initial state X that is not bisimilar to any PN system (for the proof see [Mol96]).*

$$X \overset{a}{\hookrightarrow} X.A \qquad X \overset{b}{\hookrightarrow} X.B \qquad X \overset{c}{\hookrightarrow} \varepsilon \qquad A \overset{a}{\hookrightarrow} \varepsilon \qquad B \overset{b}{\hookrightarrow} \varepsilon$$

It follows from Example 2.12 that classes BPA, PA, and PAN are not contained ($\not\subseteq$) in any of classes FS, BPP, and PN. We write

$$\text{BPA}, \text{PA}, \text{PAN} \not\subseteq \text{FS}, \text{BPP}, \text{PN}.$$

Example 2.13. (PDA non-PAN) *A PDA system with the initial state $U.X$ that is not bisimilar to any PAN system (for the proof see [May00]).*

$$\begin{array}{lll} U.X \overset{a}{\hookrightarrow} U.A.X & U.A \overset{a}{\hookrightarrow} U.A.A & U.B \overset{a}{\hookrightarrow} U.A.B \\ U.X \overset{b}{\hookrightarrow} U.B.X & U.A \overset{b}{\hookrightarrow} U.B.A & U.B \overset{b}{\hookrightarrow} U.B.B \\ U.X \overset{c}{\hookrightarrow} V.X & U.A \overset{c}{\hookrightarrow} V.A & U.B \overset{c}{\hookrightarrow} V.B \\ U.X \overset{d}{\hookrightarrow} W.X & U.A \overset{d}{\hookrightarrow} W.A & U.B \overset{d}{\hookrightarrow} W.B \\ V.X \overset{e}{\hookrightarrow} V & V.A \overset{a}{\hookrightarrow} V & V.B \overset{b}{\hookrightarrow} V \\ W.X \overset{f}{\hookrightarrow} W & W.A \overset{a}{\hookrightarrow} W & W.B \overset{b}{\hookrightarrow} W \end{array}$$

Intuitively, the system behaves as a pushdown automaton with control states U, V, W and stack alphabet $\{X, A, B\}$. Symbol X marks the bottom of the stack and symbols A and B save sequential information.

It follows from Example 2.13 that classes PDA, PAD, and PRS are not contained ($\not\subseteq$) in any of classes BPA, PA, and PAN. We write

$$\mathrm{PDA}, \mathrm{PAD}, \mathrm{PRS} \not\subseteq \mathrm{BPA}, \mathrm{PA}, \mathrm{PAN}.$$

Concerning the other equivalences, let us note that strictness of the PRS-hierarchy with respect to strong bisimulation implies strictness also with respect to isomorphism. Contrary to this, the PRS-hierarchy is not strict with respect to trace equivalence, e.g. both BPA and PDA define the class of ε-free context-free languages.

Chapter 3

State Extended PRS (sePRS)

Most research (with some recent exceptions, e.g. [BT03, Esp02]) has been devoted to the PRS classes from the lower part of the PRS-hierarchy depicted in Figure 2.1, especially to pushdown processes (PDA), Petri nets (PN) and their respective subclasses. Let us recall the successes of PDA in modelling recursive programs with value passing from callee to caller (without process creation), PN in modelling dynamic creation of communicating concurrent processes (without recursive calls), and *communicating pushdown systems (CPDS)* [BET03] as an example of modelling both features. All of these formalisms subsume a notion of a finite-state control unit keeping some kind of global information which is accessible to the redexes (the components that can be reduced) of a PRS term – hence a finite-state control unit (FSU) can regulate rewriting.

3.1 Definition of State Extended PRS

In this section, we introduce an extension of process rewrite system formalism that enriches every PRS system with an FSU, as given in [JKM01]. We call these systems *state extended PRS* and define them as follows.

Definition 3.1. *Let* $Act = \{a, b, \cdots\}$ *be a countably infinite set of* atomic actions *and* $\alpha, \beta \in \{1, S, P, G\}$ *such that* $\alpha \subseteq \beta$. *An* (α, β)-sePRS Δ *is a tuple* (M, R, m_0, t_0), *where*

- M *is a finite set of* states *of a* control unit,
- $R \subseteq ((M \times (\alpha \setminus \{\varepsilon\})) \times Act \times (M \times \beta))$ *is a finite set of* rewrite rules,
- $m_0 \in M$ *is an* initial state *of the control unit, and*
- $t_0 \in \beta$ *is an* initial term.

We write $((m, t_1) \overset{a}{\hookrightarrow} (n, t_2)) \in R$ *instead of* $((m, t_1), a, (n, t_2)) \in R$.

To shorten our notation we write mt instead of (m, t). As in the PRS case, instead of $(mt_1 \overset{a}{\hookrightarrow} nt_2) \in R$ where $\Delta = (M, R, m_0, t_0)$, we usually write $(mt_1 \overset{a}{\hookrightarrow} nt_2) \in \Delta$. The meaning of $Const(\Delta)$ (process constants used in rewrite rules or in t_0) and $Act(\Delta)$ (actions occurring in rewrite rules) for a given state extended PRS Δ is also the same as in the PRS case. Moreover, we use $M(\Delta)$ to denote the finite set of control states which occur in Δ.

Definition 3.2. *The semantics of a state extended (α, β)-PRS system Δ is given by the corresponding labelled transition system $(S, Act(\Delta), \longrightarrow_\Delta, m_0t_0)$, where*

- $S = M(\Delta) \times \{t \in \beta \mid Const(t) \subseteq Const(\Delta)\}$,
- *m_0t_0 is the initial state composed of the initial control state m_0 of the control unit $M(\Delta)$ and the initial term t_0, and*
- *the transition relation $\longrightarrow_\Delta$ is defined as the least relation satisfying the following inference rules for all $mt_1, nt_2, m(t_1 \| t), m(t_2 \| t), n(t_1.t), n(t_2.t) \in S$ and $a \in Act(\Delta)$.*

$$\frac{(mt_1 \overset{a}{\hookrightarrow} nt_2) \in \Delta}{mt_1 \overset{a}{\longrightarrow}_\Delta nt_2} \qquad \frac{mt_1 \overset{a}{\longrightarrow}_\Delta nt_2}{m(t_1 \| t) \overset{a}{\longrightarrow}_\Delta n(t_2 \| t)} \qquad \frac{mt_1 \overset{a}{\longrightarrow}_\Delta nt_2}{m(t_1.t) \overset{a}{\longrightarrow}_\Delta n(t_2.t)}$$

Note that parallel composition is commutative and, thus, the inference rule for parallel composition also holds with t_1 and t exchanged.

Instead of $(1, 1)$-sePRS, $(1, S)$-sePRS, $(1, P)$-sePRS, (S, S)-sePRS, $(1, G)$-sePRS, (P, P)-sePRS, (S, G)-sePRS, (P, G)-sePRS, and (G, G)-sePRS we use a more natural notation seFS, seBPA, seBPP, sePDA, sePA, sePN, sePAD, sePAN, and sePRS respectively. The class seBPP is also known as *multiset automata (MSA)* or *parallel pushdown automata (PPDA)*, see [Mol96].

3.2 Hierarchy of State Extended PRS

In this section we present an overview of expressive power of the defined classes. Figure 3.1 shows a graphical description of the PRS-hierarchy enriched by the state extended (α, β)-PRS classes. We call the hierarchy the *state extended PRS-hierarchy*. The dotted lines represent the relations where the strictness is only our conjecture.

An immediate observation is that the classes FS, PDA, and PN have exactly (i.e. up to isomorphism) the same expressive power as the corresponding state extended classes. We note some other trivial observations also even up to isomorphism:

X $\subseteq$ Y implies seX $\subseteq$ seY for every two classes X and Y of PRS

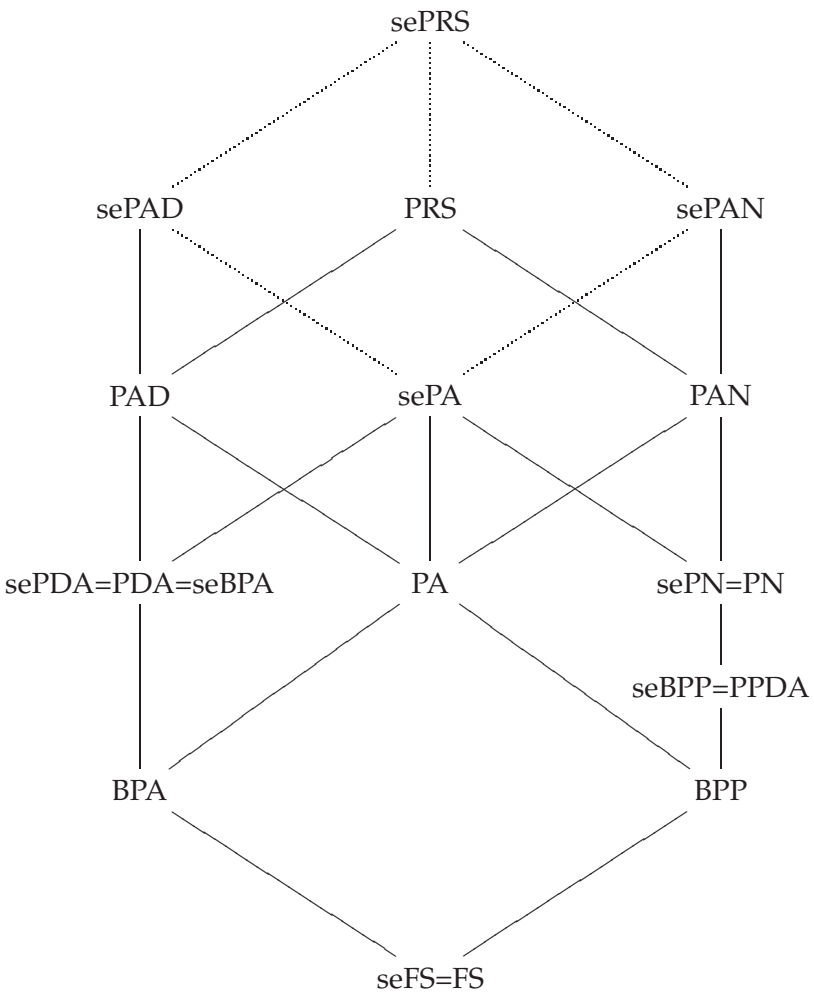


Figure 3.1: The state extended PRS-hierarchy

$$(\alpha, \beta)\text{-PRS} \subseteq (\alpha, \beta)\text{-sePRS for all } (\alpha, \beta)\text{-PRS.}$$

Looking at two lines leaving sePA down to the left and down to the right, we note the "left-part collapse" of (S, S)-PRS and PDA proved by Caucal [Cau92] (up to isomorphism). The right-part counterpart is slightly different due to Moller's result establishing PPDA $\subsetneq$ PN [Mol98]. Therefore, the previous trivial observations imply only PPDA $\subseteq$ sePA and the relation between PN and sePA left open. In Subsection 3.2.1 we prove that PN $\subseteq$ sePA. In Subsection 3.2.2 we discuss strictness ('$\subsetneq$') of the state extended PRS-hierarchy.

3.2.1 $PN \subseteq sePA$

Here we show that Petri nets are less expressive (with respect to the strong bisimulation equivalence) than state-extended PA processes. Let Δ be a Petri net and $Const(\Delta)$ be a set of k process constants $\{X_1, \ldots, X_k\}$. Speaking about PN systems in context of PRS, X^n denotes a parallel composition of n copies of X. In this subsection we consider Petri nets in the traditional "place-transition" setting where "markings" are the counterparts of the states of LTSs, see e.g. [Pet81, Rei85]. Let the Petri net Δ have k places $P_1, \ldots, P_K$, each state

$$X_1^{p_1} \| \ldots \| X_k^{p_k}$$

of the PN Δ is called *marking* and it is written as

$$(p_1, \ldots, p_k)$$

where p_i is the number of *tokens* at the *place* P_i. Any rewrite rule

$$X_1^{l_1} \| \ldots \| X_k^{l_k} \stackrel{a}{\hookrightarrow} X_1^{r_1} \| \ldots \| X_k^{r_k}$$

(where $l_i, r_i \geq 0$) is called *transition* and it is written as

$$(l_1, \ldots, l_k) \stackrel{a}{\hookrightarrow} (r_1, \ldots, r_k).$$

As we employ the marking-place description of PN, X^n stands only for a sequential composition of n copies of X in the rest of the subsection .

The heart of our argument is a construction of an sePA Δ' that is bisimilar to the given PN Δ. The main difficulty in this construction is to maintain the number of tokens at the places of a PN. To this end, we may use two types of sePA memory:

- a finite control (FSU), which cannot represent an unbounded counter,
- a term of an unbounded length, where just one constant can be rewritten in one step.

Intuition

Our construction of an sePA Δ' can be reformulated on intuitive level as follows. Let a marking $(p_1, \ldots, p_k)$ mean‘ that we have p_i units of the i-th currency, $i = 1, \ldots, k$. An application of a PN transition

$$(l_1, \ldots, l_k) \overset{a}{\hookrightarrow} (r_1, \ldots, r_k)$$

has an effect of a currency exchange from p_i to $p_i - l_i + r_i$ for all i.

An sePA reseller Δ' will have k finite pockets (in its FSU) and k bank accounts (a parallel composition of k sequential terms t_i). The reseller Δ' maintains an invariant $p_i = pocket_i + account_i$ for all i. To mimic a PN transition he must obey sePA rules, i.e. he may use all his pockets, but just one of his accounts in one exchange — transition.

A solution is to do $pocket_i \leftrightarrow account_i$ transfers cyclically, $i = 1, \ldots, k$. Hence, rebalancing $pocket_i$ the reseller Δ' must be able to perform the next $k - 1$ exchanges (while visiting the other accounts) without accessing $account_i$. Therefore, Δ' needs sufficiently large (but finite) pockets and sufficiently high (and still fixed) limits for $pocket_i \leftrightarrow account_i$ transfers. In what follows we show that these bounds exist.

Bounds

In one step the amount of the i-th currency can be changed at most by

$$L_i = max\,\{\, l_i, r_i \mid (l_1, \ldots, l_k) \overset{a}{\hookrightarrow} (r_1, \ldots, r_k) \text{ is a PN transition}\},$$

thus the upper bound for the total effect of k consecutive steps can be set up to

$$M_i = k \cdot L_i.$$

Any rebalancing of the i-th pocket sets its value into

$$\{M_i, \ldots, 2M_i - 1\}$$

(or into $\{0, \ldots, 2M_i - 1\}$ if $account_i$ is empty). Hence, after k transitions the value of $pocket_i$ is in

$$\{0, \ldots, 3M_i - 1\}.$$

Then the next rebalancing takes place and $account_i$ is increased or decreased (if it is not empty) by M_i to achieve the (rebalanced) value of $pocket_i$ in $\{M_i, \ldots, 2M_i - 1\}$.

Construction

Each state of sePA Δ' consists of a state of an FSU and a term (parallel composition of k stacks, in fact just counters, representing accounts). Each state of the FSU is a member of the following product.

$$\underset{update\ controller}{\{1,\ldots,k\}} \quad \times \quad \underset{pocket_1}{\{0,\ldots,3M_1-1\}} \quad \times \quad \ldots \quad \times \quad \underset{pocket_k}{\{0,\ldots,3M_k-1\}}$$

The $update\ controller$ goes in round-robin fashion on its range and refers to the account being updated (rebalanced) in the next step. The value of each $pocket_i$ (subsequently denoted by m_i) is equal to the number of tokens at P_i counted modulo M_i.

We define $2k$ process constants $B_i, X_i \in Const(\Delta')$, where $i = 1,\ldots,k$. The i-th stack t_i is of the form

$$X_i^n.B_i$$

where $n \geq 0$, B_i represents the bottom of the i-th stack, and each X_i represents M_i tokens at place P_i.

Given an initial marking $\alpha_0 = (p_1,\ldots,p_k)$ of Δ, we construct the initial state

$$\beta_0 = (1, m_1,\ldots,m_k)\, t_1 \| \cdots \| t_k$$

of the sePA Δ', where denoting $n_i = max(0, (p_i \text{ div } M_i) - 1)$ we put $m_i = p_i - n_i \cdot M_i$ and $t_i = X_i^{n_i}.B_i$. In other words we have $p_i = m_i + n_i \cdot M_i$ and moreover $m_i \in \{M_i,\ldots,2M_i-1\}$ if p_i is big enough (i.e. $p_i \geq M_i$).

To each transition $(l_1,\ldots,l_k) \overset{a}{\hookrightarrow} (r_1,\ldots,r_k)$ of PN Δ we construct the set of sePA rules

$$(s, m_1,\ldots,m_k)\, t \overset{a}{\hookrightarrow} (s', m_1',\ldots,m_k')\, t'$$

such that they obey the following conditions:

- Update controller conditions:
  - $s, s' \in \{1,\ldots,k\}$ and
  - $s' = (s \mod k) + 1$.
- General conditions for pockets ($1 \leq i \leq k$):
  - $m_i, m_i' \in \{0,\ldots,3M_i-1\}$,
  - $m_i \geq l_i$ (i.e. the transition "is enabled"), and
  - if $i \neq s$ then $m_i' = m_i - l_i + r_i$.
- The case $i = s$ means to specify m_s' and the terms t, t' that acts the pocket-account rebalancing transaction. These are given by the rules of the following table. The first two *Bottom rules* are the rules for working with the empty stack. The next three *Top rules* describe the rewriting of process constant X_s. Depending on the value of $\overline{m_s} = m_s - l_s + r_s$, there are *dec*, *inc*, and *basic* variants manipulating the s-th stack.

Rule	t	$\overline{m_s} \in$	m'_s	t'
Bottom-basic rule	B_s	$\{0, \ldots, 2M_s - 1\}$	$\overline{m_s}$	B_s
Bottom-inc rule	B_s	$\{2M_s, \ldots, 3M_s - 1\}$	$\overline{m_s} - M_s$	$X_s.B_s$
Top-dec rule	X_s	$\{0, \ldots, M_s - 1\}$	$\overline{m_s} + M_s$	ε
Top-basic rule	X_s	$\{M_s, \ldots, 2M_s - 1\}$	$\overline{m_s}$	X_s
Top-inc rule	X_s	$\{2M_s, \ldots, 3M_s - 1\}$	$\overline{m_s} - M_s$	$X_s.X_s$

Theorem 3.3. *PN $\subseteq$ sePA with respect to bisimulation equivalence.*

Proof. Let Δ be an arbitrary PN with an initial marking α_0. According to the construction given above we build the sePA Δ' with the initial state β_0. In the rest of the proof we show that a relation

$$\begin{aligned} R = \{(\alpha, \beta) \mid & \text{ marking } \alpha \text{ is } (p_1, \ldots, p_k) \\ & \wedge\ \beta = (s, m_1, \ldots, m_k)\, X_1^{n_1}.B_1 \| \ldots \| X_k^{n_k}.B_k \\ & \wedge\ s \in \{1, \ldots, k\} \\ & \wedge \text{ for all } i = 1, \ldots, k \text{ it holds that} \\ & \quad p_i = m_i + n_i \cdot M_i \\ & \quad \wedge\ m_i < 2M_i + (s - i \mod k) L_i \\ & \quad \wedge \text{ if } n_i > 0 \text{ then } M_i - (s - i \mod k) L_i \leq m_i \\ & \qquad\qquad \text{else } 0 \leq m_i \ \} \end{aligned}$$

is a bisimulation between Δ and Δ'.

It follows directly from the construction that the pair (α_0, β_0) of the initial states is in R.

We follow the definition of bisimulation to prove that the relation R is a bisimulation. Let $\alpha = (p_1, \ldots, p_k)$ be a marking of PN Δ and

$$\beta = (s, m_1, \ldots, m_k)\, X_1^{n_1}.B_1 \| \ldots \| X_k^{n_k}.B_k$$

be a state of sePA Δ' such that $(\alpha, \beta) \in R$.

Let us assume that a transition $(l_1, \ldots, l_k) \stackrel{a}{\hookrightarrow} (r_1, \ldots, r_k)$ is fired in α and leads to $\alpha' = (p'_1, \ldots, p'_k)$, i.e. $p_i \geq l_i$ and $p'_i = p_i - l_i + r_i$, for all $i = 1, \ldots, k$. According to the definition of bisimulation, we will show that there is also a state β' of Δ' and a transition with a label a leading from β to β' such that $(\alpha', \beta') \in R$. Looking into the construction it is easy to see that such transition exists if $m_i \geq l_i$ for all $i = 1, \ldots, k$. These inequalities can be easily proved as follows. For each $i = 1, \ldots, k$, we discus two cases:

- If $n_i = 0$ then $(\alpha, \beta) \in R$ implies $p_i = m_i + n_i \cdot M_i = m_i$. This, together with $p_i \geq l_i$, directly leads to the desired $m_i \geq l_i$.
- If $n_i > 0$ then $(\alpha, \beta) \in R$ implies $M_i - (s - i \mod k) L_i \leq m_i$. As M_i is defined to be equal to $k \cdot L_i$, we get that $m_i \geq L_i$. Now, the definition of L_i implies $L_i \geq l_i$ that directly results in $m_i \geq l_i$.

It remains to show that $(\alpha', \beta') \in R$. This can be obtained by a straightforward inspection through the definitions of all the rule types.

The symmetric case, starting with a transition from β, proceeds in a similar way. Hence, Δ and Δ' are bisimilar and so PN $\subseteq$ sePA in our notation. □

We note that the sePA system constructed by our algorithm does not need to be isomorphic to the original PN system, e.g. due to the different values of the update controller.

3.2.2 Strictness and Incomparability

To show the strictness ('$\subsetneq$') of all relations of the state extended PRS-hierarchy and incomparability of all non-related classes of the state extended PRS-hierarchy the following examples have to be presented:

- a BPP system which is not bisimilar to any PDA system,
- **a PPDA system which is not bisimilar to any PAD system**,
- **a PN system which is not bisimilar to any PPDA system**,
- **a PAN system which is not bisimilar to any sePAD system**,
- a BPA system which is not bisimilar to any PN system,
- a PDA system which is not bisimilar to any PAN system,
- **a PAD system which is not bisimilar to any sePAN system**, and
- **an sePA system which is not bisimilar to any PRS system**.

For the non-bold items we simply refer to Section 2.3, where the systems are exemplified.

Concerning the first bold item we focus on the PN non-PAD system of Example 2.11. In terms of traditional Petri net notation, we can say that the places X, Y, and Z are 1-bounded[1] and the places A and B are unbounded.[2] As there is no communication[3] between the unbounded places A and B, it is well known that the PN system can be redefined as a PPDA system. Using this we have constructed the PPDA of Example 3.4. It is easy to see that the PPDA system of Example 3.4 is isomorphic to the PN system of Example 2.11.

[1] A place is *1-bounded* if it contains at most one token in each reachable marking.
[2] A place is *unbounded* if there is no bound on the number of tokens.
[3] Two places *communicate* if they are input places of the same transition.

Example 3.4. (PPDA non-PAD) *A PPDA given as* $(1, P)$*-sePRS with the initial state* $xC\|A\|B$ *that is not bisimilar to any PAD process (it follows from Example 2.11).*

$$xC \overset{g}{\hookrightarrow} xC\|A\|B \qquad yA \overset{a}{\hookrightarrow} y\varepsilon \qquad xA \overset{d}{\hookrightarrow} z\varepsilon \qquad yA \overset{d}{\hookrightarrow} z\varepsilon$$
$$xC \overset{c}{\hookrightarrow} yC \qquad yB \overset{b}{\hookrightarrow} y\varepsilon \qquad xB \overset{d}{\hookrightarrow} z\varepsilon \qquad yB \overset{d}{\hookrightarrow} z\varepsilon$$

It follows from Example 3.4 that classes PPDA, sePA, and sePAD are not contained ($\not\subseteq$) in any of classes BPP, PA, and PAD. We write

$$\text{PPDA}, \text{sePA}, \text{sePAD} \not\subseteq \text{BPP}, \text{PA}, \text{PAD}.$$

The system of the second item written in bold (PN non-PPDA system) was conjectured in [Mol96] and proved in [Mol98] later on. The PN system of this papers is presented in the following example.

Example 3.5. (PN non-PPDA) *A PN given as* (P, P)*-PRS with the initial state* X *that is not bisimilar to any PPDA process (for the proof see [Mol98]).*

$$X \overset{a}{\hookrightarrow} X\|A \qquad X\|A\|B \overset{c}{\hookrightarrow} X \qquad Y\|A \overset{a}{\hookrightarrow} Y$$
$$X \overset{b}{\hookrightarrow} X\|B \qquad X \overset{d}{\hookrightarrow} Y \qquad Y\|A \overset{a}{\hookrightarrow} Y$$

It follows from Example 3.5 that

$$\text{PN} \not\subseteq \text{PPDA}.$$

Concerning language equivalence, we mention that the PN languages can be fully characterised in terms of PPDA. This result on language equivalence between PN and PPDA classes has not been published but only presented by Hirshfeld in his talk "Methods and tools for the verification of infinite state systems" in Grenoble, March 97 [Hir].

To finish the strictness and incomparability proofs of the state extended PRS-hierarchy (with respect to strong bisimulation equivalence), it remains to show a PAD non-sePAN system, a PAN non-sePAD system, and an sePA non-PRS system. Unfortunately, according to our best knowledge, the existence of these systems have not been proved yet. Therefore, five lines remain dotted in the state extended PRS-hierarchy. In the following, we give an intuition that supports our conjecture that all the relations are strict.

Intuition and Conjectures

Let us provide some intuition to qualify differences between the classes of of the state extended PRS-hierarchy.

First, let us discus modelling abilities implied by terms forming right-hand sides of PRS rules. A sequential composition on the right-hand side

allows us to model sequential execution of other processes; hence an arbitrary long sequential information can be stored. A parallel composition on the right-hand side of rules enables to store information as multisets.

Composed terms (i.e. terms not belonging to 1 class) on the left-hand sides of rules upgrade the formalism from "context-free" level onto "context-sensitive" one. This lifting serves to model communication between modelled processes; in particular, the process behaviour is sensitive to its sequential and/or parallel context. The sequential "context-sensitivity" can solve a problem how to sent a return value to the parent process (e.g. $X.P \longrightarrow P''$ instead of $X \longrightarrow \varepsilon$). The parallel operator on the left-hand side of rules can serve for synchronisation (and value passing) between parallel siblings. For example BPP=$(1, P)$-PRS systems, also called "communication free Petri nets," represent a context-free formalism whereas PN=(P, P)-PRS systems form its context-sensitive counterpart.

Extensions with a control finite-state unit (FSU) bring further possibilities to increase expressive power of a system. On the one hand, such extensions are similar to the left-hand side extensions because it brings communication between different subterms, but on the other hand an FSU controls the whole of rewritten term and so we suggest that it cannot completely supply the local communication. The global communication abilities coincide with the local ones in the case of BPA only, where all communicating process terms are in one location and so $(1, S)$-sePRS=seBPA=PDA=(S, S)-PRS. We conjecture that in more powerful classes (such as PA, PAD, PAN, PRS) there are state-extensions orthogonal to the PRS-hierarchy. In the following example we present systems that supports our conjecture.

Example 3.6. (PAD non-sePAN) *A PAD system with the initial state Y that is not bisimilar to any sePAN (according to our conjecture).*

$$
\begin{array}{lll}
Y \overset{g}{\hookrightarrow} (U.X) \| Y & & \\
U.X \overset{a}{\hookrightarrow} U.A.X & U.A \overset{a}{\hookrightarrow} U.A.A & U.B \overset{a}{\hookrightarrow} U.A.B \\
U.X \overset{b}{\hookrightarrow} U.B.X & U.A \overset{b}{\hookrightarrow} U.B.A & U.B \overset{b}{\hookrightarrow} U.B.B \\
U.X \overset{c}{\hookrightarrow} V.X & U.A \overset{c}{\hookrightarrow} V.A & U.B \overset{c}{\hookrightarrow} V.B \\
U.X \overset{d}{\hookrightarrow} W.X & U.A \overset{d}{\hookrightarrow} W.A & U.B \overset{d}{\hookrightarrow} W.B \\
V.X \overset{e}{\hookrightarrow} V & V.A \overset{a}{\hookrightarrow} V & V.B \overset{b}{\hookrightarrow} V \\
W.X \overset{f}{\hookrightarrow} W & W.A \overset{a}{\hookrightarrow} W & W.B \overset{b}{\hookrightarrow} W
\end{array}
$$

Intuitively, the system behaves as arbitrary many PDA non-PAN systems (Example 2.13) running in parallel. Each such a system needs to store an unbounded piece of sequential information, hence a sequential operator has to be employed. Moreover, the PAD non-sePAN system consists of an unbounded number of such systems running in parallel. As the term part is the only unbounded part of every state of sePAN system, we suppose that the term part of a desired sePAN has to be

organised in a similar way as in the case of the examined PAD, i.e. an unbounded number of unbounded sequential terms connected by parallel operators.

In what follows we want to indicate that an FSU extension cannot fully supply the communication over a sequential operator in this case. The most significant "communication over a sequential operator" is to carry some information during decreasing of the sequential stack, i.e. changing the process constants "bellow" a sequential operator during erasing the "topmost" process constant. In an sePAN system, every rewrite rule rewrites only parallel subterms, hence, such a caring of information has to be divided into more then one step. The problem is that there is an unbounded number of stacks, and so in a subsequent step we cannot remember which stack was decreased in the previous step. Contrary to the technique used in the proof showing PN⊆sePA (see Subsection 3.2.1), where the number of stacks is bounded, the stacks of the speculated sePAN cannot be distinguished using different process constants. Moreover, in the PN⊆sePA construction, FSU stores a reference to the stack rebalanced in the last step. Here, the reference can be arbitrary large and so it does not fit into any FSU.

Example 3.7. (PAN non-sePAD) *A PAN system with the initial state Y that is not bisimilar to any sePAD (according to our conjecture).*

$$Y \overset{g}{\hookrightarrow} ((X\|A\|B).U)\|Y$$

$$X \overset{g}{\hookrightarrow} X\|A\|B \qquad Y\|A \overset{a}{\hookrightarrow} Y \qquad X\|A \overset{d}{\hookrightarrow} Z \qquad Y\|A \overset{d}{\hookrightarrow} Z$$

$$X \overset{c}{\hookrightarrow} Y \qquad Y\|B \overset{b}{\hookrightarrow} Y \qquad X\|B \overset{d}{\hookrightarrow} Z \qquad Y\|B \overset{d}{\hookrightarrow} Z$$

The system behaves as arbitrary many PN non-PAD systems (Example 2.11) running in parallel. The term construction ".U" forms a communication barrier between the respective Petri net systems.

Similarly to the previous case, we conjecture that unbounded number of unbounded Petri net implies that the term part of the system has to be constructed in the same way as in this example regardless of which state extended PRS formalism we use. Therefore, using the same argumentation as above, we conclude that the local synchronisation in the subsystems cannot be supplied be the global communication via an FSU.

The last system is necessary for the incomparability result only. It implies, among others, that sePA $\not\subseteq$ PRS.

Example 3.8. (sePA non-PRS) *An sePA system with the initial state $qX\|Y$ that is not bisimilar to any PRS (according to our conjecture).*

$$qX \overset{a}{\hookrightarrow} qX.A \qquad pY \overset{c}{\hookrightarrow} pY.C \qquad oA \overset{a'}{\hookrightarrow} o\varepsilon$$

$$qX \overset{b}{\hookrightarrow} qX.B \qquad pY \overset{y}{\hookrightarrow} o\varepsilon \qquad oB \overset{b'}{\hookrightarrow} o\varepsilon$$

$$qY \overset{c}{\hookrightarrow} qY.C \qquad\qquad oC \overset{c'}{\hookrightarrow} o\varepsilon$$

$$qX \overset{x}{\hookrightarrow} p\varepsilon$$

It is easy to see that the system is isomorphic to the system of Example 4.9. It was conjectured that there is no PRS bisimilar to the system. In [Str02] there is an intuitive argumentation that supports the conjecture (see Example 4.9 for further details).

Chapter 4

PRS with Finite Constraint Systems (fcPRS)

In this chapter we recall an extension of process rewrite systems with finite constraint systems (fcPRS). This extension has been introduced by Strejček in [Str02]. A motivation comes from the theory of constraint systems used in *concurrent constraint programming* (CCP) – see e.g. [SR90]. Contrary to the state extended PRS, fcPRS employs a restricted finite state unit. The unit behaves as a shared CCP *store* which is seen as a set of constraints on the values that common variables can represent.

4.1 Definition of PRS with Finite Constraint Systems

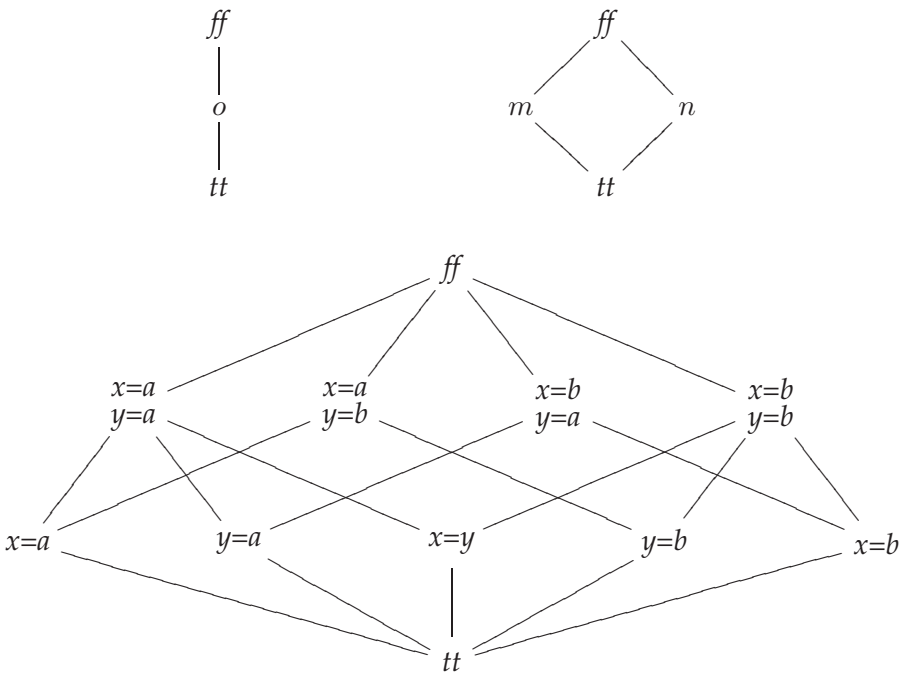
In the following definition we define a constraint system as a set of constraints with a structure of an algebraic lattice. A constraint system describes a state space and possible evolution of a CCP store unit.

Definition 4.1. *A* constraint system *is a bounded lattice* $(C, \vdash, \wedge, tt, ff)$*, where*

- C *is a set of* constraints*,*
- $\vdash$ *(called* entailment*) is an ordering on this set,*
- $\wedge$ *is the least upper bound operation, and*
- tt, ff *(*true *and* false*) are the least and the greatest elements of* C *respectively* ($ff \vdash tt$ *and* $tt \neq ff$*).*

In algebra, the symbol $\wedge$ usually denotes the greatest lower bound operation, while the least upper bound operation is rather marked with symbol $\vee$. Here we use a notation of CCP where the least upper bound corresponds to logical conjunction.

Example 4.2. *Examples of constraint systems.*



Using the notion of finite constraint systems we can define process rewrite systems with finite constraint system in the following definition.

Definition 4.3. *Let* $Act = \{a, b, \cdots\}$ *be a countably infinite set of* atomic actions *and* $\alpha, \beta \in \{1, S, P, G\}$ *such that* $\alpha \subseteq \beta$. *An* (α, β)-fcPRS (PRS with finite constraint system) *is a tuple* $\Delta = (\mathcal{C}, R, tt, t_0)$, *where*

- $\mathcal{C} = (C, \vdash, \wedge, tt, ff)$ *is a finite constraint system describing the* store*; the elements of* C *represent* values of the store,
- $R \subseteq ((C \times (\alpha \setminus \{\varepsilon\})) \times Act \times (C \times \beta))$ *is a finite set of* rewrite rules,
- tt *is the least element of* C, *and*
- $t_0 \in \beta$ *is a distinguished* initial process term.

We write $((m, t_1) \stackrel{a}{\hookrightarrow} (n, t_2)) \in R$ *instead of* $((m, t_1), a, (n, t_2)) \in R$.

To shorten our notation we write mt instead of (m, t). As in the PRS case, instead of $(mt_1 \stackrel{a}{\hookrightarrow} nt_2) \in R$ where $\Delta = (\mathcal{C}, R, tt, t_0)$, we usually write $(mt_1 \stackrel{a}{\hookrightarrow} nt_2) \in \Delta$. The meaning of $Const(\Delta)$ (process constants used in rewrite rules or in t_0), $Act(\Delta)$ (actions occurring in rewrite rules), and $M(\Delta)$ (values of the store occurring in Δ) for a given fcPRS Δ is the same as in the sePRS case.

Definition 4.4. *The semantics of an (α, β)-fcPRS system $\Delta = (\mathcal{C}, R, tt, t_0)$ is given by the LTS $(S, Act(\Delta), \longrightarrow_\Delta, tt\ t_0)$, where*

- *$S = (M(\Delta) \setminus \{ff\}) \times \{t \in \beta \mid Const(t) \subseteq Const(\Delta)\}$ is the set of states,*
- *$tt\ t_0$ is the initial state composed of the least element tt of the store $M(\Delta)$ and the initial term t_0, and*
- *the transition relation $\longrightarrow_\Delta$ is defined as the least relation satisfying the following inference rules for all mt_1, nt_2, ot_1, $(o \wedge n)t_2$, $m(t_1 \| t)$, $m(t_2 \| t)$, $n(t_1.t)$, $n(t_2.t) \in S$ and $a \in Act(\Delta)$.*

$$\frac{(mt_1 \overset{a}{\hookrightarrow} nt_2) \in \Delta}{ot_1 \overset{a}{\longrightarrow}_\Delta (o \wedge n)t_2} \quad \text{if } o \vdash m \text{ and } o \wedge n \neq ff$$

$$\frac{mt_1 \overset{a}{\longrightarrow}_\Delta nt_2}{m(t_1 \| t) \overset{a}{\longrightarrow}_\Delta n(t_2 \| t)} \qquad \frac{mt_1 \overset{a}{\longrightarrow}_\Delta nt_2}{m(t_1.t) \overset{a}{\longrightarrow}_\Delta n(t_2.t)}$$

Note that parallel composition is commutative and, thus, the inference rule for parallel composition also holds with t_1 and t exchanged.

The two side conditions in the first inference rule are very close to principles used in CCP. The first one ($o \vdash m$) ensures the rule $(mt_1 \overset{a}{\hookrightarrow} nt_2) \in \Delta$ can be used only if the current value of the store *o entails m* (it is similar to *ask(m)* in CCP). The second condition ($o \wedge n \neq ff$) guarantees that the store stays *consistent* after application of the rule (analogous to the consistency requirement when processing *tell(n)* in CCP).

An important observation is that the value of a store can move in a lattice only upwards as $o \wedge n$ always entails o, i.e. $o \wedge n$ is greater than or equal to o. Intuitively, partial information can only be added to the store, but never retracted. We say that the store is *monotonic*.

Let o be a store value of an fcPRS state and $(mt_1 \overset{a}{\hookrightarrow} nt_2) \in \Delta$ be a rule that can be applied on the state. The fcPRS definition says that $o \vdash m$ and $o \wedge n \neq ff$ where $o \wedge n$ is the store value of the state reached by the application. It follows from the monotonicity of the store that, for every subsequent value p of the store, $p \vdash m$. Moreover, associativity, commutativity, and idempotence of the least upper bound operation $\wedge$ imply that $p \wedge n = p$ for every subsequent store value p, and so every rule can change the store only during the first application. To sum up, we note that $p \vdash m$ and $p \wedge n \neq ff$ for every subsequent store value p, i.e. *any further application of a rule cannot be forbidden by a value of the store*.

Instead of $(1,1)$-fcPRS, $(1,S)$-fcPRS, $(1,P)$-fcPRS, (S,S)-fcPRS, $(1,G)$-fcPRS, (P,P)-fcPRS, (S,G)-fcPRS, (P,G)-fcPRS, and (G,G)-fcPRS we use a more natural notation fcFS, fcBPA, fcBPP, fcPDA, fcPA, fcPN, fcPAD, fcPAN, and fcPRS respectively.

4.2 Hierarchy of PRS with Finite Constraint Systems

In this section we present an overview of expressive power of the defined classes. Figure 4.1 shows a graphical description of sePRS-hierarchy enriched by (α, β)-fcPRS classes. The hierarchy is constructed with respect to strong bisimulation. The dotted lines represent the relations where the strictness is only conjectured.

It is an easy observation that

$$(\alpha, \beta)\text{-PRS} \subseteq (\alpha, \beta)\text{-fcPRS} \subseteq (\alpha, \beta)\text{-sePRS}$$

holds for (α, β)-PRS class (even up to isomorphism). Moreover, we note that for every two classes X and Y of PRS

$$\text{X} \subseteq \text{Y implies fcX} \subseteq \text{fcY}.$$

The strictness of the relations depicted in Figure 4.1 follows from the following examples:

- a BPP system which is not bisimilar to any PDA system,
- **a fcBPP system which is not bisimilar to any PAD system**,
- **a PPDA system which is not bisimilar to any fcPAD system**,
- a PN system which is not bisimilar to any PPDA system,
- a PAN system which is not bisimilar to any sePAD system,

- a BPA system which is not bisimilar to any PN system,
- **a fcBPA system which is not bisimilar to any PAN system**,
- **a PDA system which is not bisimilar to any fcPAN system**,
- a PAD system which is not bisimilar to any sePAN system,

- **a fcPA system which is not bisimilar to any PRS system**, and
- **a sePA system which is not bisimilar to any fcPRS system**.

For the bold items we simply refer to the previous sections where the systems are presented. In what follows the non-bold systems are exemplified.

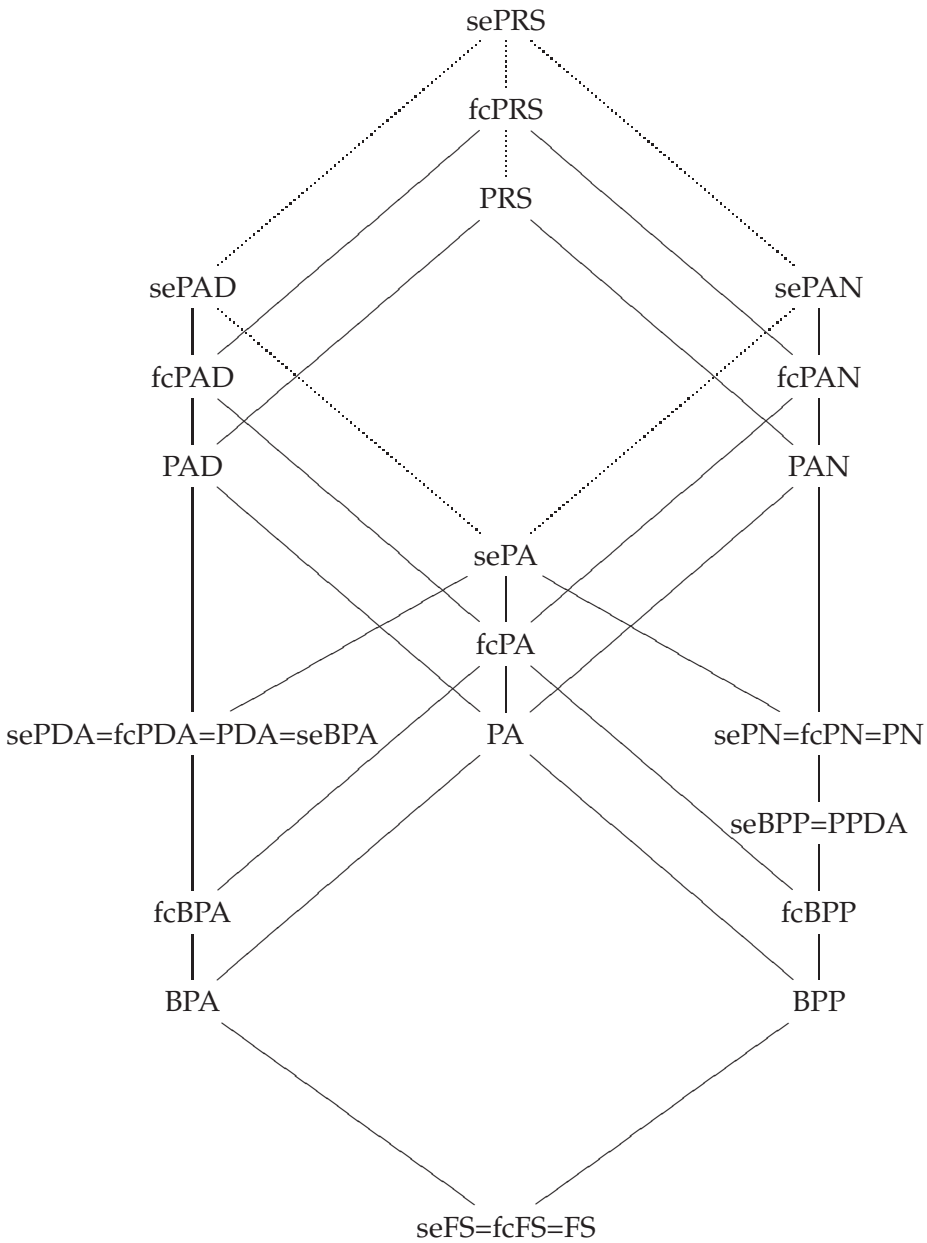
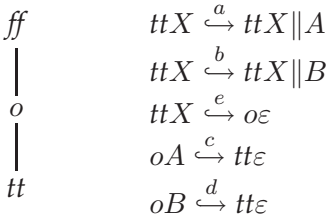


Figure 4.1: The fcPRS classes and the state extended PRS-hierarchy

Example 4.5. (fcBPP non-PAD) *Let us consider a fcBPP system given as an* $(1, P)$*-fcPRS with the constraint system depicted below and the initial process term* X*.*



$$ttX \overset{a}{\hookrightarrow} ttX \| A$$
$$ttX \overset{b}{\hookrightarrow} ttX \| B$$
$$ttX \overset{e}{\hookrightarrow} o\varepsilon$$
$$oA \overset{c}{\hookrightarrow} tt\varepsilon$$
$$oB \overset{d}{\hookrightarrow} tt\varepsilon$$

There is no PAD system bisimilar to the considered fcBPP system (for the proof see [Str02]).

It follows from Example 4.5 that classes fcBPP, fcPA, and fcPAD are not contained ($\not\subseteq$) in any of classes BPP, PA, and PAD. We write

$$\text{fcBPP}, \text{fcPA}, \text{fcPAD} \not\subseteq \text{BPP}, \text{PA}, \text{PAD}.$$

Example 4.6. (PPDA non-fcPAD) *Let* Δ *be a PPDA process with the initial state* $xC \| A \| B$ *and the following rewrite rules:*

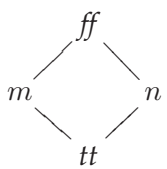
$$xC \overset{g}{\hookrightarrow} xC \| A \| B \qquad yA \overset{a}{\hookrightarrow} y\varepsilon \qquad xA \overset{d}{\hookrightarrow} z\varepsilon \qquad yA \overset{d}{\hookrightarrow} z\varepsilon$$
$$xC \overset{c}{\hookrightarrow} yC \qquad yB \overset{b}{\hookrightarrow} y\varepsilon \qquad xB \overset{d}{\hookrightarrow} z\varepsilon \qquad yB \overset{d}{\hookrightarrow} z\varepsilon$$
$$yC \overset{e}{\hookrightarrow} zC \qquad zA \overset{e}{\hookrightarrow} z\varepsilon \qquad zB \overset{e}{\hookrightarrow} z\varepsilon \qquad zC \overset{r}{\hookrightarrow} xC \| A \| B$$

The system Δ *is the same as the PPDA non-wPAD of Example 5.10. Hence, there is no fcPAD system bisimilar to the considered PPDA* Δ *(the proof directly follows from Lemma 5.11).*

It follows from Example 4.6 that classes PPDA, sePA, and sePAD are not contained ($\not\subseteq$) in any of classes fcBPP, fcPA, and fcPAD. We write

$$\text{PPDA}, \text{sePA}, \text{sePAD} \not\subseteq \text{fcBPP}, \text{fcPA}, \text{fcPAD}.$$

Example 4.7. (fcBPA non-PAN) *Let* Δ *be a fcBPA of the form* $(\mathcal{C}, R, tt, U.X)$*, where* $\mathcal{C}$ *and* R *are as follows.*



$$ttU \overset{a}{\hookrightarrow} ttU.A \qquad ttA \overset{a}{\hookrightarrow} tt\varepsilon$$
$$ttU \overset{b}{\hookrightarrow} ttU.B \qquad ttB \overset{b}{\hookrightarrow} tt\varepsilon$$
$$ttU \overset{c}{\hookrightarrow} m\varepsilon \qquad mX \overset{e}{\hookrightarrow} tt\varepsilon$$
$$ttU \overset{b}{\hookrightarrow} n\varepsilon \qquad nX \overset{f}{\hookrightarrow} tt\varepsilon$$

The system Δ *is not bisimilar to any BPA (for the proof see [Str02]).*

It follows from Example 4.7 that classes fcBPA, fcPA, and fcPAN are not contained ($\not\subseteq$) in any of classes BPA, PA, and PAN. We write

$$\text{fcBPA}, \text{fcPA}, \text{fcPAN} \not\subseteq \text{BPA}, \text{PA}, \text{PAN}.$$

Example 4.8. (PDA non-fcPAN) *A PDA system with the initial state $U.X.Y$ that is not bisimilar to any fcPAN (for the proof see [Str02]).*

$$
\begin{array}{lll}
U.X \overset{a}{\hookrightarrow} U.A.X & U.A \overset{a}{\hookrightarrow} U.A.A & U.B \overset{a}{\hookrightarrow} U.A.B \\
U.X \overset{b}{\hookrightarrow} U.B.X & U.A \overset{b}{\hookrightarrow} U.B.A & U.B \overset{b}{\hookrightarrow} U.B.B \\
U.X \overset{c}{\hookrightarrow} V.X & U.A \overset{c}{\hookrightarrow} V.A & U.B \overset{c}{\hookrightarrow} V.B \\
U.X \overset{d}{\hookrightarrow} W.X & U.A \overset{d}{\hookrightarrow} W.A & U.B \overset{d}{\hookrightarrow} W.B \\
V.X \overset{e}{\hookrightarrow} V & V.A \overset{a}{\hookrightarrow} V & V.B \overset{b}{\hookrightarrow} V \\
W.X \overset{f}{\hookrightarrow} W & W.A \overset{a}{\hookrightarrow} W & W.B \overset{b}{\hookrightarrow} W \\
V.Y \overset{x}{\hookrightarrow} U.X.Y & W.Y \overset{x}{\hookrightarrow} U.X.Y & \\
V.Y \overset{z}{\hookrightarrow} Z & W.Y \overset{z}{\hookrightarrow} Z &
\end{array}
$$

It follows from Example 4.8 that classes PDA, sePA, and sePAN are not contained ($\not\subseteq$) in any of classes fcBPA, fcPA, and fcPAN. We write

$$\text{PDA}, \text{sePA}, \text{sePAN} \not\subseteq \text{fcBPA}, \text{fcPA}, \text{fcPAN}.$$

Intuition and Conjectures

Similarly to the state extension of PRS, finite constraint systems bring some kind of global communication as well. But a constraint system is more restricted than a finite state unit. Let us recall monotonicity of the store and the feature stating that any further application of a rule cannot be forbidden by a value of the store. Therefore, we conjecture that

$$(\alpha,\beta)\text{-PRS} \subsetneq (\alpha,\beta)\text{-fcPRS} \subsetneq (\alpha,\beta)\text{-sePRS}$$

holds for each (α,β)-PRS except for FS, PDA, and PN, where it is known that (α,β)-PRS $=$ (α,β)-sePRS.

Example 4.9. (fcPA non-PRS) *Let Δ be a fcPA system with the initial process term $X\|Y$ and the following constraint system and rewrite rules.*

$$
\begin{array}{lll}
ff & ttX \overset{a}{\hookrightarrow} ttX.A & oA \overset{a'}{\hookrightarrow} tt\varepsilon \\
\mid & ttX \overset{b}{\hookrightarrow} ttX.B & oB \overset{b'}{\hookrightarrow} tt\varepsilon \\
o & ttY \overset{c}{\hookrightarrow} ttY\|C & oC \overset{c'}{\hookrightarrow} tt\varepsilon \\
\mid & ttX \overset{x}{\hookrightarrow} p\varepsilon & \\
p & pY \overset{y}{\hookrightarrow} o\varepsilon & \\
\mid & & \\
tt & &
\end{array}
$$

It was conjectured that there is no PRS bisimilar to this fcPA system Δ. In [Str02] there is the following intuitive argumentation that supports the conjecture.

The behaviour of Δ defined in the example above is as follows. At the beginning, the process X can perform some actions a, b and remember the order of the

actions, while the process Y can perform just the action c and count the number of performed actions c. The process X can also perform the action x, make a remark p on the store about this action and terminate. Thereafter, the process Y can perform the action y, make a remark o on the store and terminate. When both processes X and Y are terminated (i.e. there is $p \wedge o = o$ on the store), actions a', b', c' can be performed. The order (and the count) of actions a', b' corresponds in reversed order to actions a, b produced before termination of the process X. The count of actions c' is the same as the count of actions c performed before termination of the process Y.

We can approve that this fcPA system is not bisimilar to any PAD process. For the proof we consider the fcPA process without rules labelled by the action b (if we assume that there is a PAD process bisimilar to the original fcPA, then there is also a PAD system without b action bisimilar to the fcPA without b action). Then the behaviour of our system is very similar to the behaviour of fcBPP from Example 4.5, which is not bisimilar to any PAD process. The proof is very similar too.

We can also approve that the considered fcPA process is not bisimilar to any Petri net. The argumentation is based on the fact, that if we remove the rules labelled by c from the fcPA system, then we get a system describing the language $L = \{w.x.y.w^R \mid w \in \{a, b\}^*\}$. The proof that there is no Petri net generating the language L, can be found in [Pet81].

Now we try to explain (on very intuitive level) why we think that there is no PRS process bisimilar to the considered fcPA. Let us assume that Δ is such a bisimilar PRS system. We know this PRS cannot be described by any PAD process. Thus, there must be reachable state with some parallel composition. As the use of the parallel composition must be "non-removable", the information about performed actions a, b, c should be stored in some components of this parallel composition. There should be one parallel component (let us call it p) which saves the information about the order of actions a, b (and thus p is a sequential composition, at least at the top-level), and another parallel component (let us call it q) which remembers the number of performed actions c (the information about the count of actions c cannot be mixed with the information about the order of actions a, b, because after the action y we need a "random access" to the count of actions c). As the sequence of actions a, b can be arbitrary long, the size of corresponding parallel component p is "unbounded" (i.e. for every $n \geq 0$, there is a reachable state where $size(p) > n$). Let m be the maximum size of left-hand sides of rewrite rules in Δ. Further, consider the state of the form $(p \| q \| s).r$, where $size(p) > m$ and process terms s, r can be ε. Then there is no rule, which can change p together with some other part of the term. In other words, there is no way how can q or s provide an information to p. We need such kind of communication for the transition labelled by y, which allows to perform actions a', b', c'. One possible way how to enable these actions at the same time, is to add some term l in front of the parallel composition and enable the action by removing l. But any application of a rewrite rule on the process term of the form $l.(p \| q \| s).r$ cannot modify the process term p if p

is large enough. Thus we cannot add information about next possibly performed actions a, b to p (as well as l cannot be generated by p after the action x if p is large enough). In other words, the problem is that a very large parallel component (which is an sequential composition at the top-level) cannot get any information from other parallel components.

Example 4.10. (sePA non-fcPRS) *Let Δ be a sePA system with the initial state $pX\|Y$ and the the following rewrite rules.*

$$pX \overset{a}{\hookrightarrow} pA.X \qquad pA \overset{a}{\hookrightarrow} pA.A \qquad pB \overset{a}{\hookrightarrow} pA.B \qquad pA \overset{a'}{\hookrightarrow} p\varepsilon$$
$$pX \overset{b}{\hookrightarrow} pB.X \qquad pA \overset{b}{\hookrightarrow} pB.A \qquad pB \overset{b}{\hookrightarrow} pB.B \qquad pB \overset{b'}{\hookrightarrow} p\varepsilon$$
$$pY \overset{c}{\hookrightarrow} pY\|C \qquad pC \overset{c'}{\hookrightarrow} p\varepsilon$$
$$pY \overset{d}{\hookrightarrow} oY$$

The system Δ is the same as the wPA non-fcPRS system of Example 5.29. Hence, also in this case we conjecture that there is no fcPRS system bisimilar to the considered sePA Δ (see Example 5.29 for more information).

Chapter 5

Weakly Extended PRS (wPRS)

The state extension of PRS, described in Chapter 3, is very naturally motivated and brings useful modelling abilities. On the other hand it is too strong at least from point view of automatic verification. State extended PA systems are able to emulate a universal Turing machine. Therefore, from model checking point of view, these systems (and those of the even more expressible classes, namely sePAN, sePAD, and sePRS) are not interesting enough because even the reachability problem is undecidable for them. In the case of BPA the state extended version coincides with previously defined class of (S, S)-PRS systems. Hence, the only "new and interesting" class is the class of seBPP systems.

Being motivated by this observation, it is worth to look for a weaker extension of PRS. Strejček has defined PRS with finite constraint systems and shown that his definition brings new classes lying strictly between each original PRS class and its non-equivalent state extended counterpart (see Chapter 4). Unfortunately, the finite constraint specification is quite complicated and so it is a bit unhandy and not so common for modelling systems of processes. In other words, it remains "too close" to its initial motivation (coming from CCP) to be useful for modelling different kind of FSU.

In this chapter, we introduce yet another extension lying between the finite constraint extension and the state extension. A definition of our extension is very close to the state extension (that succeed in modelling abilities) and at the same time it keeps advantages of the finite constraint extension (esp. decidability of the reachability problem).

5.1 Definition of Weakly Extended PRS

This section defines weakly extended PRS (wPRS) classes that were introduced in [KŘS04b]. We extend PRS by a finite state control unit whose transition function satisfies some restrictions inspired by weak finite automata

(introduced in [MSS92], but used here as a nondeterministic (NFA) rather than alternating one). Roughly speaking, weakly extended PRS is a state extended PRS where the FSU is weak (its transition function exhibits some monotonicity).

Definition 5.1. *Let* $Act = \{a, b, \cdots\}$ *be a countably infinite set of* atomic actions *and* $\alpha, \beta \in \{1, S, P, G\}$ *such that* $\alpha \subseteq \beta$. *An* (α, β)-wPRS (weakly extended PRS) *is a tuple* $\Delta = (M, \geq, R, m_0, t_0)$, *where*

- $(M, \geq)$ *is partially ordered finite set representing states of* weak finite-state unit; *the elements of* M *are called* w-states,
- $R \subseteq ((M \times (\alpha \setminus \{\varepsilon\})) \times Act \times (M \times \beta))$ *is a finite set of* rewrite rules *of the form* $((m, t_1), a, (n, t_2))$ *satisfying the condition* $m \geq n$.
- $m_0 \in M$ *is an* initial w-state *of the control unit, and*
- $t_0 \in \beta$ *is an* initial term.

We write $((m, t_1) \overset{a}{\hookrightarrow} (n, t_2)) \in R$ *instead of* $((m, t_1), a, (n, t_2)) \in R$.

To shorten our notation we write mt instead of (m, t). As in the PRS case, instead of $(mt_1 \overset{a}{\hookrightarrow} nt_2) \in R$ where $\Delta = (\mathcal{C}, R, tt, t_0)$, we usually write $(mt_1 \overset{a}{\hookrightarrow} nt_2) \in \Delta$. The meaning of $Const(\Delta)$ (process constants used in rewrite rules or in t_0), $Act(\Delta)$ (actions occurring in rewrite rules), and $M(\Delta)$ (values of the store occurring in Δ) for a given wPRS Δ is the same as in the sePRS case.

Definition 5.2. *The semantics of an* (α, β)*-wPRS* $\Delta = (M, \geq, R, m_0, t_0)$ *is given by the LTS* $(S, Act(\Delta), \longrightarrow_\Delta, m_0 t_0)$, *where*

- $S = M(\Delta) \times \{t \in \beta \mid Const(t) \subseteq Const(\Delta)\}$,
- $m_0 t_0$ *is the initial state composed of the initial w-state* m_0 *and the initial term* t_0, *and*
- *the transition relation* $\longrightarrow_\Delta$ *is defined as the least relation satisfying the following inference rules for all* $mt_1, nt_2, m(t_1 \| t), m(t_2 \| t), n(t_1.t), n(t_2.t) \in S$ *and* $a \in Act(\Delta)$.

$$\frac{(mt_1 \overset{a}{\hookrightarrow} nt_2) \in \Delta}{mt_1 \overset{a}{\longrightarrow}_\Delta nt_2} \qquad \frac{mt_1 \overset{a}{\longrightarrow}_\Delta nt_2}{m(t_1 \| t) \overset{a}{\longrightarrow}_\Delta n(t_2 \| t)} \qquad \frac{mt_1 \overset{a}{\longrightarrow}_\Delta nt_2}{m(t_1.t) \overset{a}{\longrightarrow}_\Delta n(t_2.t)}$$

Note that parallel composition is commutative and, thus, the inference rule for parallel composition also holds with t_1 *and* t *exchanged.*

The presented notion of weakness corresponds to 1-weakness condition in automata theory. (As we do not consider final states here, the general weak unit would coincide with standard state-extension; more precisely, all the states of M could be included in one partition block.) In any transition sequence, following the rules of R, the w-state components form a non-increasing sequence with respect to $\geq$. Hence, similarly to the finite constraint extension, the w-state is *monotonic*. However, in contrast to fcPRS, the weak unit can forbid the application of any rewrite rule. As this is the monotonicity is the only restriction and the FSU is finite, we can also reformulate the difference between wPRS and sePRS as follows.

wPRS is a sePRS where its FSU can change its state only finitely many times during every execution.

As instead of $(1, S)$-sePRS, $(1, P)$-sePRS, etc. we use more traditional abbreviations seBPA, seBPP, etc., we also take up this notation for all weakly extended PRS classes. Thus for $(1, 1)$-wPRS, $(1, S)$-wPRS, $(1, P)$-wPRS, (S, S)-wPRS, $(1, G)$-wPRS, (P, P)-wPRS, (S, G)-wPRS, (P, G)-wPRS, and (G, G)-wPRS we use human-readable abbreviations wFS, wBPA, wBPP, wPDA, wPA, wPN, wPAD, wPAN, and wPRS respectively.

5.2 Hierarchy of Weakly Extended PRS

In this section we discus relations between classes of PRS, PRS with finite constraint systems, weakly extended PRS, and state extended PRS. A hierarchy of all these systems is depicted in Figure 5.1 and called the *extended PRS-hierarchy*.

It is an immediate observation that

$$(\alpha, \beta)\text{-PRS} \subseteq (\alpha, \beta)\text{-fcPRS} \subseteq (\alpha, \beta)\text{-wPRS} \subseteq (\alpha, \beta)\text{-sePRS}$$

hold for every (α, β)-PRS class (even up to isomorphism).

The first and the last inclusions are obvious. To justify the second inclusion we take an arbitrary (α, β)-fcPRS Δ with an initial term t_0 and a constraint system $\mathcal{C} = (C, \vdash, \wedge, tt, ff)$. The corresponding (α, β)-wPRS is

$$\Delta' = (C \setminus \{ff\}, \geq, R', tt, t_0),$$

where

$$R' = \{ot_1 \overset{a}{\hookrightarrow} (o \wedge n)t_2 \mid (mt_1 \overset{a}{\hookrightarrow} nt_2) \in \Delta \text{ and } o \vdash m \text{ and } o \wedge n \neq ff\}$$

and $\geq = (\vdash)^{-1}$.

We also note that the classes FS, PDA, and PN have the same expressive power as the corresponding $\{\text{fc}, \text{w}, \text{se}\}$ extended classes. Finally, we mention that for every two classes X and Y of PRS

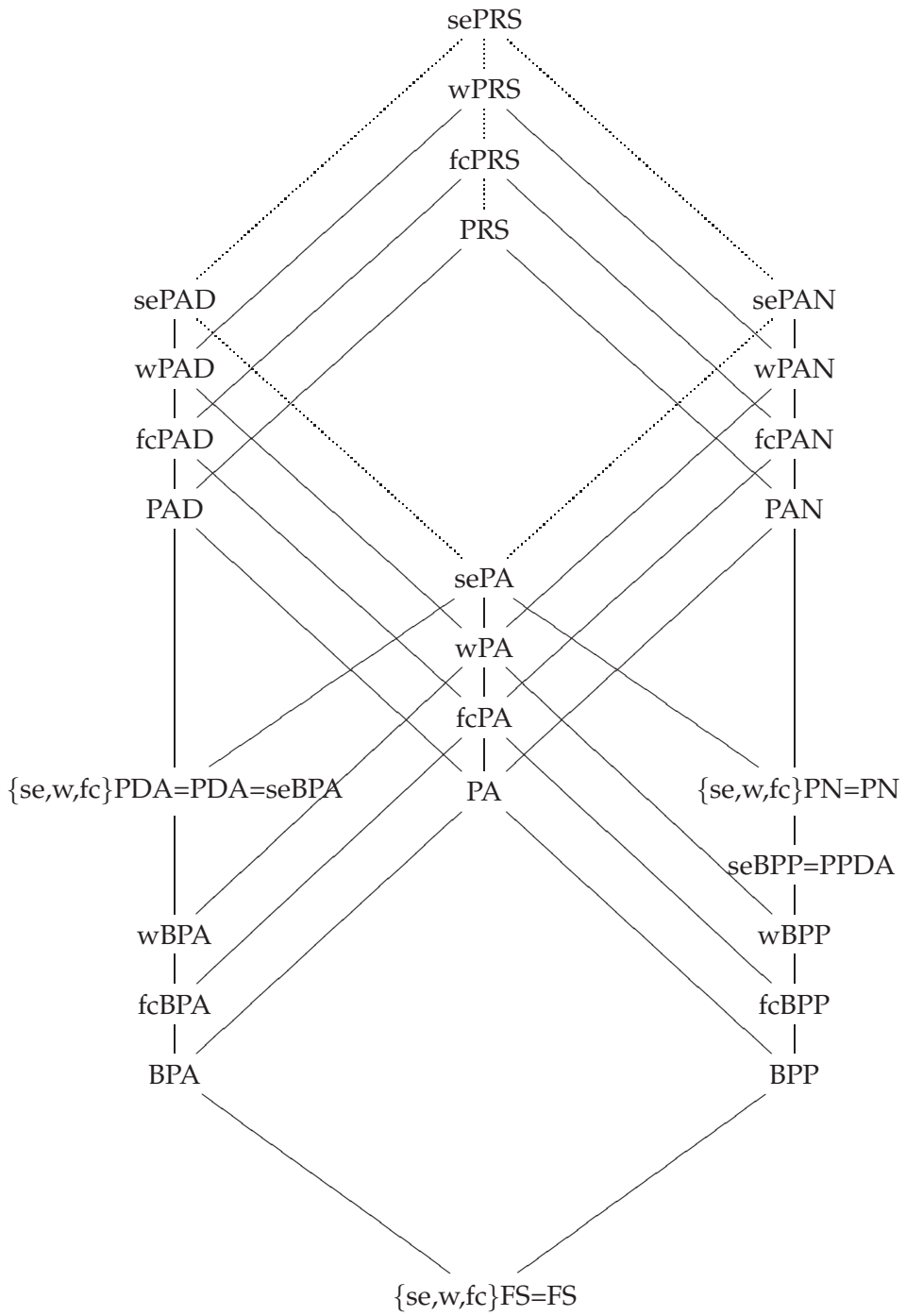


Figure 5.1: The extended PRS-hierarchy.

$$X \subseteq Y \text{ implies } wX \subseteq wY.$$

The strictness ('$\subsetneq$') of all relations of the extended PRS-hierarchy and incomparability of all non-related classes of the extended PRS-hierarchy follow from exemplifying the following systems:

- a BPP system which is not bisimilar to any PDA system,
- a fcBPP system which is not bisimilar to any PAD system,
- **a wBPP system which is not bisimilar to any fcPAD system**,
- **a PPDA system which is not bisimilar to any wPAD system**,
- a PN system which is not bisimilar to any PPDA system,
- a PAN system which is not bisimilar to any sePAD system,

- a BPA system which is not bisimilar to any PN system,
- a fcBPA system which is not bisimilar to any PAN system,
- **a wBPA system which is not bisimilar to any fcPAN system**,
- **a PDA system which is not bisimilar to any wPAN system**,
- a PAD system which is not bisimilar to any sePAN system,

- a fcPA system which is not bisimilar to any PRS system,
- **a wPA system which is not bisimilar to any fcPRS system**, and
- **a sePA system which is not bisimilar to any wPRS system**.

For the non-bold items we simply refer to Section 2.3, Section 3.2, and Section 4.2, where the systems are exemplified. The bold systems are examined in the following subsections. Unfortunately, concerning the wPA non-fcPRS and sePA non-wPRS systems, we can speak about our conjectures only. Therefore these systems are included in the last subsection of conjectures.

5.2.1 wBPP Non-bisimilar to any fcPAD

Constructing a proof of this section, we also formulate a property that forms a sufficient condition for a PAD to be bisimilar to some PDA (see the following definition and Lemma 5.6).

Definition 5.3. *An LTS $(S, Act, \longrightarrow, \alpha_0)$ is* deadlockable *if for each reachable nonterminal state α there is a transition from α to a terminal state.*

$$\forall\, \alpha \in S : \alpha_0 \longrightarrow^* \alpha \implies \exists\, a \in Act, \beta \in S : \alpha \xrightarrow{a} \beta \not\longrightarrow$$

A rewrite system Δ is deadlockable *if its underlying LTS is deadlockable.*

Let us note that deadlockable does not mean "there is a (reachable) deadlocked state in the system". The definition says that every reachable state of the system has an immediate deadlocked successor.

Definition 5.4. *A sequential subterm t (i.e. $t \in S$) of term $g \in G$ is a* ready parallel component *if and only if t is a maximal subtree in the syntax tree of term g such that t is not in the right-hand side subtree of any sequential node (i.e. node corresponding to sequential operator). A ready parallel component t is* live *in a PAD system Δ if t is not deadlocked (i.e. there is a rule applicable to t).*

The ready parallel components can be intuitively defined as the maximal sequential parts of a PAD process g such that g can perform an action a if and only if some of its ready parallel components can perform the same action a.

Lemma 5.5. *Every reachable state of an arbitrary deadlockable PAD system has at most one live ready parallel component.*

Proof. Observe that the application of a PAD rewrite rule can only modify one ready parallel component. Hence there is no way how to deadlock more than one live ready parallel component by one application of a PAD rewrite rule. □

Lemma 5.6. *Every deadlockable PAD system is bisimilar to a PDA system.*

Proof. The proof is build as a transformation of PAD rewrite rules onto corresponding PDA rewrite rules. This is sufficient as for every PAD there is a bisimilar PAD system with a single process constant as the initial process term (see Remark 2.9).

There is only one way to revive a deadlocked parallel component, namely to rewrite adjacent components onto ε. For example if $B.C$ is a deadlocked ready parallel component of $(A.C \| B.C).D$ and $(A.C \| B.C).D \xrightarrow{w}{}^* (\varepsilon \| B.C).D = B.C.D$ then the ready parallel component $B.C.D$ can be live.

Let Δ be a PAD system and $X \notin Const(\Delta)$ be a fresh process constant. Let us consider a rewrite rule of Δ with the right hand side containing a maximal subterm of the form $l.(t_1 \| t_2).r$, where $t_1, t_2 \in S$ and l, r can be ε. In an arbitrary transition sequence the components t_1, t_2 generated by the application of the considered rewrite rule become ready at the same time. Thus at least one of them is deadlocked. Let t_2 be deadlocked. We replace the subterm $l.(t_1 \| t_2).r$ of the rule by $l.X.t_1.X.t_2.r$ (or just $t_1.X.t_2.r$ whenever l is ε). The process constant X eliminates any possible (unwanted) interaction of (the tail of) the term l and (the beginning of) the term t_1 (or the tail of t_1 and the beginning of t_2 respectively). Repeating this procedure eliminates all parallel operators from rewrite rules. The resulting PDA system Δ' enriched by rewrite rules of the form $X.s \overset{a}{\hookrightarrow} s'$ for every rule $s \overset{a}{\hookrightarrow} s' \in \Delta'$ is bisimilar to the given Δ. □

Example 5.7. (wBPP non-fcPAD) *Let Δ_1 be the wBPP system with the initial state pX and the rules:*

$$pX \overset{c}{\hookrightarrow} pX \| A \| B \qquad pA \overset{a}{\hookrightarrow} p\varepsilon \qquad pB \overset{b}{\hookrightarrow} p\varepsilon \qquad pX \overset{d}{\hookrightarrow} q\varepsilon$$

Lemma 5.8. *There is no PAD system bisimilar to the wBPP system Δ_1 of Example 5.7.*

Proof. Δ_1 is deadlockable. Due to Lemma 5.6 it suffices to prove there is no PDA system bisimilar to Δ_1. This directly follows from the fact that the language L generated by Δ_1 is not context-free. More precisely, context-free languages are closed under intersection with regular languages and

$$L \cap \{c\}^*\{a\}^*\{b\}^*\{d\} = \{c^k a^l b^m d \mid 0 \leq l \leq k \wedge 0 \leq m \leq k\}$$

is not a context-free language. □

Lemma 5.9. *There is no fcPAD system bisimilar to the wBPP system Δ_1 of Example 5.7.*

Proof. For the sake of a contradiction we assume a fcPAD Δ bisimilar to Δ_1.

The finiteness of the constraint system used in Δ implies that there exists a reachable non-terminal state mt of Δ such that every non-terminal state reachable from mt has also m on the store (the contrary would mean the constraint system is infinite). As mt is non-terminal there exists a word $w \in \{a, b\}^*$ such that $mt \xrightarrow{w}{}^*_\Delta ms$ and ms is bisimilar to the initial state pX of Δ_1. The only transitions starting at states reachable from ms and changing the value of the store can be the transitions leading to terminal states, i.e. the transitions labelled by d. Hence we can directly assume that all rewrite rules of Δ labelled with $x \in \{a, b, c\}$ have the form $(tt\ t_1 \overset{x}{\hookrightarrow} tt\ t_2)$.

Let Δ' be a PAD system with the set of rewrite rules as

$$\{t_1 \overset{x}{\hookrightarrow} t_2 \mid (tt\ t_1 \overset{x}{\hookrightarrow} tt\ t_2) \in \Delta, x \neq d\} \cup$$
$$\cup \quad \{t_1 \overset{d}{\hookrightarrow} Z \mid (tt\ t_1 \overset{d}{\hookrightarrow} n\ t_2) \in \Delta, n \neq ff\},$$

where $Z \notin Const(\Delta)$ is a fresh process constant. If we restrict the systems Δ and Δ' to actions a, b, c then Δ and Δ' are bisimilar. Furthermore in every state q of Δ' reachable under $w \in \{a, b, c\}^*$ there is a transition labelled by d and starting at q. It suffices to show that this transition is leading to a terminal state.

The state ttq (corresponding to the state q) has a ready parallel component able to perform an action c. This action cannot be disabled by any action performed by another ready parallel component. Hence there is just one ready parallel component able to perform both c and d. For the same reason this component is the only one which is able to perform actions a and b if they are enabled in the state ttq. The same holds for the state q of Δ'. Moreover the ready parallel component rewritten by the action d is deadlocked by the process constant Z. Thus the state reached under d is terminal and we get a PAD system Δ' bisimilar to the wBPP Δ_1 of Example 5.7 – a contradiction (see Lemma 5.8). □

5.2.2 PPDA Non-bisimilar to any wPAD

Example 5.10. (PPDA non-wPAD) *Let Δ_2 be a PPDA process with the initial state $xC\|A\|B$ and the following rewrite rules:*

$$xC \overset{g}{\hookrightarrow} xC\|A\|B \qquad yA \overset{a}{\hookrightarrow} y\varepsilon \qquad xA \overset{d}{\hookrightarrow} z\varepsilon \qquad yA \overset{d}{\hookrightarrow} z\varepsilon$$
$$xC \overset{c}{\hookrightarrow} yC \qquad yB \overset{b}{\hookrightarrow} y\varepsilon \qquad xB \overset{d}{\hookrightarrow} z\varepsilon \qquad yB \overset{d}{\hookrightarrow} z\varepsilon$$

$$yC \overset{e}{\hookrightarrow} zC \qquad zA \overset{e}{\hookrightarrow} z\varepsilon \qquad zB \overset{e}{\hookrightarrow} z\varepsilon \qquad zC \overset{r}{\hookrightarrow} xC\|A\|B$$

The rules labelled by g, a, b, c, d correspond to the rules of the PPDA non-PAD given in Example 3.4. Hence the PPDA Δ_2 behaves as the previously mentioned PPDA non-PAD, but when the PPDA non-PAD terminates, the PPDA Δ_2 can "erase" an arbitrary number of A and B symbols from the parallel stack and then "restart" the system under action r.

Lemma 5.11. *There is no wPAD Δ' bisimilar to the PPDA Δ_2 of Example 5.10.*

Proof. To derive a contradiction assume a wPAD Δ bisimilar to the PPDA Δ_2. As the weak state unit of Δ is finite then there exists a reachable state mt of Δ_2 such that every state reachable from mt has also m as its w-state component (the opposite would imply the infiniteness of the weak state unit).[1] There exists a word $w \in \{g, a, b, c, d, e, r\}^*$ such that $mt \overset{w.r}{\longrightarrow}^*_{\Delta} mt'$,

[1] Note that contrary to the proof of Lemma 5.9, there are no terminal states reachable in Δ_2, and so there are no terminal states in Δ as they are bisimilar.

where mt' is bisimilar to the state $xC\|A\|B$ of the PPDA process Δ_1. If the rules labelled by the actions e and r are removed from Δ and mt' is taken as the initial state, we obtain the system whose all reachable states have m as w-state component and which is bisimilar to the PPDA of Example 3.4.

Now let Δ' be a PAD system with the initial state t' and with the set of rewrite rules consisting of the rules $l \overset{v}{\hookrightarrow} r$, where $(ml \overset{v}{\hookrightarrow} mr) \in \Delta$ and $v \in \{g, a, b, c, d\}$. It is obvious that this PAN system Δ' is bisimilar to the PPDA system defined in Example 3.4 – a contradiction. □

5.2.3 wBPA Non-bisimilar to any fcPAN

Definition 5.12. *A parallel subterm t ($t \in P \setminus \{\varepsilon\}$) of term $g \in G$ is a* ready sequential component *if and only if t is a maximal subtree in the syntax tree of term g such that t does not occur in any right sequential component of g.*

A ready sequential component t of term g is called non-trivial *if and only if $\exists t' \neq \varepsilon$ such that $t.t'$ is a subterm of g (i.e. t is located in subterm $t.t'$ of g, where $t' \neq \varepsilon$).*

Intuitively, the ready sequential components are defined as maximal parallel subterms of a fcPAN process g such that g can perform an action a if and only if some of its ready sequential components can perform the action a.

Definition 5.13. *Let $\mathcal{L} = (S, Act, \longrightarrow, \alpha_0)$ be a labelled transition system, α be a state of S, Σ be a subset of Act, and $u \in \Sigma^*$. We define an LTS $\mathcal{L}|_\Sigma$ to be the following restriction of $\mathcal{L}$ to Σ.*

$$(S, \Sigma, \longrightarrow \cap\, (S \times \Sigma \times S), \alpha_0)$$

We also define a notation $only^\Sigma(\alpha, u)$ stating that u is a prefix of each maximal transition sequence in $\mathcal{L}|_\Sigma$ starting in α.

Definition 5.14. *A sequential composition of t (i.e. the sequential operator '.' within t) is said to be* accessed *during a rewriting sequence from mt under w if and only if the left subterm of this sequential composition is rewritten onto ε during the rewriting sequence.*

A sequential composition of t is called accessible *from mt under w if and only if there is a rewriting sequence from mt under w such that the sequential composition is accessed during this rewriting sequence; otherwise it is called* inaccessible.

Definition 5.15. *For every fcPAN Δ and every number $n \in \mathbb{N}_0$ we define $K_n(\Delta)$ to be a set of all n-tuples $(k_1, k_2, \ldots, k_n) \in \mathbb{N}^n$ such that for every $a, b \in Act(\Delta)$, $a \neq b$, every state mt of Δ satisfying $only^{\{a,b\}}(mt, a^{k_1}ba^{k_2}b \ldots a^{k_n}b)$, and every rewriting sequence from mt under $a^{k_1}ba^{k_2}b \ldots a^{k_n}b$, t includes at least n sequential compositions accessed during the rewriting sequence.*

Intuitively, $K_n(\Delta)$ is a set of all n-tuples $(k_1, k_2, \ldots, k_n)$ such that each state mt of Δ holding the information that every run under $\{a, b\}^*$ starts with $a^{k_1}ba^{k_2}b \ldots a^{k_n}b$ needs at least n sequential compositions.

Lemma 5.16. *For every fcPAN Δ and every number $n \in \mathbb{N}_0$, $K_n(\Delta) \neq \emptyset$.*

Proof. We prove the lemma by induction on n. The base $n = 0$ is easy to prove. As every state mt and every actions a, b satisfy $only^{\{a,b\}}(mt, \varepsilon)$ and every term includes zero sequential compositions accessed during an empty rewriting sequence, it holds that $() \in K_0(\Delta)$ for every fcPAN Δ.

Induction hypothesis: Let $n \in \mathbb{N}_0$ be such that $K_n(\Delta) \neq \emptyset$ for every fcPAN Δ. We assume the contrary for $n + 1$ and derive a contradiction.

Let Δ be an fcPAN such that for every $(k_1, \ldots, k_n, k_{n+1}) \in \mathbb{N}^{n+1}$ there exist distinct actions $a, b \in Act(\Delta)$, a state mt of Δ satisfying the condition $only^{\{a,b\}}(mt, a^{k_1}b \ldots a^{k_n}ba^{k_{n+1}}b)$, and a rewriting sequence from mt under $a^{k_1}ba^{k_2}b \ldots a^{k_n}ba^{k_{n+1}}b$ such that t includes at *most* n sequential compositions accessed during this rewriting sequence. Due to induction hypothesis, we can choose $k_1, \ldots, k_n$ such that $(k_1, \ldots, k_n) \in K_n(\Delta)$. Hence, for every $l \in \mathbb{N}$, there exist distinct actions $a_l, b_l \in Act(\Delta)$, a state $m_l t_l$ of the system Δ satisfying $only^{\{a_l,b_l\}}(m_l t_l, a_l^{k_1} b_l \ldots a_l^{k_n} b_l a_l^l b_l)$, and a rewriting

$$m_l t_l \xrightarrow{a_l^{k_1} b_l \ldots a_l^{k_n} b_l}{}^*_\Delta \; m'_l t'_l \xrightarrow{a_l^l b_l}{}^*_\Delta \; m''_l t''_l$$

such that t_l includes at *most* n sequential compositions accessed during the rewriting. Moreover, due to $(k_1, \ldots, k_n) \in K_n(\Delta)$, t_l includes *exactly* n sequential compositions accessed during the rewriting

$$m_l t_l \xrightarrow{a_l^{k_1} b_l \ldots a_l^{k_n} b_l}{}^*_\Delta \; m'_l t'_l$$

and no other sequential composition of t_l is accessed during the rewriting

$$m'_l t'_l \xrightarrow{a_l^l b_l}{}^*_\Delta \; m''_l t''_l$$

As for every $l \in \mathbb{N}$ there is a state $m'_l t'_l$, we consider an infinite sequence $\{m'_l t'_l\}_l$ of these states. Let α be an infinite subsequence of the sequence $\{m'_l t'_l\}_l$ such that all states of α have the same value of the store (say m') and the same corresponding pair of letters a_l, b_l (say a, b); the existence of such a subsequence follows from the finiteness of the constraint system and $Act(\Delta)$.

We have not finished yet. There still can be a sequential composition of t'_l that is accessed during the rewriting under $a_l^l b_l$ such a composition could be created during the rewriting between t_l and t'_l. Because of this, we

differentiate between subterms of t'_l depending on whether they have been changed during the preceding rewriting

$$m_l t_l \overset{a^{k_1}b\ldots a^{k_n}b}{\longrightarrow}{}^{*}_{\Delta} \; m' t'_l \,.$$

The subterms of t'_l that *have not been rewritten* during the rewriting are called *blue subterms*. The new subterms of t'_l that *have been created* (or changed) during the rewriting are called *green subterms*. The sequential compositions of t'_l that *have not been accessed* during the rewriting are called *blue sequential compositions*. The sequential compositions of t'_l that *have been created* during the rewriting are called *green sequential compositions*. Note that the green sequential compositions are exactly those inside the green subterms; the blue ones are the other, i.e. inside blue subterms and connecting blue and green subterms. The green subterms of t'_l are created during at most $k_1 + k_2 + \cdots + k_n + n$ actions, therefore their number and syntactical lengths are bounded independently on l.

In the definition of ready sequential components we described subterms that are significant for the next rewriting step – we disregarded the right-hand sequential components. Now, we want to focus on the rewriting sequence under $a_l^l b_l$. As the blue sequential components are inaccessible in the rewriting, we disregard their right-hand sequential components but the green subterms has to be included. We call this subterms *interesting ready sequential components (irs-components)*.[2] Using associativity, commutativity, and empty terms, we consider every irs-component as a parallel composition of one blue subterm and one green subterm. For example, a term

$$((\, B_1 \parallel (G_1 . G_2) \parallel B_2 \parallel G_3 \,) . B_3) \parallel G_4$$

where B_1, B_2, B_3, B_4 are blue and G_1, G_2, G_3, G_4 are green, there is the the following irs-component that are composed of one blue and one green subterm like this:

$$(B_1 \| B_2) \parallel ((G_1 . G_2) \| G_3) \;\text{ and }\; (\varepsilon) \parallel (G_4)$$

also note that we can add an arbitrary number of $(\varepsilon)\|(\varepsilon)$ irs-components.

Let $m' t'_i$ and $m' t'_j$ $(i < j)$ be two states of α such that there is a bijection on their irs-components satisfying: the corresponding irs-components have identical green subterms and for their blue subterms $s_i, s_j \in P$ there is a term $s \in P$ such that $s_i \| s = s_j$. The existence of such states follows from the bound of the number and the syntactical lengths of green subterms and due to Dickson's lemma.

Besides considered the irs-components there are no other subterms of t'_i rewritten during the rewriting sequence under $a^i b$. Hence the

[2]Note that irs-components are not parallel subterms they can include green sequential compositions and so they are not ready sequential components according to Definition 5.12.

sequence of actions a^ib performed by $m't'_i$ can be performed also by $m't'_j$. The contradiction follows from $only^{\{a,b\}}(mt_i, a^{k_1}b \dots a^{k_n}ba^ib)$, $only^{\{a,b\}}(mt_j, a^{k_1}b \dots a^{k_n}ba^jb)$, and $i < j$. □

Example 5.17. (wBPA non-fcPAN) *Let us consider the following wBPA system Δ_3 with initial state pX.*

$$pX \overset{a}{\hookrightarrow} pAX \qquad pA \overset{a}{\hookrightarrow} pAA \qquad pB \overset{a}{\hookrightarrow} pAB \qquad pA \overset{c}{\hookrightarrow} p\varepsilon$$
$$pX \overset{b}{\hookrightarrow} pBX \qquad pA \overset{b}{\hookrightarrow} pBA \qquad pB \overset{b}{\hookrightarrow} pBB \qquad pB \overset{d}{\hookrightarrow} p\varepsilon$$

$$pA \overset{e}{\hookrightarrow} q\varepsilon \qquad\qquad qA \overset{e}{\hookrightarrow} q\varepsilon$$
$$pB \overset{f}{\hookrightarrow} q\varepsilon \qquad\qquad qB \overset{f}{\hookrightarrow} q\varepsilon$$

In the following, actions a, b, c, d are called *I-actions* and actions e, f are called *II-actions*). Rules labelled by I-actions or II-actions are called *I-rules* or *II-rules*, respectively. States reachable from the initial state through I-actions are called *I-states*. Note that all I-states are nonterminal and using c and d actions can be rewritten onto a state bisimilar with the initial state.

In the rest of this subsection we prove that there is no fcPAN system bisimilar to the wBPA system Δ_3 given above.

Let bis-fcPAN denote an assumed fcPAN system Δ bisimilar to wBPA of Example 5.17 such that I-rules of Δ are of the form $tt\ t_1 \overset{x}{\hookrightarrow} tt\ t_2$. Please note these rules cannot be forbidden by any value of the store.

Lemma 5.18. *If there is a fcPAN Δ bisimilar to the wBPA of Example 5.17, then bis-fcPAN Δ' exists.*

Proof. As the constraint system of Δ is finite it follows there exists an I-state mt of Δ such that each I-state reachable from mt has also m on the store (the contrary implies the infiniteness of the constraint system). As mt is an I-state, there exists a word $w \in \{c, d\}^*$ such that $mt \overset{w}{\longrightarrow}^*_\Delta ms$ and ms is bisimilar to the initial state pX of the wBPA. The system Δ' is derived from Δ as follows:

- s is the initial term,
- the set of rules is

$$\{\ (n \wedge m)t_1 \overset{x}{\hookrightarrow} (o \wedge m)t_2 \ \mid\ nt_1 \overset{x}{\hookrightarrow} ot_2 \text{ is a rule of } \Delta\ \},$$

- the constraint system is restricted to the part above m, and
- m is renamed tt.

□

Lemma 5.19. *Each I-state of a bis-fcPAN Δ has exactly one ready sequential component that is not deadlocked.*

Proof. As no I-state is deadlocked, each I-state contains at least one non-deadlocked ready sequential component. We prove that there is exactly one such a component. We assume contrary and derive a contradiction. Let t and s be two distinct non-deadlocked ready sequential components of an I-state. The are two cases.

At first we assume that the I-state is non-bisimilar to the initial state. Hence, the I-state can perform e action. Let t be the ready sequential component that can perform e action. There are the following facts.

- s cannot perform any I-action as the e action performed by t is not able to forbid the I-rules (i.e. to disable these actions),
- s cannot perform e. Otherwise, neither s can perform enabled I-actions, nor other ready sequential component can perform them (according to the previous item with exchanged t for s),
- s cannot perform f as this action is disabled in the considered state.

Therefore the component s is deadlocked and we have a contradiction.

Now we assume that the I-state is bisimilar to the initial state. Without loss of generality, let us assume that t can perform a action and s can perform a b action (t and s can possibly perform some other actions as well). Each possible next state has exactly one non-deadlocked ready sequential component. Thus, a action (performed by t) deadlocks or rewrites onto ε term t (the action cannot deadlock or remove s) and the same effect has action b on s. Further, these actions add the ability to perform action e or f to the next state. Hence, a action performed by t changes the ready sequential component s at the same time. This is possible only if t and s are contained in the subterm of the form $(t.r)\|s$, where $r \in P \setminus \{\varepsilon\}$ and a action rewrites t onto ε:

$$(t.r)\|s \xrightarrow{a}_{\Delta} r\|s$$

For the same reason there has to be a term $r' \in P \setminus \{\varepsilon\}$ such that t and s are contained in the subterm of the form $t\|(s.r')$. This is a contradiction. □

Definition 5.20. *A ready sequential component is called* dead *if and only if it is non-trivial and deadlocked whenever the value of the store is tt. A left subterm of a sequential composition is called* dead *if and only if it contains (or is) dead ready sequential component. A sequential composition is called* dead *if and only of its left subterm is dead.*

We distinguish between a *type* and an *instance* of a ready sequential component. The type is given by (syntax of) the corresponding parallel

subterm while the instance is given by the subterm together with its position within the term. If it is clear from the context, we do not specify the meaning explicitly.

In what follows any dead ready sequential component occurring in some I-state of Δ is referred to as *dead ready sequential component of bis-fcPAN* Δ.

Lemma 5.21. *Given bis-fcPAN Δ, the set of types of dead ready sequential components occurring in I-states of Δ is finite.*

Proof. As a dead ready sequential component is non-trivial, it remains ready and unchanged during an arbitrary sequence of I-actions. In a bis-fcPAN, there are only two possibilities of creating a dead ready sequential component. Either it is included in the initial term, or it is on the right-hand side of an applied rule (it cannot be created by deadlocking some non-deadlocked ready sequential component as each I-state has exactly one non-deadlocked ready sequential component). Hence the lemma follows from the fact that the length of an initial term and the set of rules are both finite. □

Definition 5.22. *Let Δ be a bis-fcPAN and r be a dead ready sequential component of Δ. Then r is called*

- restricted *for an I-state tt t of Δ if and only if there is no I-state with an added instance of dead ready sequential component r reachable from tt t,*
- multiplicative *for an I-state tt t of Δ if and only if for each I-state tt t' reachable from tt t, there is an I-state tt t'' reachable from tt t' such that there are more instances of r in mt'' than in mt' (i.e. arbitrary many instances of r can be added).*

We call an I-state tt t_{dr} of Δ dead-restricted *if and only if every dead ready sequential component of Δ is either restricted, or multiplicative for tt t_{dr}.*

Lemma 5.23. *Let Δ be a bis-fcPAN. There is a dead-restricted state tt t_{dr} of Δ reachable from the initial state.*

Proof. If every dead ready sequential component of Δ is either restricted, or multiplicative in an I-state *tt* t then the *tt* t_{dr} is found. Otherwise, there is a dead ready sequential component r such that it is neither restricted, nor multiplicative. As r is not multiplicative for *tt* t, there is an I-state *tt* t' reachable from *tt* t such that every I-state reachable from *tt* t' has the same number of instances of r as *tt* t'. Hence, r is restricted for the I-state *tt* t'. Compared to *tt* t, at least one more type of dead ready sequential component is restricted. Due to Lemma 5.21, we can find *tt* t_{dr} by applying this procedure finitely many times. □

Lemma 5.24. *Given $i \in \mathbb{N}$, bis-fcPAN Δ, and dead-restricted state tt t_{dr}, there is a dead-restricted state $t_{dr}^{(i)}$ of Δ reachable from the state tt t_{dr} such that $t_{dr}^{(i)}$ includes at least i instances of each multiplicative dead ready sequential component.*

Proof. This lemma is a straightforward consequence of the definition of multiplicative dead ready sequential components. □

In what follows, by *a sequential composition is behind a subterm s in term r* we mean that, in the term r, s is included in the left subterm of the sequential composition. By *a subterm t is behind a subterm s in term r* we mean that there is a sequential composition in the term r such that t is included in the left subterm and s is included in the right subterm of this sequential composition.

Lemma 5.25. *If there are more than i instances of dead ready sequential component r in a state tt t of a bis-fcPAN, then all the sequential compositions behind these instances are inaccessible from tt t under any sequence of the form $e^{k_1} f e^{k_2} f \dots e^{k_n} f$ such that $n \in \mathbb{N}_0$ and $k_j \leq i$ for every $1 \leq j \leq n$.*

Proof. We assume the contrary and derive a contradiction. Let *tt t* be a state of a bis-fcPAN with more than i instances of some dead ready sequential component such that a sequential composition behind some of these instances is accessible under $e^{k_1} f e^{k_2} f \dots e^{k_n} f$. The definition of bis-fcPAN implies that $only^{\{e,f\}}(tt\ t, e^{k_1} f e^{k_2} f \dots e^{k_n} f)$ holds. From the definition of fcPAN, it follows that a rule which has been already used cannot be forbidden in future. Thus, whenever an action is performed by one of the instances, the same rule can be immediately applied on the other instances. Hence, a coherent sequence of more that i actions with the same label can be performed. This is a contradiction with $only^{\{e,f\}}(tt\ t, e^{k_1} f e^{k_2} f \dots e^{k_n} f)$. □

Lemma 5.26. *There is no fcPAN system bisimilar to the wBPA of Example 5.17.*

Proof. Lemma 5.18 implies that it is sufficient to show that there is no bis-fcPAN. For the sake of contradiction, let us assume that there is a bis-fcPAN system Δ with an initial term t_0 and a set of constraints C. Let us consider the following transition sequence:

$$tt\ t_0 \longrightarrow_\Delta^* tt\ t_{dr} \longrightarrow_\Delta^* tt\ t_{dr}^{(k)} \xrightarrow{w_n}{}_\Delta^*\ tt\ r$$

The state $tt\ t_{dr}$ is a dead-restricted state (its reachability follows from Lemma 5.23). Let l be the number of sequential compositions in t_{dr} and $n = l + |C| + 1$. Lemma 5.16 implies that $K_n(\Delta)$ is non-empty. Let $(k_1, k_2, \dots, k_n) \in K_n(\Delta)$ and k be the maximum of $k_2, k_2, \dots, k_n$. Then $tt\ t_{dr}^{(k)}$ denotes the dead-restricted state with more than k instances of each multiplicative dead ready sequential component (it is reachable due to Lemma 5.24). Further, $w_n = ba^{k_n} \dots ba^{k_2} ba^{k_1}$.

The term r is a non-deadlocked term, hence it can be written in the form

$$(\cdots((((((\cdots((((t.t_1)\|s_1).t_2)\|s_2)\cdots .t_n)\|s_n).\gamma)\|\delta).\gamma')\|\delta')\cdots .\gamma^{(m)})\|\delta^{(m)},$$

where $t \in P \setminus \{\varepsilon\}$ is the only one non-deadlocked ready sequential component (Lemma 5.19), and $t_1, \ldots, t_n, s_1, \ldots, s_n, \gamma, \ldots, \gamma^{(m)}, \delta, \ldots, \delta^{(m)} \in G$. As $\varepsilon \in G$, this form says only that the term t is a ready sequential component in an arbitrary position of r.

As $(k_1, \ldots, k_n) \in K_n(\Delta)$ and $only^{\{c,d\}}((ttr, c^{k_1}d \ldots c^{k_n}d)$ then r includes at least n sequential compositions accessed (see Definition 5.14) during the rewriting sequence from $tt\ r$ under $c^{k_1}d \ldots c^{k_n}d$.

Let us assume that $t_1, \ldots, t_{n-1} \in P$ for now. If s_i includes a sequential composition then s_i includes a non-trivial ready sequential component. According to Lemma 5.19, this non-trivial ready sequential component is deadlocked. Hence s_i includes a dead ready sequential component and all sequential compositions behind this component are dead. This means that the compositions are inaccessible under $c^{k_1}d \ldots c^{k_n}d$. Thus at most $i-1$ sequential compositions are accessed under I-actions. This is less than n and so it contradicts the property obtained in the previous paragraph. Hence $s_1, \ldots, s_{n-1} \in P$, $t, t_1, \ldots, t_{n-1} \in P \setminus \{\varepsilon\}$, and all dead ready sequential components of r are included in subterms $\delta, \ldots, \delta^{(m)}$. Further, similarly to the sequential compositions of s_i discussed above, all the sequential compositions of $\delta, \ldots, \delta^{(m)}$ are dead.

From Lemma 5.19 it follows that $t_i\|s_i$ (where $1 \leq i \leq n$) is a new ready sequential component appeared by accessing the i-th sequential compositions.

If t_i includes a sequential composition then it is not necessary to access all the compositions down to t_n – those of t_i $(i < n)$ will be accessed instead. A sequential composition in t_i also implies that t_i includes a non-trivial ready sequential component. This ready sequential component will be a proper subterm of $t_i\|s_i$ appearing by accessing the sequential composition before t_i.

As $(k_1, \ldots, k_n) \in K_n(\Delta)$ and $only^{\{e,f\}}(tt\ r, e^{k_1}f \ldots e^{k_n}f)$, r includes at least n sequential compositions accessed during the rewriting sequence from $tt\ r$ under $e^{k_1}f \ldots e^{k_n}f$.

We recall that all dead sequential compositions are in $\delta, \ldots, \delta^{(m)}$. From Lemma 5.25 it follows that all the sequential compositions behind a multiplicative dead ready sequential components are inaccessible. Hence, the only accessible sequential compositions of $\delta, \ldots, \delta^{(m)}$ are behind restricted dead ready sequential components. According to the definitions, all restricted dead ready sequential components and the terms behind them have already been created in t_{dr}. Hence, there are at most l such accessible compositions in r. Thus at least $|C| + 1$ $(= n - l)$ sequential compositions

of the subterm $((\cdots((t.t_1)\|s_1)\cdots.t_n)\|s_n)$ are accessed during a rewriting sequence from *tt* r under $e^{k_1}f\ldots e^{k_n}f$.

If there is t_i ($i \leq |C|+1$) with a sequential composition, then accessing the i-th composition "creates" the same ready sequential composition that appears as in the rewriting sequence under $c^{k_1}d\ldots c^{k_n}d$. Hence, a I-action can be performed and we derived a contradiction.

Therefore, $t_1,\ldots,t_{|C|+1} \in P$ and all the sequential compositions of $((\cdots((t.t_1)\|s_1)\cdots.t_{|C|+1})\|s_{|C|+1})$ are accessed during the sequence under $e^{k_1}f\ldots e^{k_n}f$.

For every $1 \leq i \leq |C|+1$, accessing the i-th composition have to be preceded by changing s_i; otherwise we get the same ready sequential component $t_i\|s_i$ as in the rewriting sequence under $c^{k_1}d\ldots c^{k_n}d$ and I-actions can be performed. The only possibility of forcing the preceding is to change the store during at least one action performed by s_i. Otherwise all the actions performed by s_i can be omitted without any effect on the rewriting sequence accessing the i-th composition. Hence we have to change a constraint at least $|C|+1$ times. It is the contradiction and Lemma 5.26 is proved. □

5.2.4 PDA Non-bisimilar to any wPAN

Example 5.27. (PDA non-wPAN) *Let us consider a PDA system of Example 2.13*

$$\begin{array}{lll}
U.X \overset{a}{\hookrightarrow} U.A.X & U.A \overset{a}{\hookrightarrow} U.A.A & U.B \overset{a}{\hookrightarrow} U.A.B \\
U.X \overset{b}{\hookrightarrow} U.B.X & U.A \overset{b}{\hookrightarrow} U.B.A & U.B \overset{b}{\hookrightarrow} U.B.B \\
U.X \overset{c}{\hookrightarrow} V.X & U.A \overset{c}{\hookrightarrow} V.A & U.B \overset{c}{\hookrightarrow} V.B \\
U.X \overset{d}{\hookrightarrow} W.X & U.A \overset{d}{\hookrightarrow} W.A & U.B \overset{d}{\hookrightarrow} W.B \\
V.X \overset{e}{\hookrightarrow} V & V.A \overset{a}{\hookrightarrow} V & V.B \overset{b}{\hookrightarrow} V \\
W.X \overset{f}{\hookrightarrow} W & W.A \overset{a}{\hookrightarrow} W & W.B \overset{b}{\hookrightarrow} W
\end{array}$$

but having $U.X.Y$ as the initial state and being extended with the following two rewrite rules:

$$V.Y \overset{x}{\hookrightarrow} U.X.Y \qquad\qquad W.Y \overset{x}{\hookrightarrow} U.X.Y$$

This system, denoted by Δ_4, behaves like that defined in Example 2.13, but whenever the original system terminates, the enhanced Δ_4 is restarted under the action x.

Lemma 5.28. *There is no wPAN Δ bisimilar to the PDA Δ_4 of Example 5.27.*

Proof. The proof is similar to the proof of Lemma 5.11 using the fact the PDA of Example 2.13 is not bisimilar to any PAN system.

To derive a contradiction assume a wPAN Δ bisimilar to the PDA Δ_4. As the weak state unit of Δ is finite then there exists a reachable state mt

of Δ such that every state reachable from mt has also m as its w-state component (the opposite would imply the infiniteness of the weak state unit). There exists a word $w \in \{a, b, c, d, e, f\}^*$ such that $mt \xrightarrow{w.x}{}^*_\Delta mt'$, where mt' is bisimilar to the state $U.X.Y$ of the PDA process Δ_4. If the rules labelled by the action x are removed from Δ and mt' is taken as the initial state, we obtain the system whose all reachable states have m as w-state component and which is bisimilar to the pushdown process of Example 2.13.

Now let Δ' be a PAN system with the initial state t' and with the set of rewrite rules consisting of the rules $l \overset{v}{\hookrightarrow} r$, where $(ml \overset{v}{\hookrightarrow} mr) \in \Delta$ and $v \in \{a, b, c, d, e, f\}$. It is obvious that this PAN system Δ' is bisimilar to the PDA system defined in Example 2.13 – a contradiction. □

5.2.5 Intuition and Conjectures

To accomplish the strictness/incomparability proof it remains to show a wPA non-fcPRS system and an sePA non-wPRS system. We conjecture that the following two systems exemplify the desired systems.

Example 5.29. (wPA non-fcPRS) *Let Δ be a wPA system with the initial state $pX\|Y$ and the the following rewrite rules.*

$$
\begin{array}{llll}
pX \overset{a}{\hookrightarrow} pA.X & pA \overset{a}{\hookrightarrow} pA.A & pB \overset{a}{\hookrightarrow} pA.B & pA \overset{a'}{\hookrightarrow} p\varepsilon \\
pX \overset{b}{\hookrightarrow} pB.X & pA \overset{b}{\hookrightarrow} pB.A & pB \overset{b}{\hookrightarrow} pB.B & pB \overset{b'}{\hookrightarrow} p\varepsilon \\
pY \overset{c}{\hookrightarrow} pY\|C & pC \overset{c'}{\hookrightarrow} p\varepsilon & & \\
pY \overset{d}{\hookrightarrow} oY & & &
\end{array}
$$

We conjecture that there is no fcPRS system bisimilar to this wPA Δ.

This system is composed of two subsystems running in parallel. The fist subsystem is a stack with "push" actions a, b and "pop" actions a', b'. The second subsystem forms a counter with an increment action c and a decrement action c'. Due to the rule with a label d, the system is deadlockable (see Definition 5.3). It is easy to see that for every reachable non-deadlocked state α, there is a sequence of actions $w \in \{a', b', c'\}^*$ such that $\alpha \xrightarrow{w}{}^* \beta$ and β is bisimilar to the initial state. Hence, similarly to Lemma 5.18, we can assume, without loss of generality, that a desired fcPRS system has all $x \in \{a, b, c, a', b', c'\}$ rules of the form

$$tt\ t_1 \overset{x}{\hookrightarrow} tt\ t_2\ .$$

Let us note that these rules cannot be forbidden by any value of a finite constraint store.

Assume an fcPRS system Δ' bisimilar to the wPA Δ of Example 5.29. Therefore, the system Δ' can perform an unbounded number of c actions.

An execution of arbitrary large number of c actions leads to a state from where exactly the same number of c' actions can be performed. This "unbounded" number has to be saved in a possibly very large term inducing the c' actions; let us call this term the *counter*.

Now, let us execute a very long sequence of a and b actions, an achieved state has to store this sequential information. Hence, a long sequence of sequential components forms a *stack* in the term part. While building this stack, it is impossible to carry the long "counter" subterm on the top of this stack. Hence, the "counter" term (inducing c' actions) and the top of the stack (inducing a sequence of a' and b' actions) are far away from each other (at any arbitrary distance), and so they cannot be both changed during one rewriting step. This is a contradiction with a deadlockable property and the type of all a, b, c, a', b', c' rules.

Example 5.30. (sePA non-wPRS) *Let Δ be a sePA system with the initial state $pX\|Y$ and the the following rewrite rules.*

$$pX \overset{a}{\hookrightarrow} pA.X \qquad pA \overset{a}{\hookrightarrow} pA.A \qquad pB \overset{a}{\hookrightarrow} pA.B \qquad pA \overset{a'}{\hookrightarrow} p\varepsilon$$
$$pX \overset{b}{\hookrightarrow} pB.X \qquad pA \overset{b}{\hookrightarrow} pB.A \qquad pB \overset{b}{\hookrightarrow} pB.B \qquad pB \overset{b'}{\hookrightarrow} p\varepsilon$$
$$pY \overset{c}{\hookrightarrow} pY\|C \qquad pC \overset{c'}{\hookrightarrow} p\varepsilon$$
$$pY \overset{d}{\hookrightarrow} oY$$

$$oA \overset{e}{\hookrightarrow} o\varepsilon \qquad oB \overset{e}{\hookrightarrow} o\varepsilon \qquad oC \overset{e}{\hookrightarrow} o\varepsilon$$
$$oX \overset{f}{\hookrightarrow} pX$$

We conjecture that there is no wPRS system bisimilar to this sePA Δ.

In this system the actions a, b, c, a', b', c' and d work in the same way as in the previous example. Moreover, after performing an action d the system is not deadlocked but the counter and the stack can be "discharged" using e actions and then an action f can "restart" the system. In other words, for every reachable state α, there is a sequence of actions $w \in \{d, e, f\}^*$ such that $\alpha \overset{w}{\longrightarrow}^* \beta$ and β is bisimilar to the initial state.

Assume a wPRS system Δ' bisimilar to the sePA Δ of Example 5.30. As the weak state unit of Δ' is finite, there exists a reachable state mt of Δ' such that every state reachable from mt has also m as its w-state component (the opposite would imply the infiniteness of the weak state unit). As every execution of Δ' can be prolong to reach a state bisimilar to the initial state, we can obtain a wPRS system whose all reachable states have the same w-states. Therefore, we derived that there is also a PRS system Δ'' bisimilar to the system Δ'. Removing all rewrite rules with the label e and all rewrite rules with the label f from the PRS system Δ'' we create a PRS system bisimilar to the wPA non-fcPRS system of Example 5.29. This is a contradiction with our previous assumption.

Chapter 6

Strong Bisimulation Equivalence

In this chapter we focus on (un)decidability as well as complexity boundaries on strong bisimilarity between two states of a given extended PRS. Concerning this research area we have not reach any new results, therefore we only summarises all known results and briefly mention what we have tried to do and where our attempt has failed.

6.1 Motivation

Equivalence checking is one of the main streams in verification of concurrent systems. It aims at demonstrating some semantic equivalence between two systems, one of which is usually considered as representing the specification, while the other as its implementation or refinement. The semantic equivalences are designed to correspond to the system behaviours at the desired level of abstraction; the most prominent ones being strong and weak bisimulations. We mention some results on equivalence checking with strong bisimilarity here. The weak bisimilarity is discussed in the next chapter. For the other equivalence checking problems we refer to surveys [BCMS01, Srb04] for example.

The bisimilarity problem for BPA is known to be in 2-EXPTIME [BCS95] and PSPACE-hard [Srb02c]. The first result showing that the bisimilarity problem for PDA is decidable was published in [Sén98]. Later on, bisimilarity was determined to be EXPTIME-hard for PDA [KM02]. EXPTIME-hardness is also the best known lower bound for the PAD, fcPAD, and wPAD classes where, moreover, the decidability left open.

Considering the parallel systems, the strong bisimilarity problem is undecidable for seBPP (also known as PPDA) [Mol96] using the technique introduced in [Jan95b]. However, the strong bisimilarity is known to be decidable for BPP [CHM93]. Moreover, it was shown that the problem

belongs to PSPACE [Jan03]. Combining with PSPACE-hardness result of [Srb02b], PSPACE-completeness was established. The result of [Srb02b] (PSPACE-hardness for BPP) presents also the best known lower bounds for bisimilarity for PA, fcPA, wPA, fcBPP, and wBPP, but in these cases, contrary to BPP, the decidability of bisimilarity left open.

6.2 Definition of Strong Bisimilarity

Bisimilarity was originally introduced by Park [Par81] and Milner [Mil89].

Definition 6.1. *A binary relation R on set of states S is a* strong bisimulation *if and only if for each $(\alpha, \beta) \in R$ the following conditions hold:*

- $\forall\, \alpha' \in S, a \in Act : \alpha \xrightarrow{a} \alpha'$ *implies* $(\exists \beta' \in S : \beta \xrightarrow{a} \beta' \wedge (\alpha', \beta') \in R)$
- $\forall\, \beta' \in S, a \in Act : \beta \xrightarrow{a} \beta'$ *implies* $(\exists \alpha' \in S : \alpha \xrightarrow{a} \alpha' \wedge (\alpha', \beta') \in R)$

States α and β are strongly bisimilar*, written $\alpha \sim \beta$, if and only if $(\alpha, \beta) \in R$ for some strong bisimulation R. Two labelled transition systems are* strongly bisimilar *if and only if its initial states are strongly bisimilar.*

Problem:	*Strong bisimilarity problem for an extended (α, β)-PRS class*
Instance:	An extended (α, β)-PRS system Δ and two of its states $mt, m't'$
Question:	Are the two states mt and $m't'$ strongly bisimilar?

6.3 Summary

As we have mentioned in Section 6.1, the problem for BPP is in PSPACE but it is undecidable for the state extended variant (seBPP). This encouraged us to focus on the strong bisimilarity problem for wBPP staying in between these two classes. We tried to prove the problem is decidable.

The crucial difference between BPP and wBPP is as follows. It is known that the following congruence holds for every parallel terms t_1, t_2, t_3, and t_4 of a BPP system.

$$t_1 \sim t_2 \ \wedge \ t_3 \sim t_4 \implies t_1 \| t_3 \sim t_2 \| t_4$$

But in the context of wBPP systems

$$pt_1 \sim pt_2 \text{ does not imply either } pt_1 \| t \sim pt_2 \| t.$$

This can be shown e.g. by the following example.

Example 6.2. *Let Δ be a wBPP with the following rewrite rules.*

$$pA \overset{a}{\hookrightarrow} p\varepsilon \qquad pB \overset{a}{\hookrightarrow} q\varepsilon \qquad qC \overset{c}{\hookrightarrow} qC$$

It is easy to see that

$$pA \sim pB \ \wedge \ pA\|C \not\sim pB\|C.$$

This behaviour of wBPP systems counterwork all our attempts to adopt known proves for BPP such as a tableau decision method [CHM93] or a DD-function characterisation [Jan03]. Therefore the decidability of bisimilarity problem for wBPP left open.

Figure 6.1 shows (un)decidability borders of strong bisimilarity problems.

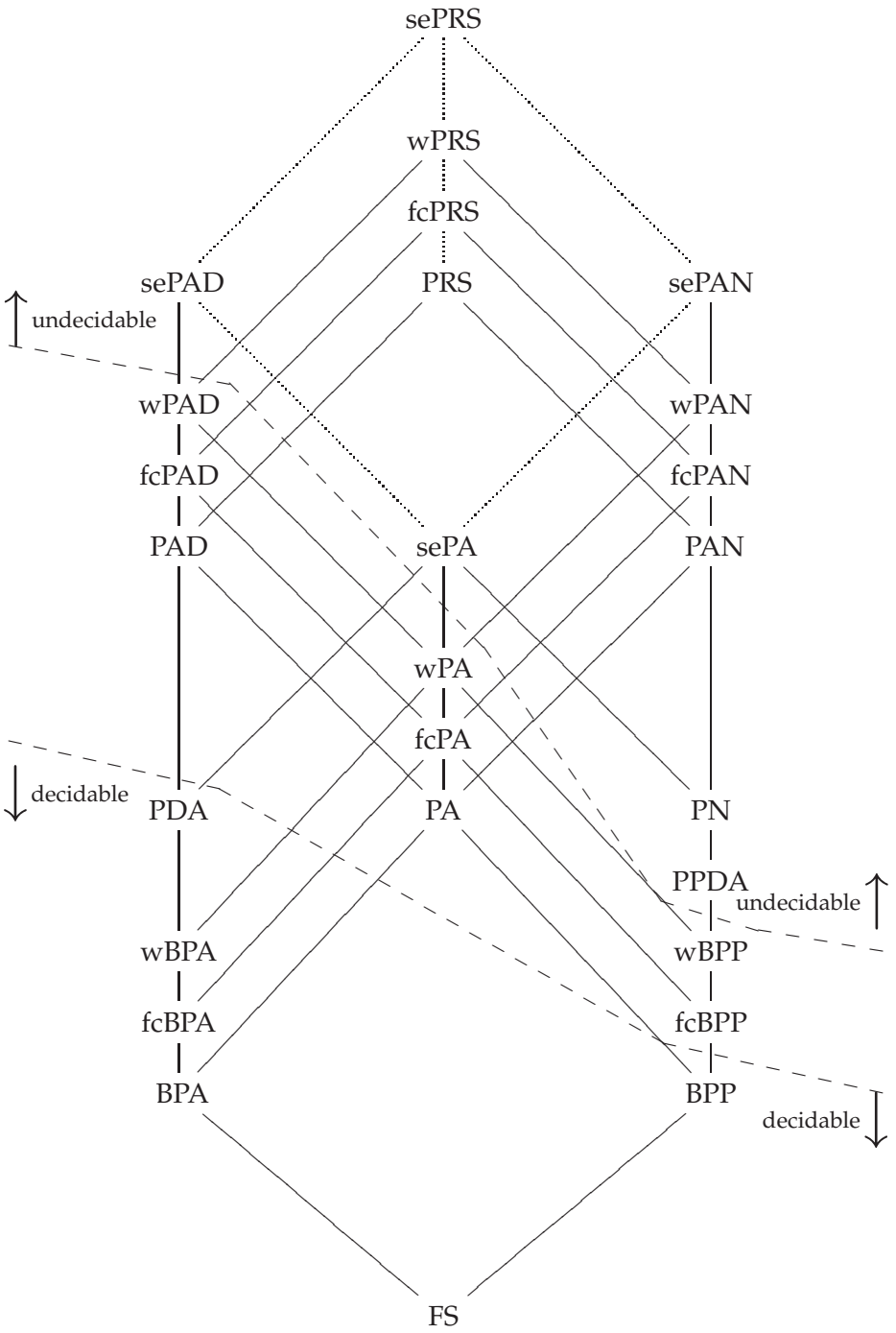


Figure 6.1: The extended PRS-hierarchy with (un)decidability boundaries of strong bisimilarity.

Chapter 7

Weak Bisimulation Equivalence

Weak bisimilarity is one of the most studied behavioural equivalences. This equivalence is undecidable for *pushdown processes* (PDA), *process algebras* (PA), and *parallel pushdown processes* (PPDA). Its decidability is an open question for *basic process algebras* (BPA) and *basic parallel processes* (BPP). We move the undecidability border towards these classes by showing that the equivalence remains undecidable for weakly extended versions of BPA and BPP. In fact, we show that the weak bisimulation equivalence problem is undecidable even for normed subclasses of BPA and BPP extended with finite constraint systems.

7.1 Motivation

Weak bisimulation equivalence is one of the semantic equivalences with a *silent* action. These equivalences are based on a notion of observable behaviour of systems: only the interactions of the system with the environment (observer) are observable. The internal structure of the system is not considered observable and system internal activities are modelled by silent (τ) actions which can precede and/or follow any observable action. For an overview of equivalences with silent moves and more general setting with respect to various testing scenarios we refer to [vG93].

Now, we mention some of the results on checking with weak bisimulation equivalence (bisimilarity). Regarding sequential systems, i.e. those without parallel composition, the weak bisimilarity problem is undecidable for PDA even for the normed case [Srb02d]. However, it is conjectured [May05] that weak bisimilarity is decidable for *basic process algebras* (BPA); the best known lower bound is EXPTIME-hardness [May05].

Considering parallel systems, even strong bisimilarity is undecidable for *parallel pushdown processes* as shown in (PPDA) [Mol96] using the tech-

nique introduced in [Jan95b]. However, it is conjectured [Jan03] that the weak bisimilarity problem is decidable for *basic parallel processes* (BPP); the best known lower bound is PSPACE-hardness [Srb03a].

For the simplest systems combining both parallel and sequential operators, called PA processes [BW90], the weak bisimilarity problem is undecidable [Srb03b]. It is an open question for the normed PA; the best known lower bound is EXPTIME-hardness [May05].

In this chapter, we move the undecidability border of the weak bisimilarity problem towards the classes of BPA and BPP, where the problem is conjectured to be decidable. Section 7.3 contains (relatively simple) proofs of undecidability of the considered problem for the weakly extended versions of BPA (wBPA) and BPP (wBPP). In Section 7.4, we strengthen the results by showing that even for more restricted systems, namely for normed fcBPA and normed fcBPP systems, this equivalence remains undecidable. In fact, the result is not new for wBPA due to the following reasons: Mayr [May05] has shown that adding a finite-state unit with 2 states (i.e. of the minimal non-trivial size) to a BPA process already makes weak bisimilarity undecidable. Our inspection of his proof shows that a finite state unit used in the proof is weak and so the result is valid for wBPA as well.

7.2 Definition of Weak Bisimilarity

It is common to admit silent transitions to model the internal unobservable evolution of a system. In standard automata theory these are typically referred to as "epsilon" transitions, but in concurrency theory they are commonly represented by a distinguished action $\tau \in Act$.

Definition 7.1. *Let $(S, Act, \longrightarrow, \alpha_0)$ be an LTS and $\tau \in Act$ be a distinguished action. A* relation of observable transitions $\Longrightarrow \subseteq (S \times Act \times S)$ *is defined as follows:*

$$\alpha \stackrel{\tau}{\Longrightarrow} \beta \quad \textit{if and only if} \quad \alpha \stackrel{w}{\longrightarrow}{}^{*} \beta \textit{ where } w \in \{\tau\}^*$$

$$\alpha \stackrel{a}{\Longrightarrow} \beta \quad \textit{if and only if} \quad \alpha \stackrel{w}{\longrightarrow}{}^{*} \beta \textit{ where } w \in \{\tau\}^* \cdot \{a\} \cdot \{\tau\}^* \textit{ and } a \neq \tau.$$

We also use a natural generalisation $\alpha \stackrel{w}{\Longrightarrow} \beta$ for finite sequences $w \in Act^*$.

Definition 7.2. *A binary relation R on set of states S is a* weak bisimulation *if and only if for each $(\alpha, \beta) \in R$ the following conditions hold:*

- $\forall\, \alpha' \in S, a \in Act : \alpha \stackrel{a}{\longrightarrow} \alpha'$ *implies* $(\exists \beta' \in S : \beta \stackrel{a}{\Longrightarrow} \beta' \wedge (\alpha', \beta') \in R)$
- $\forall\, \beta' \in S, a \in Act : \beta \stackrel{a}{\longrightarrow} \beta'$ *implies* $(\exists \alpha' \in S : \alpha \stackrel{a}{\Longrightarrow} \alpha' \wedge (\alpha', \beta') \in R)$

States α and β are weakly bisimilar*, written $\alpha \approx \beta$, if and only if $(\alpha, \beta) \in R$ for some weak bisimulation R. Two labelled transition systems are* weakly bisimilar *if and only if their initial states are weakly bisimilar.*

We use a characterisation of weak bisimilarity in terms of a *bisimulation game*, see e.g. [Sti96]. This is a two-player game between an *attacker* and a *defender* played in rounds on pairs of states of a considered labelled transition system. In a round starting at a pair of states (α_1, α_2), the attacker first chooses $i \in \{1, 2\}$, an action $a \in Act$, and a state α_i' such that $\alpha_i \xrightarrow{a} \alpha_i'$. The defender then has to choose a state α_{3-i}' such that $\alpha_{3-i} \stackrel{a}{\Longrightarrow} \alpha_{3-i}'$. The states α_1', α_2' form a pair of starting states for the next round. A *play* is a maximal sequence of pairs of states chosen by players in the given way. The defender is the winner of every infinite play. A finite game is lost by the player who cannot make any choice satisfying the given conditions. It can be shown that two states α_1, α_2 of a labelled transition system are not weakly bisimilar if and only if the attacker has a winning strategy for the bisimulation game starting in these states.

We study the following problems for extended (α, β)-PRS classes.

Problem:	*Weak bisimilarity problem for an extended (α, β)-PRS class*
Instance:	An extended (α, β)-PRS system Δ and two of its states $mt, m't'$
Question:	Are the two states mt and $m't'$ weakly bisimilar?

7.3 Undecidability of Weak Bisimilarity

In this section, we show that weak bisimilarity is undecidable for the classes wBPA and wBPP.

7.3.1 wBPA

In [May05] Mayr studied the question of how many control states are needed in a pushdown automaton to make weak bisimilarity undecidable.

Theorem 7.3 ([May05], Theorem 29)**.** *Weak bisimilarity is undecidable for pushdown automata with only 2 control states.*

The proof is done by a reduction of Post's correspondence problem to the weak bisimilarity problem for PDA. The constructed pushdown automaton has only two control states, p and q. Quick inspection of the construction shows that the resulting pushdown automata are in fact wBPA systems as there is no transition rule changing q to p and each rule has only one process constant on the left hand side. Hence Mayr's theorem can be reformulated as follows.

Theorem 7.4. *Weak bisimilarity is undecidable for wBPA systems with only 2 control states.*

7.3.2 wBPP

We show that the non-halting problem for Minsky 2-counter machines can be reduced to the weak bisimilarity problem for wBPP. First, let us recall the notions of Minsky 2-counter machine and the non-halting problem.

A *Minsky 2-counter machine*, or a *machine* for short, is a finite sequence

$$\begin{aligned} N = \;& l_1 : i_1, \\ & l_2 : i_2, \\ & \ldots \\ & l_{n-1} : i_{n-1}, \\ & l_n : \texttt{halt} \end{aligned}$$

where $n \geq 1$, $l_1, l_2, \ldots, l_n$ are *labels*, and each i_j is an instruction of one of the following forms:

- *increment*:
$$\texttt{c}_k\texttt{:=c}_k\texttt{+1; goto } l_r$$
- *test-and-decrement*:
$$\texttt{if c}_k\texttt{>0 then c}_k\texttt{:=c}_k\texttt{-1; goto } l_r \texttt{ else goto } l_s$$

where c_k are *counters*, $k \in \{1, 2\}$, and $1 \leq r, s \leq n$.

The semantics of a machine N is given by a labelled transition system. The states are *configurations* of the form (l_j, v_1, v_2) where l_j is a label of an instruction to be executed and v_1, v_2 are nonnegative integers representing current values of counters $\texttt{c}_1$ and $\texttt{c}_2$, respectively. The transition relation is the smallest relation satisfying the following conditions: if i_j is an instruction of the form

- $\texttt{c}_1\texttt{:=c}_1\texttt{+1; goto } l_r$, then
$(l_j, v_1, v_2) \xrightarrow{inc} (l_r, v_1 + 1, v_2)$ for all $v_1, v_2 \geq 0$;
- $\texttt{if c}_1\texttt{>0 then c}_1\texttt{:=c}_1\texttt{-1; goto } l_r \texttt{ else goto } l_s$, then
$(l_j, v_1 + 1, v_2) \xrightarrow{dec} (l_r, v_1, v_2)$ for all $v_1, v_2 \geq 0$ and
$(l_j, 0, v_2) \xrightarrow{zero} (l_s, 0, v_2)$ for all $v_2 \geq 0$;

and the analogous condition for instructions manipulating $\texttt{c}_2$. We say that the (computation of) machine N *halts* if there are numbers $v_1, v_2 \geq 0$ such that $(l_1, 0, 0) \longrightarrow^* (l_n, v_1, v_2)$. Let us note that the system is deterministic, i.e. for every configuration there is at most one transition leading from the configuration.

The *non-halting problem* is to decide whether a given machine N does not halt. The problem is undecidable [Min67].

Let us fix a machine $N = l_1 : i_1,\ l_2 : i_2,\ \ldots,\ l_{n-1} : i_{n-1},\ l_n : \texttt{halt}$. We construct a wBPP system Δ such that its states

$simL_1$ and $simL'_1$ are weakly bisimilar if and only if N does not halt.

Roughly speaking, we create a set of wBPP rules allowing us to simulate the computation of N by two separate sets of terms. If the `halt` instruction is reached in the computation of N, the corresponding term from one set can perform the action $halt$, while the corresponding term from the other set can perform the action $halt'$. Therefore, the starting terms are weakly bisimilar if and only if the machine does not halt.

The wBPP system Δ we are going to construct uses five control states, namely $sim, check_1, check'_1, check_2, check'_2$. We associate each label l_j and each counter $\mathtt{c}_k$ with process constants L_j, L'_j and X_k, Y_k respectively. A parallel composition of x copies of X_k and y copies of Y_k, written $X_k^x \| Y_k^y$, represents the fact that the counter $\mathtt{c}_k$ has the value $x - y$. Hence, terms $simL_j \| X_1^{x_1} \| Y_1^{y_1} \| X_2^{x_2} \| Y_2^{y_2}$ and $simL'_j \| X_1^{x_1} \| Y_1^{y_1} \| X_2^{x_2} \| Y_2^{y_2}$ are associated with a configuration $(l_j, x_1 - y_1, x_2 - y_2)$ of the machine N. Some rules contain auxiliary process constants. In what follows, β stands for a term of the form $\beta = X_1^{x_1} \| Y_1^{y_1} \| X_2^{x_2} \| Y_2^{y_2}$. The control states $check_k, check'_k$ for $k \in \{1, 2\}$ are intended for testing emptiness of the counter $\mathtt{c}_k$. The only rules applicable in these control states are:

$$check_1 X_1 \overset{chk_1}{\hookrightarrow} check_1 \varepsilon \qquad check_2 X_2 \overset{chk_2}{\hookrightarrow} check_2 \varepsilon$$
$$check'_1 Y_1 \overset{chk_1}{\hookrightarrow} check'_1 \varepsilon \qquad check'_2 Y_2 \overset{chk_2}{\hookrightarrow} check'_2 \varepsilon$$

One can readily confirm that $check_k \beta \approx check'_k \beta$ if and only if the value of $\mathtt{c}_k$ represented by β equals zero.

In what follows we construct a set of wBPP rules for each instruction of the machine N. At the same time we argue that the only chance for the attacker to win the bisimulation game is to simulate the machine without cheating as every cheating can be punished by the defender's victory. Therefore this attacker's strategy is winning if and only if the machine halts.

Halt: l_n : `halt`

Halt instruction is translated into the following two rules:

$$simL_n \overset{halt}{\hookrightarrow} sim\varepsilon \qquad simL'_n \overset{halt'}{\hookrightarrow} sim\varepsilon$$

Clearly, *the states $simL_n \| \beta$ and $simL'_n \| \beta$ are not weakly bisimilar.*

Increment: l_j : $\mathtt{c}_k$`:=`$\mathtt{c}_k$`+1; goto` l_r

To each such an instruction of the machine N we add to Δ the following rules:

$$simL_j \overset{inc}{\hookrightarrow} simL_r \| X_k \qquad simL'_j \overset{inc}{\hookrightarrow} simL'_r \| X_k$$

Obviously, *every round of the bisimulation game starting at states* $simL_j\|\beta$ *and* $simL'_j\|\beta$ *ends up in states* $simL_r\|X_k\|\beta$ *and* $simL'_r\|X_k\|\beta$.

Test-and-decrement:

l_j: `if` $\mathtt{c}_k$`>0 then` $\mathtt{c}_k$`:=` $\mathtt{c}_k$`-1; goto` l_r `else goto` l_s

To each such an instruction of the machine N we add to Δ two sets of rules, one for the $\mathtt{c}_k > 0$ case and the other for the $\mathtt{c}_k = 0$ case. The wBPP formalism has no power to rewrite a process constant corresponding to a label l_j and to check whether $\mathtt{c}_k > 0$ at the same time. Therefore, in the bisimulation game it is the attacker who has to decide whether $\mathtt{c}_k > 0$ holds or not, i.e. whether he will play an action dec or an action $zero$. We show that whenever the attacker tries to cheat, the defender can win the game.

At this point our construction of wBPP rules uses a variant of the technique called *defender's choice* [JS04]. In a round starting at the pair of states α_1, α_2, the attacker is forced to choose one specific transition (indicated by a wavy arrow henceforth). If he chooses a different transition, say $\alpha_k \xrightarrow{a} \alpha$ where $k \in \{1, 2\}$, then there exists a transition $\alpha_{3-k} \xrightarrow{a} \alpha$ that enables the defender to reach the same state and win the play. The name of this technique refers to the fact that after the attacker chooses the specific transition, the defender can choose an arbitrary transition with the same label. These transitions are indicated by solid arrows. The dotted arrows stands for auxiliary transitions which compel the attacker to play the specific transition.

First, we discuss the following rules designed for the case when $\mathtt{c}_k > 0$.

$$
\begin{array}{lll}
simL_j \overset{dec}{\hookrightarrow} simA_{k,r} & simA_{k,r} \overset{dec}{\hookrightarrow} check_k\varepsilon & simB_{k,r} \overset{dec}{\hookrightarrow} simL_r\|Y_k \\
simL_j \overset{dec}{\hookrightarrow} simB_{k,r} & simA_{k,r} \overset{dec}{\hookrightarrow} simL'_r\|Y_k & simB_{k,r} \overset{dec}{\hookrightarrow} simL'_r\|Y_k \\
simL'_j \overset{dec}{\hookrightarrow} simA_{k,r} & simA_{k,r} \overset{dec}{\hookrightarrow} check'_k\varepsilon & simB_{k,r} \overset{dec}{\hookrightarrow} check'_k\varepsilon \\
simL'_j \overset{dec}{\hookrightarrow} simB_{k,r} & & simC_{k,r} \overset{dec}{\hookrightarrow} simL'_r\|Y_k \\
simL'_j \overset{dec}{\hookrightarrow} simC_{k,r} & & simC_{k,r} \overset{dec}{\hookrightarrow} check'_k\varepsilon
\end{array}
$$

The situation is depicted in Figure 7.1.

Let us assume that in a round starting at states $simL_j\|\beta, simL'_j\|\beta$ the attacker decides to perform the action dec. Due to the principle of defender's choice employed here, the attacker has to play the transition $simL'_j\|\beta \xrightarrow{dec}_{\Delta} simC_{k,r}\|\beta$, while the defender can choose between the transitions leading from $simL_j\|\beta$ either to $simA_{k,r}\|\beta$ or to $simB_{k,r}\|\beta$. Thus, the round will finish either in states $simA_{k,r}\|\beta, simC_{k,r}\|\beta$ or in states $simB_{k,r}\|\beta, simC_{k,r}\|\beta$. Using the defender's choice again, one can easily see that the next round ends up in $check_k\beta$ or $simL_r\|Y_k\|\beta$, and $simL'_r\|Y_k\|\beta$

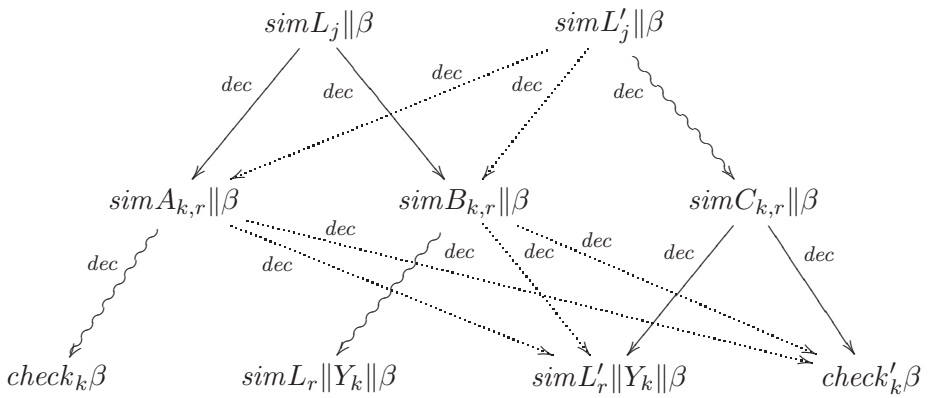


Figure 7.1: Decrement actions simulating a test-and-decrement instruction.

or $check'_k\beta$. The exact combination is chosen by the defender. The defender will not choose any pair of states where one control state is sim and the other is not as such states are clearly not weakly bisimilar. Hence, the two considered rounds of the bisimulation game end up in a pair of states either $simL_r\|Y_k\|\beta, simL'_r\|Y_k\|\beta$ or $check_k\beta, check'_k\beta$. The latter pair is weakly bisimilar if and only if the value of $\mathtt{c}_k$ represented by β is zero, i.e. iff the attacker cheats when he decides to play an action dec. This means that *if the attacker cheats, the defender wins. If the attacker plays the action* dec *correctly, the only chance for either player to force a win is to finish these two rounds in states* $simL_r\|Y_k\|\beta, simL'_r\|Y_k\|\beta$ *corresponding to the correct simulation of an test-and-decrement instruction with a label* l_j.

Now, we focus on the following rules designed for the $\mathtt{c}_k = 0$ case:

$$
\begin{array}{lll}
simL_j \overset{zero}{\hookrightarrow} simD_{k,s} & simD_{k,s} \overset{zero}{\hookrightarrow} check_k\varepsilon & simE_{k,s} \overset{zero}{\hookrightarrow} simL_s \\
simL_j \overset{zero}{\hookrightarrow} simE_{k,s} & simD_{k,s} \overset{zero}{\hookrightarrow} simL'_s & simE_{k,s} \overset{zero}{\hookrightarrow} simL'_s \\
simL'_j \overset{zero}{\hookrightarrow} simD_{k,s} & simD_{k,s} \overset{zero}{\hookrightarrow} simG_k & simE_{k,s} \overset{zero}{\hookrightarrow} simG_k \\
simL'_j \overset{zero}{\hookrightarrow} simE_{k,s} & simF_{k,s} \overset{zero}{\hookrightarrow} simL'_s & simG_k \overset{\tau}{\hookrightarrow} simG_k\|Y_k \\
simL'_j \overset{zero}{\hookrightarrow} simF_{k,s} & simF_{k,s} \overset{zero}{\hookrightarrow} simG_k & simG_k \overset{\tau}{\hookrightarrow} check'_k Y_k
\end{array}
$$

The situation is depicted in Figure 7.2.

Let us assume that the attacker decides to play the action $zero$. The defender's choice technique allows the defender to control the two rounds of the bisimulation game starting at states $simL_j\|\beta$ and $simL'_j\|\beta$. The two rounds end up in a pair of states $simL_s\|\beta, simL'_s\|\beta$ or in a pair of the form $check_k\beta, check'_k Y_k^m\|\beta$ where $m \geq 1$; all the other choices of the defender lead to his loss. As in the previous case, the latter possibility is designed to punish any possible attacker's cheating. The attacker is cheating if he plays

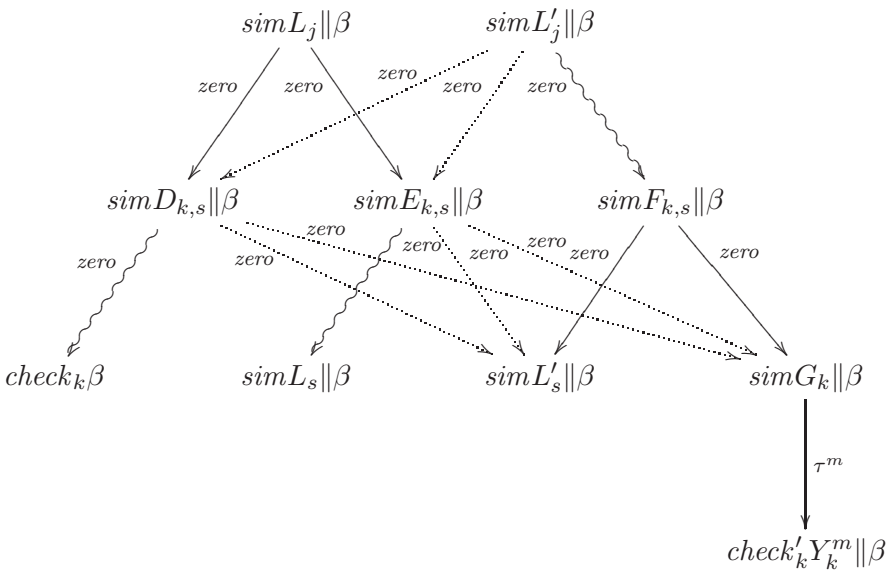


Figure 7.2: Zero actions simulating a test-and-decrement instruction.

the action $zero$ and the value of $\mathtt{c}_k$ represented by β, say v_k, is positive[1]. In such a case, the defender can control the two rounds to end up in states $check_k\beta$, $check'_kY_k^{v_k}\|\beta$ which are weakly bisimilar. If the attacker plays correctly, i.e. the value of $\mathtt{c}_k$ represented by β is zero, then the defender has to control the two discussed rounds to end up in states $simL_s\|\beta$, $simL'_s\|\beta$ as the states $check_k\beta$, $check'_kY_k^m\|\beta$ are not weakly bisimilar for any $m \geq 1$. To sum up, *the attacker's cheating can be punished by defender's victory. If the attacker plays correctly, the only chance for both players to win is to end up after the two rounds in states $simL_s\|\beta$, $simL'_s\|\beta$ corresponding to the correct simulation of the considered instruction.*

It has been argued that if each of the two players wants to win, then both players will correctly simulate the computation of the machine N. The computation is finite if and only if the machine halts. The states $simL_1$ and $simL'_1$ are not weakly bisimilar in this case. If the machine does not halt, the play is infinite and the defender wins. Hence, the two states are weakly bisimilar in this case. In other words, *the states $simL_1$ and $simL'_1$ of the constructed wBPP Δ are weakly bisimilar if and only if the Minsky 2-counter machine N does not halt.* Hence, we have proved the following theorem.

Theorem 7.5. *Weak bisimilarity is undecidable for wBPP systems.*

[1]We do not have to consider the case when β represents a negative value of $\mathtt{c}_k$ as such a state is reachable in the game starting in states $simL_1, simL'_1$ only by unpunished cheating.

7.4 Weak Bisimilarity for More Restricted Classes

Here, we strengthen the results of the previous section. We will show that weak bisimilarity remains undecidable for both fcBPP and fcBPA systems.

Definition 7.6. *An (α, β)-fcPRS Δ is* normed in a state $m_0 t_0$ *of Δ if and only if, for all states mt satisfying $m_0 t_0 \longrightarrow_\Delta^* mt$, it holds that $mt \longrightarrow_\Delta^* o\varepsilon$ for some $o \in C(\Delta)$.*

Moreover, we show that weak bisimilarity remains undecidable even for their respective normed versions (i.e. if, in an instance of the weak bisimilarity problem, a given fcBPP/fcBPA system is normed in both given states). Hence, we show that weak bisimilarity is undecidable for normed wBPP and normed wBPA as well.

7.4.1 Normed fcBPP

In this subsection, we show that weak bisimilarity is undecidable for normed fcBPP systems.

Let Δ be the wBPP system constructed in Subsection 7.3.2. We recall that given any fixed Minsky machine N, we have constructed a wBPP system Δ such that its states $simL_1$ and $simL_1'$ are weakly bisimilar if and only if N does not halt. Based on Δ, we now construct a fcBPP Δ' and two of its states $simL_1 \| D$ and $simL_1' \| D$ such that they satisfy the same condition as given in the previous paragraph and moreover Δ' is normed in both of the states $simL_1 \| D$ and $simL_1' \| D$.

Let $Const(\Delta') = \{D\} \cup Const(\Delta)$ and $Act(\Delta') = \{norm\} \cup Act(\Delta)$, where $D \notin Const(\Delta)$ is a fresh process constant and $norm \notin Act(\Delta)$ is a fresh action. The constraint system of Δ' is depicted in Figure 7.3.

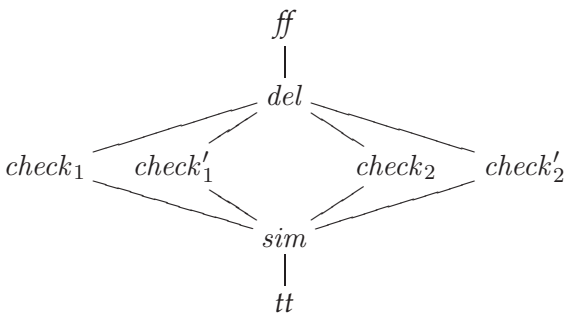


Figure 7.3: The constraint system of Δ'.

The set of rewrite rules of Δ' consists of all the rewrite rules of Δ and

moreover the following rules are added:

(1) $ttD \overset{norm}{\hookrightarrow} delD$,
(2) $delX \overset{\tau}{\hookrightarrow} del\varepsilon$ for all $X \in Const(\Delta')$,
(3) $delX \overset{a}{\hookrightarrow} delX$ for all $X \in Const(\Delta')$ and $a \in Act(\Delta)$.

The process constant D enables the $norm$ action changing the value of the store onto del. Starting in the state $simL_1 \| D$ or $simL'_1 \| D$, every reachable state includes the process constant D or the current value of the store has been already changed onto del. Whenever the value of the store is set to del, the rules of type (2) can be used to make the state normed. Hence, Δ' is normed in both of the states $simL_1 \| D$ and $simL'_1 \| D$.

It remains to show that these new rules do not affect the bisimulation game. Let us note that $del \vdash m$ and $del \wedge n \neq ff$ for every $m, n \in \{tt, sim, check_1, check'_1, check_2, check'_2\}$. Therefore, changing the store to del value does not forbid an application of the Δ rules – those with a label inc, dec, $zero$, chk_1, etc. But the rewrite rules of the type (3) cause that using of the $norm$ action in the game leads into weakly bisimilar states without respect to an application of the Δ rules. As the attacker can only decide to perform the $norm$ action, this reconstruction of Δ onto Δ' does not change the winning strategies discussed in Subsection 7.3.2. Hence the Theorem 7.5 can be strengthen as follows.

Theorem 7.7. *Weak bisimilarity is undecidable for normed fcBPP systems.*

7.4.2 Normed fcBPA

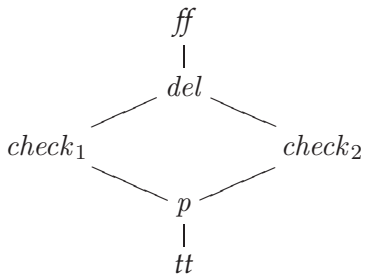
In this subsection, we show that the problem remains undecidable for the case of normed fcBPA systems. Our proof is a slightly extended translation of the proof for PDA of [May05] into fcBPA framework. We used the notation of [May05] to make the proofs comparable.

Our proof (as well as Mayr's one) is based on a reduction of Post's correspondence problem, which is known to be undecidable [HU79].

Problem:	*Post's correspondence problem (PCP)*
Instance:	A non-unary alphabet Σ and two ordered sets of words $\mathcal{A} = \{u_1, \ldots, u_n\}$ and $\mathcal{B} = \{v_1, \ldots, v_n\}$ where $u_i, v_i \in \Sigma^+$
Question:	Do there exist finitely many indices $i_1, \ldots, i_m \in \{1, \ldots, n\}$ such that $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$?

Given any instance of PCP we now construct a normed fcBPA Δ and two of its states $pTB, pT'B$ such that pTB and $pT'B$ are weakly bisimilar if and only if the instance of PCP has a solution.

A constraint system of Δ contains elements *tt*, p, $check_1$, $check_2$, del, and *ff* that are ordered as follows.



We use process constants $T, T', T_1, T_1', T_2, T_2', G_l, G_r, B$ and U_i, V_i, for each $1 \leq i \leq n$. Actions of Δ are $a, b, c, \tau, norm, 1, \ldots, n$, and the letters of Σ. In what follows, $\mathcal{U}$ stands for a sequential term of process constants of $\{U_i \mid 1 \leq i \leq n\}$ and similarly $\mathcal{V}$ stands for a sequential term of process constants of $\{V_i \mid 1 \leq i \leq n\}$.

Now, we construct a set of rewrite rules Δ. The rules of types (1)–(10) are exactly the same as those of Mayr's proof and forms a defender's choice construction.

(1)	$pT \overset{a}{\hookrightarrow} pT_1$	
(2)	$pT \overset{\tau}{\hookrightarrow} pG_r$	
(3)	$pT' \overset{\tau}{\hookrightarrow} pG_r$	
(4)	$pG_r \overset{\tau}{\hookrightarrow} pG_rV_i$	for all $i \in \{1, \ldots, n\}$
(5)	$pG_r \overset{a}{\hookrightarrow} pT_1'$	
(6)	$pT_1 \overset{a}{\hookrightarrow} pG_l$	
(7)	$pT_1' \overset{a}{\hookrightarrow} pG_lB$	
(8)	$pT_1' \overset{a}{\hookrightarrow} pT_2'$	
(9)	$pG_l \overset{\tau}{\hookrightarrow} pG_lU_i$	for all $i \in \{1, \ldots, n\}$
(10)	$pG_l \overset{\tau}{\hookrightarrow} pT_2$	

If there is a solution of the instance of PCP, the defender can use these rules to finish the first two rounds of the bisimulation game (starting in pTB and $pT'B$) in states $pT_2\mathcal{U}B$ and $pT_2'\mathcal{V}B$, where $\mathcal{U}$ and $\mathcal{V}$ form a solution of the PCP instance. The discussed first two rounds of the bisimulation game are depicted in Figure 7.4. We use the same notation for arrows as in Subsection 7.3.2.

The following six rules form two subsequent rounds of the bisimulation game and allow attacker to decide whether to check equality of indices or equality of the words of $\mathcal{U}$ and $\mathcal{V}$. In the first case, the attacker uses action b leading to the constraint $check_1$, while the second possibility is labelled by

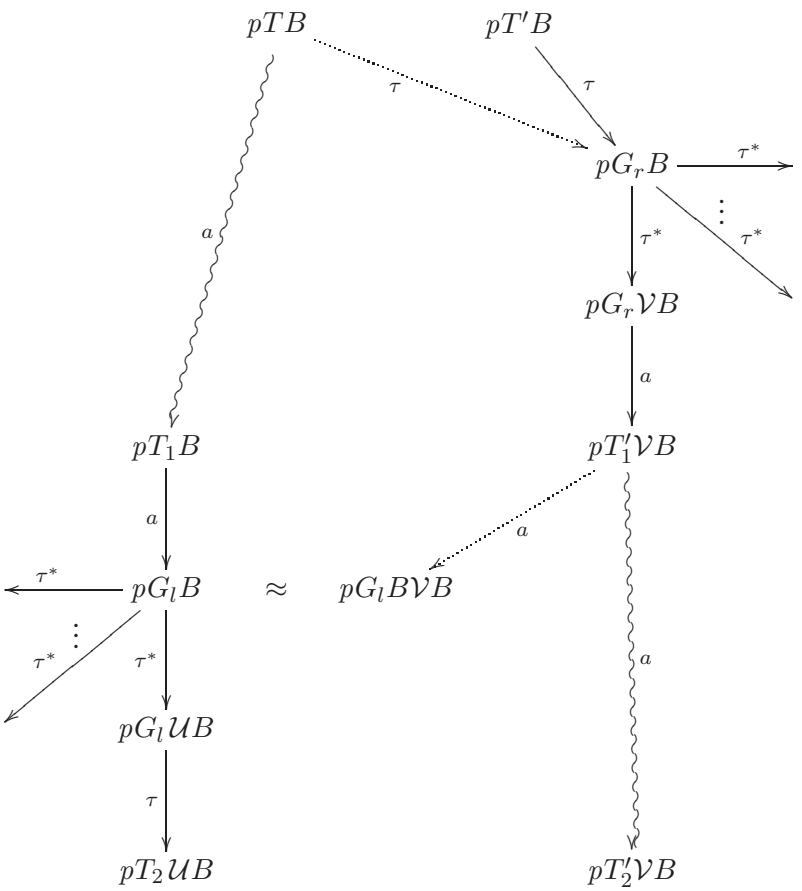


Figure 7.4: The first two rounds of the bisimulation game.

c and ends in the constraint $check_2$. The rewrite rules are as follows.

(11)	$pT_2 \overset{a}{\hookrightarrow} pT_3$	(12)	$pT_2' \overset{a}{\hookrightarrow} pT_3'$
(13)	$pT_3 \overset{b}{\hookrightarrow} check_1\varepsilon$	(14)	$pT_3' \overset{b}{\hookrightarrow} check_1\varepsilon$
(15)	$pT_3 \overset{c}{\hookrightarrow} check_2\varepsilon$	(16)	$pT_3' \overset{c}{\hookrightarrow} check_2\varepsilon$

Now, we list the rules that serve for the checking phases mentioned in the previous paragraph. In rules (19) and (20), we use a short notation that can be easily expressed by standard rules. The rewrite rules are as follows.

(17)	$check_1U_i \overset{i}{\hookrightarrow} check_1\varepsilon$	for all $i \in \{1, \ldots, n\}$
(18)	$check_1V_i \overset{i}{\hookrightarrow} check_1\varepsilon$	for all $i \in \{1, \ldots, n\}$
(19)	$check_2U_i \overset{u_i}{\hookrightarrow} check_2\varepsilon$	for all $i \in \{1, \ldots, n\}$
(20)	$check_2V_i \overset{v_i}{\hookrightarrow} check_2\varepsilon$	for all $i \in \{1, \ldots, n\}$

Finally, we add rules that make the system normed. The construction of rules (21) and (22) is also discussed in Remark 30 of [May05]. The rules of type (21) enables the $norm$ action changing the value of the store onto del. In any state, whenever the value of the store is set to del, the rules of type (22) can be used to make the state normed. Hence, Δ is normed in all of its states. The rules of type (23) adapt the construction to the concept of fcBPA. They make all states composed of the constraint del and a non-empty term weakly bisimilar.

(21)	$ttX \overset{norm}{\hookrightarrow} delX$	for all $X \in Const(\Delta)$
(22)	$delX \overset{\tau}{\hookrightarrow} del\varepsilon$	for all $X \in Const(\Delta)$
(23)	$delX \overset{x}{\hookrightarrow} delX$	for all $X \in Const(\Delta)$ and $x \in Act(\Delta)$

Hence, we have strengthen the Mayr's result [May05], Theorem 29 (also reformulated as Theorem 7.4 of this paper) as follows.

Theorem 7.8. *Weak bisimilarity is undecidable for normed fcBPA systems.*

7.5 Conclusion

First, we have shown that the weak bisimilarity problem remains undecidable for weakly extended versions of BPP (wBPP) and BPA (wBPA) process classes. We note that the result for wBPA is just our interpretation (in terms of weakly extended systems) of Mayr's proof showing that the problem is undecidable for PDA with two control states only ([May05], Theorem 29).

In terms of parallel systems, our technique used for wBPP is new. To mimic the computation of a Minsky 2-counter machine, one has to be able to maintain its state information – the label of a current instruction and the values of the counters $\mathtt{c}_1$ and $\mathtt{c}_2$. As a finite-state control unit of wBPP is weak, it cannot be used to store even a part of such often changing information. Hence, contrary to the constructions in more expressive systems (PN [Jan95b] and PPDA [Mol96]) we have made a term part to manage it. Further, in a test-and-decrement instruction, a process constant L_j, which represents a label of the instruction, has to be changed and one of the counters $\mathtt{c}_1, \mathtt{c}_2$ has to be decreased at the same time (assuming its value is positive). As two process constants cannot be rewritten by one wBPP rewrite rule, we introduce new process constants Y_1 and Y_2 to represent "inverse elements" to X_1 and X_2 respectively and we make a term $X_k^x \| Y_k^y$ to represent the counter c_k the value of which is $x - y$. We note that a weak finite state control unit serves for controlling the correct order of the (finitely many) successive stages in the progress of a bisimulation game.

Moreover, we have shown that our undecidability results hold even for more restricted classes fcBPA and fcBPP and remain valid also for the

normed versions of fcBPP and fcBPA. Hence, they hold for normed wBPP and normed wBPA as well.

We recall that the decidability of weak bisimilarity is an open question for BPA and BPP. Note that these problems are conjectured to be decidable (see [May05] and [Jan03] respectively) in which case our results would establish a fine undecidability border of weak bisimilarity (see Figure 7.5).

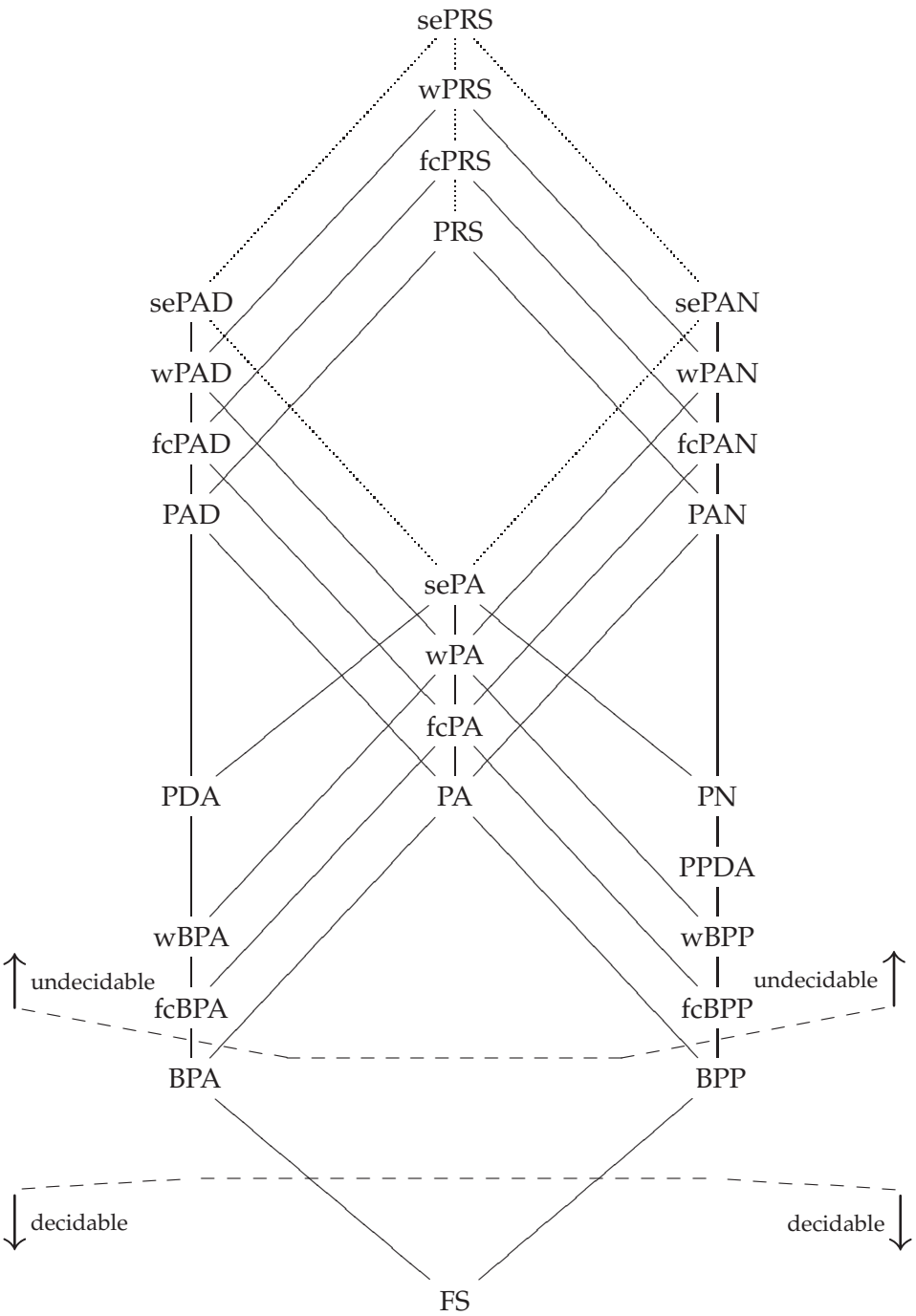


Figure 7.5: The extended PRS-hierarchy with (un)decidability boundaries of weak bisimilarity.

Chapter 8

Reachability Problem

In this chapter we show that the reachability problem remains decidable for the wPRS class and thus we determine the decidability borderline of the reachability problem in the extended PRS-hierarchy.

8.1 Motivation

A reachability problem (i.e. given two states α, β: is the state β reachable from the state α, written $\alpha \longrightarrow^* \beta$?) can be considered as a basis for all model checking problems. In the beginning of Chapter 3, we indicate how a finite-state control unit (FSU) is useful for modelling of systems. On the other hand, using an FSU to extend the PRS rewriting mechanism is very powerful since the reachability problem becomes undecidable for a *state extended* version of *PA processes* (sePA) [BEH95]. Concerning decidability results, Mayr [May00] has shown that the reachability problem for PRS is decidable.

In the context of reachability analysis one can see at least two approaches: (i) abstraction (approximate) analysis techniques on 'stronger' models such as sePA and its superclasses with undecidable reachability, e.g. see [BET03], and (ii) precise techniques for computing the set of states that are reachable from a given (regular) set of states, e.g. [LS98, EP00, BT03]. In the latter approach, the sets are represented symbolically and various term structural equivalences are considered. The papers dealing with this approach usually work with the classes PA or PAD rather than with general PRS systems.

In this chapter we study the reachability problem on the classes where the problem was open, i.e. the classes more expressive than (or incomparable with) the PRS class and less expressive than (or incomparable with) the sePA class. As the main contribution of this chapter we show that the reachability problem remains decidable for the wPRS class. This result determines the decidability borderline of the reachability problem in the ex-

tended PRS-hierarchy: the problem is decidable for all classes except the sePA class and its superclasses. Moreover, the result has several interesting applications. In this chapter we mention some of them, namely

- decidability of some safety properties over wPRS,
- semi-decidability of weak trace non-equivalence for wPRS, and
- decidability of the reachability problem for a replicative variant of Dolev and Yao's ping-pong protocols [HS05].

Some other applications can be found in the following chapters.

The outline of this chapter is as follows: In Section 8.2 we show that the reachability problem is decidable for weakly extended PRS. Section 8.3 is devoted to the aforementioned applications of our decidability result. The last section summarises our results.

8.2 Reachability Problem for wPRS

In this section we show that the *reachability problem* for wPRS is decidable. More precisely, we solve the following problem.

Problem:	*Reachability problem for an extended (α, β)-PRS class*
Instance:	An extended (α, β)-PRS system Δ and two of its states $mt, m't'$
Question:	Is the state $m't'$ reachable from mt in Δ, i.e. $mt \longrightarrow_{\Delta}^{*} m't'$?

Our proof exhibits a similar structure to the proof of decidability of the reachability problem for PRS [May00]. First we reduce the general reachability problem to the subproblem of reachability for wPRS with rules containing at most one occurrence of a sequential or parallel operator. Then second, we solve this subproblem using the fact that the reachability problems for both PN and PDA are decidable [May81, Büc64]. The latter part of our proof is based on our new idea of *passive steps* presented later.

To get just a sketch of the entire proof we suggest to read the definitions and statements (skipping their technical proofs). Several of them are preceded by comments that provide some intuition. As the labels on rewrite rules are not relevant in this section, we omit them.

Definition 8.1. *Let Δ be a wPRS. A rewrite rule in Δ is* parallel *or* sequential *if it has one of the following forms:*

parallel rules*:* $pX \hookrightarrow q(Y \| Z)$ $p(X \| Y) \hookrightarrow qZ$ $pX \hookrightarrow qY$ $pX \hookrightarrow q\varepsilon$,
sequential rules*:* $pX \hookrightarrow q(Y.Z)$ $p(X.Y) \hookrightarrow qZ$ $pX \hookrightarrow qY$ $pX \hookrightarrow q\varepsilon$,

where X, Y, Z are process constants and p, q are control states. A rule is trivial

if it is both parallel and sequential (i.e. it has the form $pX \hookrightarrow qY$ or $pX \hookrightarrow q\varepsilon$). A wPRS Δ is in normal form *if every rewrite rule in Δ is either parallel or sequential.*

Lemma 8.2. *Given a wPRS Δ with terms t_1 and t_2, we can effectively construct a wPRS Δ' in normal form over the same control states, along with terms t'_1 and t'_2, such that $rt_1 \longrightarrow^*_\Delta st_2$ if and only if $rt'_1 \longrightarrow^*_{\Delta'} st'_2$.*

Proof. In this proof we assume that the sequential composition is left-associative. It means that the term $X.Y.Z$ is considered as $(X.Y).Z$, hence its proper subterms are X, Y, Z, and $X.Y$, but not $Y.Z$. However, the term $Y \| Z$ is a subterm of $X.(Y \| Z)$.

Let $size(t)$ denote the number of sequential and parallel operators in a term t. We put $size(pt \hookrightarrow qt') = size(t) + size(t')$. Given any wPRS Δ, let k_i be the number of rules $(pt \hookrightarrow qt') \in \Delta$ that are neither parallel nor sequential and $size(pt \hookrightarrow qt') = i$. Thus, Δ is in normal form if and only if $k_i = 0$ for all i. In this case, let $n = 0$. Otherwise, let n be the largest i such that $k_i \neq 0$ (n exists as the set of rules is finite). We define $norm(\Delta)$ to be the pair (n, k_n).

We now describe a procedure transforming any given wPRS Δ which is not in normal form and terms t_1, t_2 into a wPRS Δ' and terms t'_1, t'_2 such that $rt_1 \longrightarrow^*_\Delta st_2 \iff rt'_1 \longrightarrow^*_{\Delta'} st'_2$ and $norm(\Delta') < norm(\Delta)$ with respect to the lexicographical ordering on norms as pairs of integers.

Let us assume that a wPRS Δ is not in normal form. Then there is a rule that is neither sequential nor parallel and has the maximal $size$. If the rule is a of the form $p(X_1.X_2) \hookrightarrow q(Y_1 \| Y_2)$ or $p(Y_1 \| Y_2) \hookrightarrow q(X_1.X_2)$, let t be $X_1.X_2$; otherwise, let t be a non-atomic and proper subterm of this rule. Now, replace every occurrence of the subterm t in rewrite rules of Δ and in the terms t_1, t_2 by a fresh constant X_t. Then add two rules $pX_t \hookrightarrow pt$ and $pt \hookrightarrow pX_t$ for each control state p. This yields a new wPRS Δ' and terms t'_1 and t'_2 where the constant X_t serves as an abbreviation for the term t. By the definition of $norm$ we get $norm(\Delta') < norm(\Delta)$. The correctness of our transformation remains to be demonstrated, namely that

$$rt_1 \longrightarrow^*_\Delta st_2 \iff rt'_1 \longrightarrow^*_{\Delta'} st'_2.$$

The implication $\Longleftarrow$ is obvious. For the opposite direction we show that every rewriting step in Δ from pl_1 to ql_2 under a rule $(pl \hookrightarrow ql') \in \Delta$ corresponds to a sequence of several rewriting steps in Δ' leading from pl'_1 to ql'_2, where l'_1, l'_2 are equal to l_1, l_2 with all occurrences of t replaced by X_t. Let us assume the rule $pl \hookrightarrow ql'$ modifies a subterm t of pl_1, and/or a subterm t appears in ql_2 after the rule application (the other cases are trivial). If the rule modifies a subterm t of l_1 then there are two cases.

1. Let l include the whole t. Then the corresponding rule in Δ' (with t replaced by X_t) can be applied directly on pl'_1.

2. Let l contain a part of t only. Due to the left-associativity of a sequential operator, t is not a subterm of the right part of any sequential composition in l_1. Thus we apply the added rule $pX_t \hookrightarrow pt$ on pl'_1 first and then we apply the rule in Δ' corresponding to the rule $pl \hookrightarrow ql'$.

The situation when t appears in ql_2 after the application of the considered rule is similar. Either l' includes the whole t and then the application of the corresponding rule in Δ' results directly in ql'_2, or t is not a subterm of the right part of any sequential composition in l_2 and thus the application of the corresponding rule in Δ' is followed by an application of the added rule $qt \hookrightarrow qX_t$ reaching the state ql'_2.

By repeating this procedure we finally get a wPRS Δ'' in normal form and terms t''_1,t''_2 satisfying $rt_1 \longrightarrow^*_\Delta st_2 \iff rt''_1 \longrightarrow^*_{\Delta''} st''_2$. □

Mayr's proof for PRS now transforms the PRS Δ in normal form into a PRS Δ' in so-called *transitive normal form* satisfying $(X \hookrightarrow Y) \in \Delta'$ whenever $X \longrightarrow^*_{\Delta'} Y$. This step employs the fact that rewriting under sequential rules in a parallel environment (or vice versa) has "local effect" only. Intuitively, whenever there is a rewriting sequence

$$X \| Y \longrightarrow^*_\Delta (X_1.X_2) \| Y \longrightarrow^*_\Delta (X_1.X_2) \| Z \longrightarrow^*_\Delta X_2 \| Z$$

in a PRS in normal form, then the rewriting of each parallel component is independent in the sense that there are also rewriting sequences

$$X \longrightarrow^*_\Delta X_1.X_2 \longrightarrow^*_\Delta X_2 \text{ and } Y \longrightarrow^*_\Delta Z.$$

This does not hold for wPRS in normal form as the rewriting in one parallel component can influence the rewriting in other parallel components via a weak control. To get this independence back we introduce the concept of *passive steps* emulating the changes of a control state produced by the environment.

Definition 8.3. *A finite sequence of control state pairs $PS = \{(p_i, q_i)\}_{i=1}^n$ satisfying $p_1 > q_1 \geq p_2 > q_2 \geq \cdots \geq p_n > q_n$ is called a sequence of* passive steps, *or just passive steps for short.*

Let Δ be a wPRS and PS be passive steps. By Δ^{PS} we denote the system Δ with an added rule $pX \hookrightarrow qX$ for each (p, q) in PS and $X \in Const(\Delta)$.

Further, we define Δ_{triv}, Δ_{seq}, and Δ_{par} to be the subset of trivial, sequential, and parallel rules of Δ, respectively.

Informally, $rt_1 \longrightarrow^*_{\Delta^{PS}} st_2$ means that the state rt_1 can be rewritten into the state st_2 provided a control state can be passively changed from p to q for every passive step (p, q) in PS. Please note that there is only a finite number of different sequences of passive steps for a given wPRS system.

Definition 8.4. *Let wPRS Δ be in normal form. If for every $X, Y \in Const(\Delta)$, control states r, s, and passive steps PS it holds that*

$rX \longrightarrow^*_{\Delta^{PS}} sY \implies rX \longrightarrow^*_{\Delta^{PS}_{triv}} sY$ *then Δ is in* flat normal form,

$rX \longrightarrow^*_{\Delta^{PS}_{seq}} sY \implies rX \longrightarrow^*_{\Delta^{PS}_{triv}} sY$ *then Δ is in* sequential flat normal form,

$rX \longrightarrow^*_{\Delta^{PS}_{par}} sY \implies rX \longrightarrow^*_{\Delta^{PS}_{triv}} sY$ *then Δ is in* parallel flat normal form.

The following lemma says that it is sufficient to check reachability via sequential rules and via parallel rules in order to construct a wPRS in flat normal form. This allows us to reduce the reachability problem for wPRS to the reachability problems for wPN and wPDA, i.e. to the reachability problems for PN and PDA.

Lemma 8.5. *If a wPRS is in both sequential and parallel flat normal form then it is in flat normal form as well.*

Proof. We assume the contrary and derive a contradiction. Let Δ be a wPRS in sequential and parallel flat normal form. Let us choose passive steps PS and a rewriting sequence $rX \longrightarrow^*_{\Delta^{PS}} sY$ such that $rX \not\longrightarrow^*_{\Delta^{PS}_{triv}} sY$ and the number of applications of non-trivial rewrite rules applied in the sequence is minimal. As the wPRS Δ is in both sequential and parallel flat normal form, $rX \not\longrightarrow^*_{\Delta^{PS}_{seq}} sY$ and $rX \not\longrightarrow^*_{\Delta^{PS}_{par}} sY$. Hence, both sequential and parallel operators occur in the rewriting sequence $rX \longrightarrow^*_{\Delta^{PS}} sY$. There are two following cases.

1. Assume that a sequential operator appears first. The parallel operator is then introduced by a rule of the form $pU \hookrightarrow q(T\|S)$ applied to a state $p(U.t)$, where t is a (possibly empty) sequential term. Note that $q((T\|S).t) \longrightarrow^*_{\Delta^{PS}} sY$ and recall the fact that at most one process constant can be removed in one rewriting step. Hence, first of all the term $T\|S$ is rewritten onto some single process constant V in the rest of the sequence considered. Let o be a control state after this rewriting. Using the same rewriting steps as in the original sequence, pU can be rewritten to oV in system Δ^{PS}. Let $PS'' = PS$.

2. Assume that a parallel operator appears first. The sequential operator is then introduced by a rule of the form $pU \hookrightarrow q(T.S)$ applied to a state $p(U\|t)$, where t is a (possibly empty) parallel term. The rest of the sequence subsumes steps rewriting the term $T.S$ onto some single process constant V. Let o be a control state in the state where $T.S$ is rewritten to V. Contrary to the previous case, the mentioned steps can be interleaved with steps rewriting the parallel component t and possibly changing a control state. Let PS' be a sequence of control state pairs corresponding to the changes of control states caused by rewriting of the parallel component t. We merge PS' with the subsequence of PS containing only the steps employed in the considered

rewriting sequence. As the result we get one sequence of passive steps denoted as PS''. Please note that the elimination of the unused steps of PS ensures that PS'' satisfies the definition of passive steps. Now, making use of the passive steps PS'' and the steps rewriting U to V in the original sequence, we construct a rewriting sequence in system $\Delta^{PS''}$ leading from pU to oV.

Thus we have obtained a rewriting sequence in $\Delta^{PS''}$ from pU to oV with fewer applications of non-trivial rewrite rules – we omit at least the first application of a non-trivial rewrite rule in the original sequence. Further, at least the first step of the new sequence is an application of a non-trivial rewrite rule. Moreover, as the number of applications of non-trivial rewrite rules used in the original sequence is minimal, we get $pU \not\longrightarrow^* \Delta^{PS''}_{triv} oV$. This contradicts our initial assumptions about the choices of PS and the rewriting sequence in Δ^{PS}. □

Example 8.6. *Here, we illustrate a possible change of passive steps (PS to PS'') described in the second case of the proof above. Let us consider a wPRS Δ with control states $r > p > q > t > v > o > s$ and the following rewrite rules*

$$\begin{array}{lll} rX \hookrightarrow p(U\|Z) & pU \hookrightarrow q(T.S) & v(T.S) \hookrightarrow oV \\ & qZ \hookrightarrow tY & o(V\|Y) \hookrightarrow sY \end{array}$$

as well as the following sequence in Δ^{PS} where $PS = \{(t, v)\}$

$$\begin{array}{llllll} & r\underline{X} & \longrightarrow_{\Delta^{PS}} & p(\underline{U}\|Z) & \longrightarrow_{\Delta^{PS}} & \\ \longrightarrow_{\Delta^{PS}} & q((T.S)\|\underline{Z}) & \longrightarrow_{\Delta^{PS}} & \underline{t}((T.S)\|Y) & \overset{passive}{\longrightarrow}_{\Delta^{PS}} & \\ \overset{passive}{\longrightarrow}_{\Delta^{PS}} & v((\underline{T.S})\|Y) & \longrightarrow_{\Delta^{PS}} & o(\underline{V\|Y}) & \longrightarrow_{\Delta^{PS}} & sY \end{array}$$

where redexes are underlined. The sequence of passive steps constructed due to the case 2 is $PS'' = \{(q, t), (t, v)\}$ and the constructed rewriting sequence is

$$p\underline{U} \longrightarrow_{\Delta^{PS''}} \underline{q}(T.S) \overset{passive}{\longrightarrow}_{\Delta^{PS''}} \underline{t}(T.S) \overset{passive}{\longrightarrow}_{\Delta^{PS''}} v(\underline{T.S}) \longrightarrow_{\Delta^{PS''}} oV.$$

The following lemma employs the algorithms deciding the reachability problem for PDA and PN. Recall that the classes PDA and PN coincide with the classes of wPDA and wPN, respectively.

Lemma 8.7. *For every wPRS Δ in normal form, terms t_1, t_2 over $Const(\Delta)$, and control states r, s of Δ, a wPRS Δ' can be constructed such that Δ' is in flat normal form and satisfies $rt_1 \longrightarrow^*_{\Delta} st_2 \iff rt_1 \longrightarrow^*_{\Delta'} st_2$.*

Proof. To obtain Δ' we enrich Δ by trivial rewrite rules transforming the system into sequential and parallel flat normal form, which suffices thanks to Lemma 8.5.

Using the algorithms deciding reachability for PDA and PN, our algorithm checks if there are some control states r, s, constants $X, Y \in Const(\Delta)$, and passive steps $PS = \{(p_i, q_i)\}_{i=1}^{n}$ (satisfying $r \geq p_1$ and $q_n \geq s$ as control states pairs out of this range are of no use here) such that $rX \not\longrightarrow^*_{\Delta^{PS}_{triv}} sY$, but $rX \longrightarrow^*_{\Delta^{PS}_{seq}} sY$ or $rX \longrightarrow^*_{\Delta^{PS}_{par}} sY$ hold. We finish if the answer is negative. Otherwise we add to Δ the rules $rX \hookrightarrow p_1 Z_1$, $q_i Z_i \hookrightarrow p_{i+1} Z_{i+1}$ for $i = 1, \ldots, n-1$, and $q_n Z_n \hookrightarrow sY$, where $Z_1, \ldots, Z_n$ are fresh process constants; if $n = 0$ then we add just the rule $rX \hookrightarrow sY$. Hence, $rX \longrightarrow^*_{\Delta''^{PS}_{triv}} sY$ where Δ'' is the system Δ with the new rules.

The algorithm then repeats this procedure on the system Δ'' with one difference: the X, Y range over the constants of the original system Δ. This is sufficient as the new constants occur only in trivial rules. Thus, if the system with added rules is not in sequential or parallel flat normal form, then there is a counterexample with the constants X, Y of the original system Δ. The algorithm eventually terminates as the number of iterations is bounded by the number of pairs of states rX, sY of Δ, times the number of sequences of passive steps PS. The correctness follows from the fact that the added rules only duplicate existing rewrite sequences between states of Δ. □

Theorem 8.8. *The reachability problem for wPRS is decidable.*

Proof. Let Δ be a wPRS with states rt_1, st_2. We want to decide whether $rt_1 \longrightarrow^*_\Delta st_2$ or not. We assume that $rt_1 \neq st_2$ (the other case is trivial).

Clearly $rt_1 \longrightarrow^*_\Delta st_2 \iff rX \longrightarrow^*_{\Delta''} sY$, where X, Y are fresh constants and Δ'' arises from Δ by the addition of the rules $rX \hookrightarrow rt_1$ and $st_2 \hookrightarrow sY$ (if $t_2 = \varepsilon$ then the latter rule is not a correct rule; in this case we add to Δ'' a rule $pt \hookrightarrow qY$ for each rule $(pt \hookrightarrow q\varepsilon) \in \Delta$ instead of this incorrect rule). Lemma 8.2 and Lemma 8.7 successively reduce the question whether $rX \longrightarrow^*_{\Delta''} sY$ to the question whether $rX \longrightarrow^*_{\Delta'} sY$, where Δ' is in flat normal form – note that the algorithm in the proof of Lemma 8.2 does not change terms t_1, t_2 if they are process constants. The definition of flat normal form implies $rX \longrightarrow^*_{\Delta'} sY \iff rX \longrightarrow^*_{\Delta'_{triv}} sY$. Finally the relation $rX \longrightarrow^*_{\Delta'_{triv}} sY$ is easy to check. □

8.3 Applications

In this section we discuss some of the applications of our decidability result presented in the previous section.

8.3.1 Model Checking Some Safety Properties

In the context of verification, one often formulates a property expressing that *some "bad" states are not reachable*. These properties are called *safety*

properties. If the number of bad states is finite, the problem can be directly solved using reachability problem; but it is usually not the case. Usually, the bad states are characterised as those satisfying some specific property, e.g. to be a deadlock state, an internal variable x is equal to zero, stack overflows, division by zero is performed, etc. In what follows, we solve model checking for wPRS and only such safety properties that express the bad states as those where a transition with a given label is enabled. In particular, we solve a problem whether, for given wPRS Δ and its action bad, there is a reachable state in which a transition with the label bad is enabled.

Lemma 8.9. *Given a wPRS Δ and $bad \in Act(\Delta)$, it is decidable whether there exists a reachable state mt such that $mt \xrightarrow{bad}_{\Delta} nt'$ for some state nt'.*

Proof. The proof is done by reduction to the reachability problem. Let $\Delta = (M, \leq, R, m_0, t_0)$. We construct a wPRS $\Delta' = (M', \leq', R', m_0, t_0)$, where $(M', \leq')$ is $(M, \leq)$ extended with a new control state r which is the least with respect to $\leq$ and where R' arises from R by adding the following rewrite rules:

(1) $mt_1 \overset{bad}{\hookrightarrow} rt_1$ for all $(mt_1 \overset{bad}{\hookrightarrow} nt_2) \in \Delta$,

(2) $rX \overset{bad}{\hookrightarrow} r\varepsilon$ for all $X \in Const(\Delta)$.

The rules of type (1) allow us to change any control state to r whenever a bad transition is enabled in the original system. Entering the control state r, a term can be rewritten to ε using the rules of type (2). Hence, a state mt such that $mt \xrightarrow{bad}_{\Delta} nt'$ for some state nt' is reachable in Δ if and only if the state $r\varepsilon$ is reachable in Δ'. □

Therefore, our decidability result can be seen as a contribution to an automatic verification of infinite-state systems as well. For model checking more complex safety properties we refer to Chapter 9.

8.3.2 Semi-decidability of Weak Trace Non-equivalence

Weak trace equivalence is a familiar notion which can be found already, for instance, in [Hoa80]. It is one of the semantic equivalences with a silent action τ. We refer to Chapter 7 for more information about silent actions and silent moves. Here we employ a straightforward definition of weak trace equivalence, see [JEM99].

Given a labelled transition system $(S, Act, \longrightarrow, \alpha_0)$ with a distinguished action $\tau \in Act$, we define the *weak trace set* of a state $s \in S$ as

$$wtr(s) = \{w \in (Act \setminus \{\tau\})^* \mid s \overset{w}{\Longrightarrow} t \text{ for some } t \in S\},$$

where $s \overset{w}{\Longrightarrow} t$ means that there is some $w' \in Act^*$ such that $s \xrightarrow{w'}{}^* t$ and w is equal to w' with its τ actions deleted. Two states of a system are said to

be *weak trace equivalent* if they have the same weak trace sets. It is already known that weak trace non-equivalence is semi-decidable for Petri nets (see e.g. [Jan95a]), pushdown processes (due to [Büc64]), and PA processes (due to [LS98]). Before we strengthen the result to wPRS and all its subclasses, we prove an auxiliary lemma stating that the weak trace sets are recursive.

Lemma 8.10. *Given a wPRS Δ, its state mt, and a word $w \in Act(\Delta)^*$, it is decidable whether $w \in wtr(mt)$ or not.*

Proof. We show that the problem can be reduced to the reachability problem. Let $\Delta = (M, \sqsubseteq, R, m_0, t_0)$ be a wPRS, mt be its state, and $w = w(0)w(1)w(2)\dots w(k) \in (Act \setminus \{\tau\})^+$ be a word (the case $w = \varepsilon$ is trivial as $mt \longrightarrow_\Delta^* mt$). We construct a wPRS $\Delta' = (M', \sqsubseteq', R', (m_0, 0), t_0)$, where

- $M' = \{e\} \cup M \times \{0, 1, \dots, k\}$,
- $\sqsubseteq'$ is defined as $e \sqsubseteq' e$ and $e \sqsubseteq' (m, i)$ for all $(m, i) \in M'$, and $(n, j) \sqsubseteq' (m, i)$ for all $(m, i), (n, j) \in M'$ satisfying $n \sqsubseteq m$ and $i \leq j$,
- R' consists of the following rules:

(1)	$(m,i)t_1 \overset{\tau}{\hookrightarrow} (n,i)t_2$	for all $0 \leq i \leq k$ and $(mt_1 \overset{\tau}{\hookrightarrow} nt_2) \in \Delta$,
(2)	$(m,i)t_1 \overset{w(i)}{\hookrightarrow} (n,i+1)t_2$	for all $0 \leq i < k$ and $(mt_1 \overset{w(i)}{\hookrightarrow} nt_2) \in \Delta$,
(3)	$(m,k)t_1 \overset{w(k)}{\hookrightarrow} e\varepsilon$	for all $(mt_1 \overset{w(k)}{\hookrightarrow} nt_2) \in \Delta$,
(4)	$eX \overset{\tau}{\hookrightarrow} e\varepsilon$	for all $X \in Const(\Delta)$.

Roughly speaking the second components of control states allow us to use the rewrite rules of type (1) labelled with τ while the rules of type (2) can be used only in the order given by w. According to rules (3), the transition corresponding to the last letter of w changes the control state to e. Rules (4) then allow us to rewrite the current term to ε. Hence, one can readily confirm that $w \in wtr(mt)$ with respect to Δ if and only if the state $e\varepsilon$ is reachable from the state $(m, 0)t$ in the system Δ'. □

Theorem 8.11. *Weak trace non-equivalence for wPRS is semi-decidable.*

Proof. Let mt_1 and nt_2 be states of a wPRS Δ. A semi-decidability algorithm goes through all words $w \in (Act(\Delta) \setminus \{\tau\})^*$ and tests whether $w \in wtr(mt_1) \setminus wtr(nt_2)$ or $w \in wtr(nt_2) \setminus wtr(mt_1)$. The membership of w in these sets is decidable due to the previous lemma. If the algorithm finds such a word, then two given states mt_1, nt_2 are weak trace non-equivalent. Moreover, if the states are weak trace non-equivalent, the algorithm will eventually find a witness w. Hence, the weak trace non-equivalence is semi-decidable. □

To sum up, the border of the semi-decidability is moved up to the class of wPRS in the hierarchy. We emphasise that the semi-decidability result is new for classes PAN, PAD, and PRS of the original PRS-hierarchy, too. As the reachability problem is undecidable for the other classes of the extended PRS-hierarchy (i.e. sePA and its superclasses), it is easy to see that the weak trace non-equivalence is not even semi-decidable for them.

8.3.3 Other Applications

The decidability of the reachability problem for wPRS has recently been applied in the area of cryptographic protocols. Hüttel and Srba [HS05] define a replicative variant of a calculus for Dolev and Yao's ping-pong protocols [DY83]. They show that the reachability problem for their calculus is decidable as it can be reduced to the reachability problem for wPRS. We note that this application does not employ the full power of our result, as all the systems produced by the reduction mentioned belong to wPAD class.

8.4 Conclusion

We have shown that an extension of the Process Rewrite System mechanism with a weak finite-state control unit (wPRS) keeps the reachability problem decidable. Some applications of this result have been discussed as well, namely those of model checking some safety properties and semi-decidability of weak trace non-equivalence for wPRS. We also noted that the decidability of the reachability problem for wPRS has already been used in [HS05] to show that the reachability problem for a replicative variant of a calculus for Dolev and Yao's ping-pong protocols is decidable.

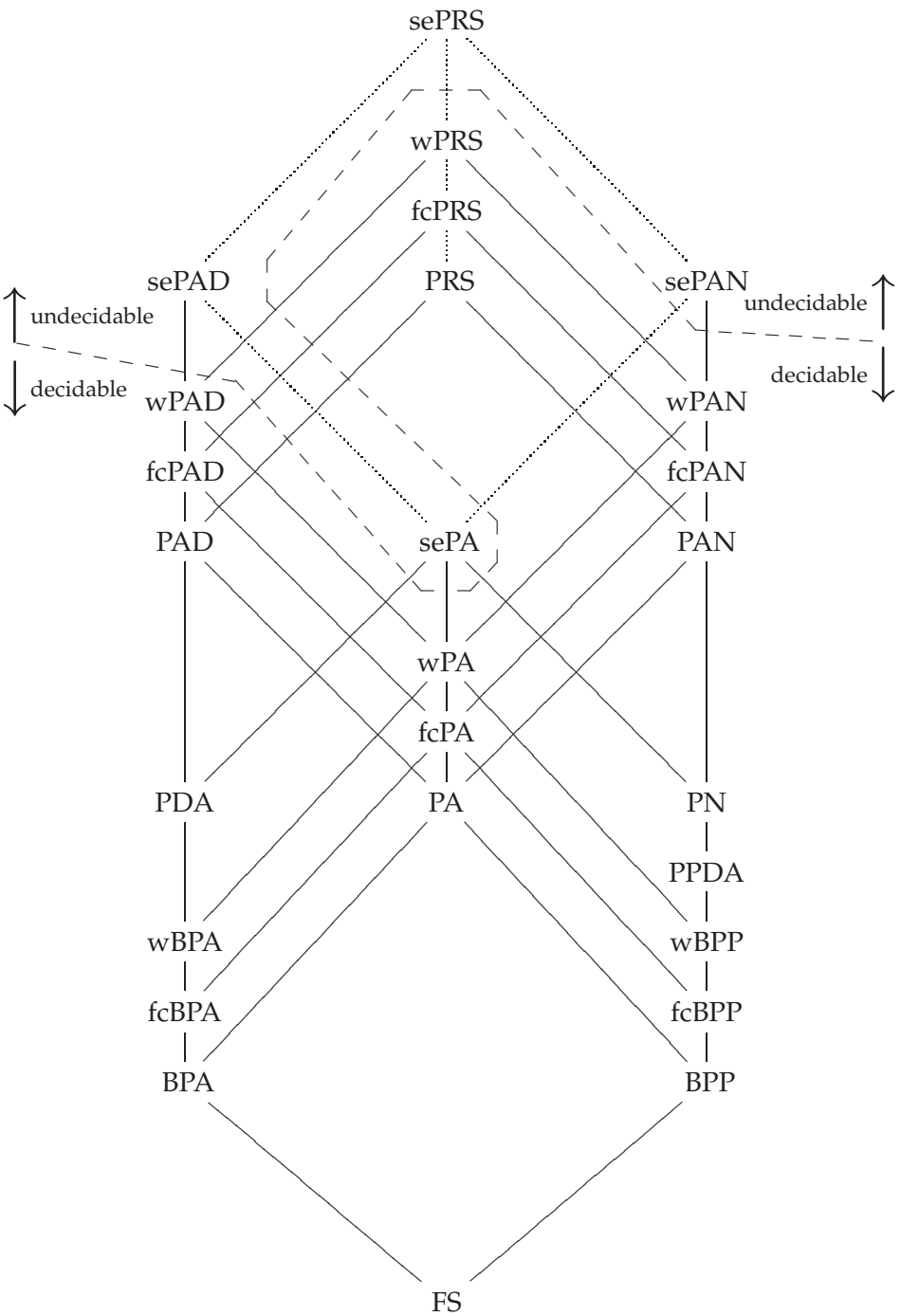


Figure 8.1: The extended PRS-hierarchy with (un)decidability boundaries of the reachability problem.

Chapter 9

Branching Time Logics

In this chapter we examine the problem whether a given weakly extended process rewrite system (wPRS) contains a reachable state satisfying a given formula of Hennessy–Milner logic. We show that this problem is decidable. As a corollary we observe that also the problem of strong bisimilarity between wPRS and finite-state systems is decidable. Decidability of the same problem for wPRS subclasses, namely PAN and PRS, has been formulated as an open question, see e.g. [Srb02a]. We also strengthen some related undecidability results on some PRS subclasses.

9.1 Motivation

Research on the expressive power of process classes has been accompanied by exploring algorithmic boundaries of various verification problems. In this chapter we focus on model checking some (fragments of) simple branching time logics, namely EF and EG.

Most of verification problems studied here are known as *reachability properties*. Such properties express that some some "bad" state(s), for example deadlock, should not be reached along any execution path or that some "good" state(s) should be reached along at least one execution path. The former can be naturally expressed as $\neg\mathsf{EF}\varphi$ where a formula φ characterises the set of states which should not be reached, while the latter can be stated as $\mathsf{EF}\varphi$ where φ characterises the set of states to be reached.

Now, we briefly recall the state of the art for EF and EG logics on PRS subclasses; details can be found in [BCMS01] and references given there. Also we mention our contribution using wPRS and sePRS subclasses.

First, we recall that the *reachability problem*, i.e. to decide whether a given state is reachable from the initial one, is decidable for the classes of PRS [May00] and wPRS (Theorem 8.8), while it is undecidable for sePA [BEH95]. See Chapter 8 for more information. All the problems mentioned below remain undecidable on the sePA class due to its Turing power.

A *reachability property problem*, for a given system Δ and a given formula φ, is to decide whether $\mathsf{EF}\varphi$ holds in the initial state of Δ. Hence, these problems are parametrised by the class to which the system Δ belongs, and by the type of the formula φ. In most of practical situations, φ specifies error states and the reachability property problem is a formalisation of a natural verification problem whether some error state is reachable in a given system.

We recall that the (full) EF logic is decidable for the PAD class, as shown in [May98]. It is undecidable for PN [Esp94]; an inspection of the proof moves this undecidability border down to PPDA. If we consider the *reachability HM property problem*, i.e. the reachability property problem where φ is a formula of Hennessy–Milner logic (HM formula), then this problem has been shown to be decidable for the classes of PN [JM95] and PAD [JKM01]. In this chapter we present our proof lifting the decidability border for this problem to the wPRS class (published in [KŘS05]). This results also moves the decidability border for the *reachability simple property problem*, i.e. the reachability property problem where φ is a HM formula without any nesting of modal operators $\langle a \rangle$ (the problem has been know to be decidable for PRS [May00] so far). Recently, EF logic was shown to be decidable also for the wPAD class (see [BST06]).

Let us recall that the (full) EG logic is decidable for the PDA class (a consequence of [MS85] and [Cau92]), whilst undecidability has been obtained for its $\mathsf{EG}\varphi$ fragment on (deterministic) BPP [EK95], where φ is a HM formula. We show that this problem remains undecidable on (deterministic) BPP even if we restrict φ to a HM formula without nesting of modal operators $\langle a \rangle$.

As a corollary of our main result of this chapter, i.e. decidability of the reachability HM property problem for wPRS, we observe that the problem of strong bisimilarity between wPRS systems and finite-state ones is decidable. As PRS and its subclasses are proper subclasses of wPRS, it follows that we positively answer the question of the reachability HM property problem for the PRS class and hence the questions of bisimilarity checking the PAN and PRS processes with finite-state ones, which have been open problems, see for example [Srb04]. Their relevance to program specification and verification is advocated, for example, in [JKM01, KS04].

9.2 Logics and Studied Problems

In this chapter we work with fragments of *unified system of branching-time logic* (UB) [BAPM83].

Definition 9.1. *Let $Act = \{a, b, \cdots\}$ be a countably infinite set of atomic actions.*

UB formulae have the following syntax:

$$\varphi \ ::= \ tt \ \mid \ \neg\varphi \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \langle a \rangle \varphi \ \mid \ \mathsf{EF}\varphi \ \mid \ \mathsf{EG}\varphi,$$

where $a \in Act$ is an action.

We define the semantics of UB formulae over states of a labelled transition system.

Definition 9.2. *Let φ be a UB formula, $\mathcal{L} = (S, \longrightarrow, \alpha_0)$ be a labelled transition system and α be a state of $\mathcal{L}$. The validity of the formula φ in the state α of the labelled transition system $\mathcal{L}$, written $(\alpha, \mathcal{L}) \models \varphi$, is defined by induction on the structure of φ:*

$(\alpha, \mathcal{L}) \models tt$

$(\alpha, \mathcal{L}) \models \neg\varphi$ *iff* $(\alpha, \mathcal{L}) \not\models \varphi$

$(\alpha, \mathcal{L}) \models \varphi_1 \wedge \varphi_2$ *iff* $(\alpha, \mathcal{L}) \models \varphi_1 \ \wedge \ (\alpha, \mathcal{L}) \models \varphi_2$

$(\alpha, \mathcal{L}) \models \langle a \rangle \varphi$ *iff* $\exists \alpha'.\ \alpha \xrightarrow{a}_{\mathcal{L}} \alpha' \ \wedge \ (\alpha', \mathcal{L}) \models \varphi$

$(\alpha, \mathcal{L}) \models \mathsf{EF}\varphi$ *iff* $\exists \alpha'.\ \alpha \longrightarrow^*_{\mathcal{L}} \alpha' \ \wedge \ (\alpha', \mathcal{L}) \models \varphi$

$(\alpha, \mathcal{L}) \models \mathsf{EG}\varphi$ *iff* *there is a maximal (finite or infinite) transition sequence $\alpha_1 \xrightarrow{a_1}_{\mathcal{L}} \alpha_2 \xrightarrow{a_2}_{\mathcal{L}} \alpha_3 \xrightarrow{a_3}_{\mathcal{L}} \ldots$ such that $\alpha = \alpha_1$ and all states in the sequence satisfy $(\alpha_i, \mathcal{L}) \models \varphi$, for all $i \geq 1$*

We write $\mathcal{L} \models \varphi$ if φ is valid in the initial state α_0 of $\mathcal{L}$.

Definition 9.3. *For each UB formula φ, we define $depth(\varphi)$ as a nesting depth of $\langle a \rangle$ operators in φ by induction on the structure of the formula:*

$$\begin{aligned}
&depth(tt) = 0 \\
&depth(\neg\varphi) = depth(\varphi) \\
&depth(\varphi \wedge \psi) = \max\{depth(\varphi), depth(\psi)\} \\
&depth(\langle a \rangle \varphi) = depth(\varphi) + 1 \\
&depth(\mathsf{EF}\varphi) = depth(\varphi) \\
&depth(\mathsf{EG}\varphi) = depth(\varphi)
\end{aligned}$$

Introducing some restriction on the syntax level, we define four fragments of UB logic.

- A UB formula φ is called an *EF formula* if it does not contain any EG operator. Hence, the syntax of *EF logic* is defined as follows.

$$\varphi \ ::= \ tt \ \mid \ \neg\varphi \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \langle a \rangle \varphi \ \mid \ \mathsf{EF}\varphi$$

- A UB formula φ is called an *EG formula* if it does not contain any EF operator. Hence, the syntax of *EG logic* is defined as follows.

$$\varphi \ ::= \ tt \ \mid \ \neg\varphi \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \langle a \rangle \varphi \ \mid \ \mathsf{EG}\varphi$$

- A UB formula φ is called a *Hennessy–Milner formula* (or *HM formula* for short) if it contains neither EG nor EF operators. Hence, the syntax of *HM logic* is defined as follows.

$$\varphi \ ::= \ tt \ \mid \ \neg\varphi \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \langle a \rangle \varphi$$

- A UB formula φ is called a *simple formula* if it is an HM formula satisfying $depth(\varphi) = 1$. Hence, the syntax of *simple logic* is defined as

$$\varphi \ ::= \ tt \ \mid \ \neg\varphi \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \langle a \rangle \varphi'$$

  where $$\varphi' \ ::= \ tt \ \mid \ \neg\varphi' \ \mid \ \varphi'_1 \wedge \varphi'_2.$$

In the following, we deal with six problems parametrised by a subclass of sePRS systems.

EF properties:

Problem:	*Decidability of EF logic for $\mathcal{C}$*
Instance:	An EF formula φ and a system $\Delta \in \mathcal{C}$
Question:	Is the system Δ a model of the formula φ, i.e. $L(\Delta) \models \varphi$?

Problem:	*Reachability HM property for $\mathcal{C}$*
Instance:	An HM formula φ and a system $\Delta \in \mathcal{C}$
Question:	Is the system Δ a model of a formula $\mathsf{EF}\varphi$, i.e. $L(\Delta) \models \mathsf{EF}\varphi$?

Problem:	*Reachability simple property for $\mathcal{C}$*
Instance:	A simple formula φ and a system $\Delta \in \mathcal{C}$
Question:	Is the system Δ a model of a formula $\mathsf{EF}\varphi$, i.e. $L(\Delta) \models \mathsf{EF}\varphi$?

EG properties:

Problem:	*Decidability of EG logic for $\mathcal{C}$*
Instance:	An EG formula φ and a system $\Delta \in \mathcal{C}$
Question:	Is the system Δ a model of the formula φ, i.e. $L(\Delta) \models \varphi$?

Problem:	*Evitability HM property for $\mathcal{C}$*
Instance:	An HM formula φ and a system $\Delta \in \mathcal{C}$
Question:	Is the system Δ a model of a formula $\mathsf{EG}\varphi$, i.e. $L(\Delta) \models \mathsf{EG}\varphi$?

Problem:	*Evitability simple property for $\mathcal{C}$*
Instance:	A simple formula φ and a system $\Delta \in \mathcal{C}$
Question:	Is the system Δ a model of a formula $\mathsf{EG}\varphi$, i.e. $L(\Delta) \models \mathsf{EG}\varphi$?

It is easy to see that the reachability simple property problem is a subproblem of the reachability HM property problem, that is, again, a subproblem of the decidability of EF logic. The same holds for the three EG properties given above. Other combinations are incomparable. The situation can be depicted as follows.

$$\begin{array}{ccccc} \text{decidability} \\ \text{EF logic} \end{array} \Longrightarrow \begin{array}{c} \text{reachability} \\ \text{HM property} \end{array} \Longrightarrow \begin{array}{c} \text{reachability} \\ \text{simple property} \end{array}$$

$$\begin{array}{ccccc} \text{decidability} \\ \text{EG logic} \end{array} \Longrightarrow \begin{array}{c} \text{evitability} \\ \text{HM property} \end{array} \Longrightarrow \begin{array}{c} \text{evitability} \\ \text{simple property} \end{array}$$

9.3 Reachability HM Property

In this section, we study a *reachability HM property problem* for wPRS, i.e. the problem to decide whether a given wPRS Δ and a given HM formula φ satisfy $\Delta \models \mathsf{EF}\varphi$ or not. We prove that the problem is decidable. The proof reduces this problem to the *reachability problem* for wPRS, i.e. the problem to decide whether a given state of a given wPRS is reachable or not, which is decidable due to Theorem 8.8.

For the rest of this section, let Δ be a fixed wPRS system, $D \notin \mathit{Const}(\Delta)$ be a fixed fresh process constant, and $\mathcal{C} = \mathit{Const}(\Delta) \cup \{D\}$. Further, let φ be a HM formula and $n = \mathit{depth}(\varphi)$. We assume that $n > 0$.

Definition 9.4. *A term t' is called* n-equivalent to *a state pt of Δ if and only if, for each HM formula ψ satisfying $\mathit{depth}(\psi) \leq n$, it holds:*

$$(\Delta, pt) \models \psi \iff (\Delta, pt') \models \psi$$

Our proof will proceed in two steps. In the first step we show that there exists a *finite* set T of terms such that, for each reachable state pt of Δ, the set T contains a term t' which is n-equivalent to pt. In the second step we enrich the system with rules allowing us to rewrite an arbitrary reachable state pt to a state $[p, t']\, D$, where the control state $[p, t']$ represents the original control state p and a term t' which is n-equivalent to pt. Finally, for each $p \in M(\Delta), t' \in T$ satisfying $(\Delta, pt') \models \varphi$ we add a rule $[p, t']\, D \overset{a}{\hookrightarrow} \mathit{acc}\, D$. Let us note that the validity of $(\Delta, pt') \models \varphi$ is decidable as wPRS systems are finitely branching. To sum up, φ is valid for some reachable state pt of Δ if and only if the state $\mathit{acc}\, D$ is reachable in the modified system.

First, we introduce some auxiliary terminology and notation. A nonempty proper subterm t' of a term t is called *idle* if t' is the right-hand-side component of some sequential composition in t (such that its left-hand-side component is nonempty), where sequential composition is considered to be left-associative. For example, a term $(X.Y.Z)\|(U.(V\|W))$ should be interpreted as $((X.Y).Z)\|(U.(V\|W))$ and its idle subterms are $Y, Z, V\|W$ but not $Y.Z$. By $IdleTerms$ we denote a set of all idle terms occurring in the initial term or in terms on the right-hand sides of rewrite rules of Δ. Observe that each idle subterm of any reachable state of Δ is contained in $IdleTerms$.

We define a *length* of a term t, written $|t|$, as the number of all occurrences of process constants in the term. For example, $|X\|(X.Y)\|\varepsilon| = 3$. Formally, *length* of a term is defined by induction on the term structure as follows.

$$\begin{gathered}|\varepsilon| = 0\\ |X| = 1\\ |t_1.t_2| = |t_1| + |t_2|\\ |t_1\|t_2| = |t_1| + |t_2|\end{gathered}$$

Further, for each $j \geq 0$, we define a set

$$SmallTerms(j) = \{t \mid t \text{ is a term over } \mathcal{C} \text{ and } 0 < |t| \leq j\}.$$

Definition 9.5. *Let $h > 0$ be an integer. We put $k = \max\{|t| \mid t \in IdleTerms\}$ and $H = h \cdot (h + k) \cdot |SmallTerms(h + k)|$. We define $Rules(h)$ to be the set of rewrite rules of three types (see the proof of Lemma 9.6 for their respective roles):*

(1)	$p\,(s'.D) \overset{del}{\hookrightarrow} pD$	*for all*	$p \in M(\Delta)$ *and* $s' \in SmallTerms(H)$,
(2)	$p\,s^{h+1} \overset{del}{\hookrightarrow} p\,s^h$	*for all*	$p \in M(\Delta)$ *and* $s \in SmallTerms(H)$,
(3)	$p\,(s'.s) \overset{del}{\hookrightarrow} pD$	*for all*	$p \in M(\Delta)$, $s \in IdleTerms$, *and* $s' \in SmallTerms(H) \setminus SmallTerms(h)$,

where s^i denotes a parallel composition of i copies of term s.

Lemma 9.6. *For each $h > 0$ and for each reachable state pt of Δ it holds that $p\,(t.D) \longrightarrow^*_{Rules(h)} pD$.*

Proof. As every rule in $Rules(h)$ has its right-hand side shorter than its left-hand side and an application of a rule in $Rules(h)$ cannot produce any new idle subterm, it is sufficient to prove that, for each $p \in M(\Delta)$ and each term t over $\mathcal{C}$ with all idle subterms in $IdleTerms$, there is a rule of $Rules(h)$ applicable to $p\,(t.D)$. We assume the contrary and derive a contradiction. Let $p \in M(\Delta)$ be a control state and t be a term of the minimal length such that t satisfies the preconditions and no rule of $Rules(h)$ is applicable to $p\,(t.D)$. Then $|t| > H$ as in the other case a rule of type (1) is applicable to $p\,(t.D)$. There are two cases:

$t = u.v$ As $v \in IdleTerms$ we have $|v| \leq k$. Further, $|t| > H$ implies $|u| > H - k > h$. If $h < |u| \leq H$, then there is a rule of type (3) that can be applied to $p\,(t.D)$. Hence $|u| > H$. As no rule of $Rules(h)$ can be applied to $p\,(t.D) = p\,(u.v.D)$, no such rule can be applied to pu. The inequality $|u| > H$ gives us that a rule of type (1) is applicable to $p\,(u.D)$ if and only if it is applicable to pu. The same holds for rules of type (2) and (3) as well due to the shape of these rules and due to the fact that D does not occur in any term of $IdleTerms$. To sum up, no rule of $Rules(h)$ can be applied to $p\,(u.D)$ and thus u contradicts the minimality of t.

$t = u \| v$ As '$\|$' is associative and commutative, it can be seen as an operator with an unbounded arity. Thus, t can be seen as a parallel composition of several components which are nonempty sequential terms. The length of each of these components is less than or equal to H; a component u satisfying $|u| > H$ would contradict the minimality of t using the same arguments as in the previous case. Further, as no rule of type (3) can be applied, the length of each component is at most $h + k$. Moreover, as rules of type (2) are not applicable, we have that the parallel composition contains at most h copies of each component. Hence, $|t| \leq h \cdot (h + k) \cdot |SmallTerms(h + k)| = H$. This contradicts the relation $|t| > H$.

□

Definition 9.7. *Let l be the maximal length of a left-hand-side term of a rule in Δ. Lemma 9.6 implies that, for each reachable state pt of Δ, there exists a transition sequence $p\,(t.D) \longrightarrow^*_{Rules(nl)} pD$. By $MultiSet_{nl}(pt)$ (or just $MultiSet(pt)$ if no confusion can arise) we denote a multiset containing exactly all the subterms that are rewritten during this transition sequence and correspond to a subterm s' of rewrite rules of types (1) and (3). Further, for each multiset of terms $S = \{t_1, t_2, \ldots, t_j\}$, we define its* characteristic term *$t_S = (t_1.D) \| (t_2.D) \| \ldots \| (t_j.D)$.*

Lemma 9.8. *Let pt be a reachable state of Δ. Then $t_{MultiSet(pt)}$ is n-equivalent to pt.*

Proof. Let us fix a transition sequence $p\,(t.D) \longrightarrow^*_{Rules(nl)} pD$ and the corresponding multiset $MultiSet(pt)$. The proof proceeds by induction on the number of transition steps in the transition sequence.

Let S_i denote a part of $MultiSet(pt)$ obtained in the first i transition steps and pu_i be the state reached by these steps. It is sufficient to prove that, for each i, $u_i \| t_{S_i}$ is n-equivalent to pt. We note that D cannot be rewritten by any rewrite rule of Δ – it is used to prevent unwanted rewriting.

The basic step is trivial as $u_0 = t$, $S_0 = \emptyset$, and thus $u_i \| t_{S_i} = t \| \varepsilon$. Now we assume that $u_i \| t_{S_i}$ is n-equivalent to pt and we prove that the same holds

for $u_{i+1}\|t_{S_{i+1}}$. Let l be the maximal length of a left-hand-side term of a rule in Δ. There are three cases reflecting the type of the rewrite rule applied in a transition step $pu_i \xrightarrow{del}_{Rules(nl)} pu_{i+1}$:

type (1) We note that no rule in $Rules(nl)$ can introduce D on the right-hand side of a sequential composition. Thus, a rule $p\,(s'.D) \overset{del}{\hookrightarrow} pD$ of type (1) is applicable to pu_i iff $u_i = s'.D$. Therefore, $u_{i+1} = D$, $S_{i+1} = S_i \cup \{s'\}$, and $u_{i+1}\|t_{S_{i+1}} = D\|(s'.D)\|t_{S_i} = D\|u_i\|t_{S_i}$. As $u_i\|t_{S_i}$ is n-equivalent to pt, it is obvious that so is $u_{i+1}\|t_{S_{i+1}}$.

type (2) Let ψ be an HM formula such that $depth(\psi) \leq n$. Then its validity in a state depends only on the first n successive transitions performable from the state. At most nl process constants of the term t can be rewritten during n successive steps. Hence, at most nl parallel components can be rewritten during these steps. Thus, reducing of the number of identical parallel components from $nl + 1$ to nl does not affect the validity of ψ. To sum up, $u_{i+1}\|t_{S_{i+1}} = u_{i+1}\|t_{S_i}$ is n-equivalent to pt.

type (3) The term s' occurring in the applied rule satisfies $|s'| > nl$. Hence, the part of the term t corresponding to the subterm s of the rule is "too far" to be rewritten in the first n steps of any transition sequence. The term $s'.s$ in u_i is replaced by D in u_{i+1}. It is easy to see that $u_{i+1}\|t_{S_{i+1}} = u_{i+1}\|(s'.D)\|t_{S_i}$ is n-equivalent to pt.

□

Given a multiset of terms S, by $S{\downarrow}n$ we denote the largest subset of S containing at most n copies of each element. One can readily confirm that a characteristic term t_S is n-equivalent to some state of Δ if and only if $t_{S\downarrow n}$ is n-equivalent to this state.

To sum up, for each reachable state pt of Δ, we can construct a multiset $MultiSet(pt){\downarrow}n$ such that its characteristic term $t_{MultiSet(pt)\downarrow n}$ is n-equivalent to pt. Moreover, there is a bound on the size of each such a multiset which depends on Δ and n only. More precisely, such a multiset contains at most n copies of terms $s' \in SmallTerms(nl\cdot(nl+k)\cdot|SmallTerms(nl+k)|)$, where l is the maximal length of a left-hand-side term of a rule in Δ and k is the maximal length of a term in $IdleTerms$. We now present the reduction of the reachability HM property problem for wPRS to the reachability problem for wPRS.

Lemma 9.9. *Let Δ be a wPRS system and φ be a Hennessy–Milner formula. Then we can construct a wPRS Δ' with a state $acc\ D$ such that*

$$\Delta \models \mathsf{EF}\varphi \iff \textit{acc } D \textit{ is reachable in } \Delta'.$$

Proof. Let n, D, $\mathcal{C}$, $IdleTerms$, $SmallTerms(j)$, and $MultiSet(pt)$ have the same meanings as above.

Let k be the maximal length of a term in $IdleTerms$, l be the maximal length of a left-hand-side term in any rule from Δ, and $H = nl \cdot (nl + k) \cdot |SmallTerms(nl + k)|$. Further, let $\mathcal{S}$ be a set of all multisets containing at most n copies of each term $s' \in SmallTerms(H)$.

The system Δ' uses control states of the original system, a distinguished control state $acc \notin M(\Delta)$, and control states of the form (p, S) where $p \in M(\Delta)$ and $S \in \mathcal{S}$.

Let $p_0 t_0$ be the initial state of Δ. Then Δ' has the initial state $p_0\,(t_0.D)$ and the following rules, where p and S range over $M(\Delta)$ and $\mathcal{S}$ respectively. We omit labels as they are not relevant.

(1)	$pt \hookrightarrow qt'$	for all $(pt \overset{a}{\hookrightarrow} qt') \in \Delta$
(2)	$pX \hookrightarrow (p, \emptyset)\, X$	for all $X \in \mathcal{C}$
(3)	$(p, S)\, s'.D \hookrightarrow (p, (S \cup \{s'\})\downarrow n)\, D$	for all $s' \in SmallTerms(H)$
(4)	$(p, S)\, s^{nl+1} \hookrightarrow (p, S)\, s^{nl}$	for all $s \in SmallTerms(H)$
(5)	$(p, S)\, s'.s \hookrightarrow (p, (S \cup \{s'\})\downarrow n)\, D$	
	for all $s \in IdleTerms$ and $s' \in SmallTerms(H) \setminus SmallTerms(nl)$	
(6)	$(p, S)\, D \hookrightarrow acc\, D$	whenever $(\Delta, pt_S) \models \varphi$

Intuitively, the rules of type (1) mimic the behaviour of Δ and enable Δ' to reach a state $p\,(t.D)$ if and only if pt is a reachable state of Δ. A rule of type (2) stops this mimic phase and starts a checking phase where only rules of types (3)–(6) are applicable. The rules of types (3), (4), and (5) correspond to the rules of type (1), (2), and (3) in $Rules(nl)$, respectively. Let $p\,(t.D)$ be a final state reached in the mimic phase. The rules of types (3)–(5) allow us to rewrite this state to the state $(p, MultiSet(pt)\downarrow n)\, D$. Finally, the control state $(p, MultiSet(pt)\downarrow n)$ can be changed to acc using a rule of type (6) if and only if $(\Delta, t_{MultiSet(pt)\downarrow n}) \models \varphi$. As $t_{MultiSet(pt)\downarrow n}$ is n-equivalent to pt, the control state can be changed to acc if and only if $(\Delta, pt) \models \varphi$. □

The following theorem is an immediate corollary of Lemma 9.9 and Theorem 8.8.

Theorem 9.10. *The reachability HM property problem is decidable for wPRS.*

9.4 Corollaries and Remarks

An interesting corollary of Theorem 9.10 arises in connection with one of the results of [JKM01].

Theorem 9.11 ([JKM01], Theorem 22)**.** *If the model checking problem for simple EF formulae (i.e. reachability HM property problem) is decidable in a class K of transition systems, then strong bisimilarity is decidable between processes of K and finite-state ones.*

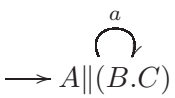
A combination of Theorem 9.10 and Theorem 9.11 yields the following corollary.

Theorem 9.12. *Strong bisimilarity is decidable between wPRS systems and finite-state ones.*

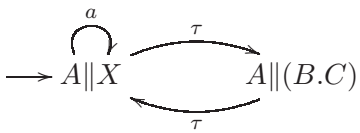
Let us mention few following remarks to figure out some corollaries and related results.

Remark 9.13. *Theorem 9.10 also implies that reachability simple property problem is decidable for PRS. This result has been previously presented in [May98] under the name* reachable property *problem. However, the proof given there contains a nontrivial mistake which was not fixed in subsequent papers [MR98, May00]. The weak point is the proof showing a transformation of an arbitrary PRS onto a PRS in normal form.*

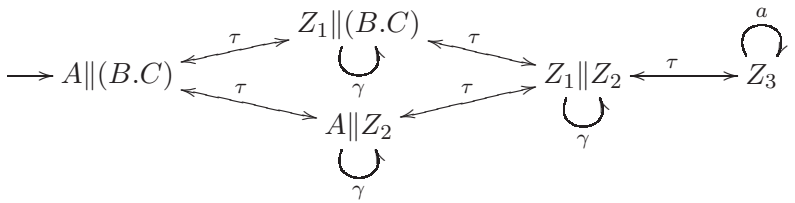
Considering a PRS $\Delta = (\{A\|(B.C) \stackrel{a}{\hookrightarrow} A\|(B.C)\}, A\|(B.C))$



that does not model a formula $\mathsf{EF}(\neg\langle a\rangle tt)$*, one receives a transformed PRS* Δ' *in normal form that models this formula. According to the construction of [May00] the LTS of* Δ' *is as follows.*



The construction of [May98] results in the following system where contrary to the lemma $\Delta' \models \mathsf{EF}(\neg\langle a\rangle tt \wedge \neg\langle\gamma\rangle tt)$*.*



Intuitively, the construction enriches a given PRS with and additional rules enabling folding and unfolding of large terms. The construction of [May98] also introduces γ *rules to sign intermediate states during folding and unfolding. The problem is that neither the* a *action nor the* γ *action is enabled in the state with the unfolded term. We do not see any easy way how to repair the construction. For example, labelling the folding rules by* γ *does not help because it cause other problems (e.g. folding part of one rewriting action can disturb performing of other action).*

Remark 9.14. *It is known that (full) EF logic is undecidable for PN [Esp94]. An inspection of the proof given in [Esp97] shows that this undecidability result is valid even for seBPP class.*

9.5 Evitability Simple Property for Deterministic BPP

Esparza and Kiehn have proved that EG logic is undecidable for (deterministic) BPP [EK95]. In this section we describe a modification of their proof showing that for (deterministic) BPP even the evitability simple property problem is undecidable. As we just describe the necessary changes to be done within the proof given in [EK95], we use the same notation as introduced in their paper.

The original proof is done by a reduction from the halting problem of a Minsky counter machine. A quick inspection of the reduction shows that it demonstrates undecidability of the *inevitability HM property problem* for the class of deterministic BPP systems. We note that it is not a proof of undecidability for the *inevitability simple property problem* due to the following reason. In the definition of $\widehat{EN}(a_1, \ldots, a_k)$, there is a subformula

$$\bigwedge_{i:=1}^{k} \neg\exists(a_i)EN(a_i) \text{ corresponding to } \bigwedge_{i=1}^{k} \neg\langle a_i\rangle\langle a_i\rangle tt \text{ in our notation}$$

which expresses that no sequence $a_i a_i$ is enabled. Omitting this subformula from $\widehat{EN}(a_1, \ldots, a_k)$, the construction produces a simple property formula.

In what follows, we present some other changes to be done within the construction in order to keep its correctness for the case of the simple property formula as well. In other words, we prove that even the inevitability *simple* property problem remains undecidable for the deterministic BPP systems.

The following definitions of $\mathtt{SM}$, $\mathtt{M}$, and $\mathtt{C_j}$ are the same as in [EK95]:

$$\mathtt{SM} \stackrel{\mathtt{def}}{=} (\mathtt{SQ_1} \| \ldots \| \mathtt{SQ_{n+1}}) \qquad \mathtt{M} \stackrel{\mathtt{def}}{=} \mathtt{SM} \| \mathtt{Q_0} \qquad \mathtt{C_j} \stackrel{\mathtt{def}}{=} \mathtt{dec}^1_\mathtt{j} \cdot \mathtt{dec}^2_\mathtt{j} \cdot \mathtt{dec}^3_\mathtt{j} \cdot \mathbf{0}$$

Without loss of generality, we assume that there is no self loop in the counter machine $\mathcal{M}$ (i.e. $k \neq i \neq k'$, for each transition rule of $\mathcal{M}$). Hence, it is not necessary to create a new parallel instance of a process constant $\mathtt{Q_i}$ from $\mathtt{SQ_i}$ as far as the rewriting on the existing instance of $\mathtt{Q_i}$ is not finished. In the following, we reformulate the definitions of $\mathtt{SQ_i}$ and $\mathtt{Q_i}$ to prevent sequences of the form $\mathtt{out}^1_\mathtt{i}\mathtt{out}^1_\mathtt{i}$ or $\mathtt{out}^2_\mathtt{i}\mathtt{out}^2_\mathtt{i}$.

The halting state definition is reformulated as follows.

$$\mathtt{SQ_{n+1}} \stackrel{\mathtt{def}}{=} \mathtt{in}^1_{\mathtt{n+1}} \cdot \mathtt{Q_{n+1}} \qquad\qquad \mathtt{Q_{n+1}} \stackrel{\mathtt{def}}{=} \mathtt{halt} \cdot \mathtt{SQ_{n+1}}$$

A state q_i of type II is modelled as follows.

$$\mathtt{SQ_i} \stackrel{\mathtt{def}}{=} \mathtt{in_i^1} \cdot \mathtt{Q_i} \qquad\qquad \mathtt{Q_i} \stackrel{\mathtt{def}}{=} \mathtt{out_i^1} \cdot \mathtt{out_i^2} \cdot \mathtt{SQ_i}$$

A state q_i of type I has to proceed to the state q_k. To prevent multiple occurrences of the process constant $\mathtt{Q_k}$, we use the same technique as in the case of states of type II. Hence, $\mathtt{SQ_i}$ and $\mathtt{Q_i}$ are modelled as

$$\mathtt{SQ_i} \stackrel{\mathtt{def}}{=} \mathtt{in_i^1} \cdot \mathtt{Q_i} \qquad\qquad \mathtt{Q_i} \stackrel{\mathtt{def}}{=} \mathtt{out_i^1} \cdot \mathtt{out_i^2} \cdot (\mathtt{SQ_i} \| \mathtt{C_j})$$

and we add the following disjunct to the formula ϕ_h to guarantee a move to the state q_k in an honest run.

Therefore, the formula ϕ_h is a disjunction constructed as follows. For each state q_i of type I, ϕ_h contains a disjunct

$$\widehat{EN}(\mathtt{out_i^1}) \vee \widehat{EN}(\mathtt{out_i^2}) \vee \widehat{EN}(\mathtt{out_i^2}, \mathtt{out_k^1}) \vee \widehat{EN}(\mathtt{out_k^1}).$$

For each state q_i of type II, ϕ_h contains two disjuncts. The first is

$$\neg EN(\mathtt{dec_j^1}) \wedge \neg EN(\mathtt{dec_j^2}) \wedge \neg EN(\mathtt{dec_j^3}) \wedge$$

$$\wedge (\widehat{EN}(\mathtt{out_i^1}) \vee \widehat{EN}(\mathtt{out_i^2}) \vee \widehat{EN}(\mathtt{out_i^2}, \mathtt{out_k^1}) \vee \widehat{EN}(\mathtt{out_k^1}))$$

and the second is

$$(EN(\mathtt{dec_j^1}) \vee EN(\mathtt{dec_j^2}) \vee EN(\mathtt{dec_j^3})) \wedge$$

$$\wedge (\widehat{EN}(\mathtt{out_i^1}) \vee \widehat{EN}(\mathtt{out_i^1}, \mathtt{dec_j^2}) \vee \widehat{EN}(\mathtt{out_i^2}, \mathtt{dec_j^2}) \vee$$

$$\vee \widehat{EN}(\mathtt{out_i^2}, \mathtt{dec_j^3}) \vee \widehat{EN}(\mathtt{out_i^2}, \mathtt{out_k^1}, \mathtt{dec_j^3}) \vee \widehat{EN}(\mathtt{out_k^1}, \mathtt{dec_j^3})).$$

Hence, the multiple enabling of $\mathtt{out_i^1}$ and $\mathtt{out_i^2}$ is omitted by the construction. It remains to focus on situations for $\mathtt{dec_i^2}$ and $\mathtt{dec_i^3}$. As the states where both $\mathtt{dec_i^2}$ and $\mathtt{dec_i^3}$ are enabled do not satisfy ϕ_h, each state satisfying $\exists(\mathtt{dec_i^2})EN(\mathtt{dec_i^2})$ has no continuation to make a honest run and each state satisfying $\exists(\mathtt{dec_i^3})EN(\mathtt{dec_i^3})$ is unreachable in any honest run.

9.6 Summary

The following table describes the current state of (un)decidability results for the six problems defined at the end of Subsection 9.2 for the classes of PRS-hierarchy and their extended counterparts. The results established by this chapter are typeset in bold.

problem	decidable for	undecidable for
decidability of EF logic	wPAD [BST06]	**seBPP**
reachability HM property	**wPRS**	sePA[BEH95]
reachability simple property	**wPRS**	sePA[BEH95]
decidability of EG logic	PDA [MS85, Cau92]	BPP [EK95]
evitability HM property	PDA [MS85, Cau92]	BPP [EK95]
evitability simple property	PDA [MS85, Cau92]	**BPP**

To sum up, the situation with (un)decidability of these six problems for all the considered classes is clear for now. See Figure 9.1 and Figure 9.2 depicting (un)decidability boundaries of all EF and EG properties. (Note that s.p. stands for simple property in these figures.)

9.7 Conclusion

In this chapter we have shown that given any wPRS system Δ and any Hennessy–Milner formula φ, one can decide whether there is a state α of Δ reachable from the initial state of Δ such that α satisfies φ. Using Theorem 22 of [JKM01], our result implies that strong bisimilarity between wPRS and finite-state systems is decidable. Decidability of the same problem for some of the wPRS subclasses, namely PAN and PRS, has been formulated as an open question, see e.g. [Srb02a].

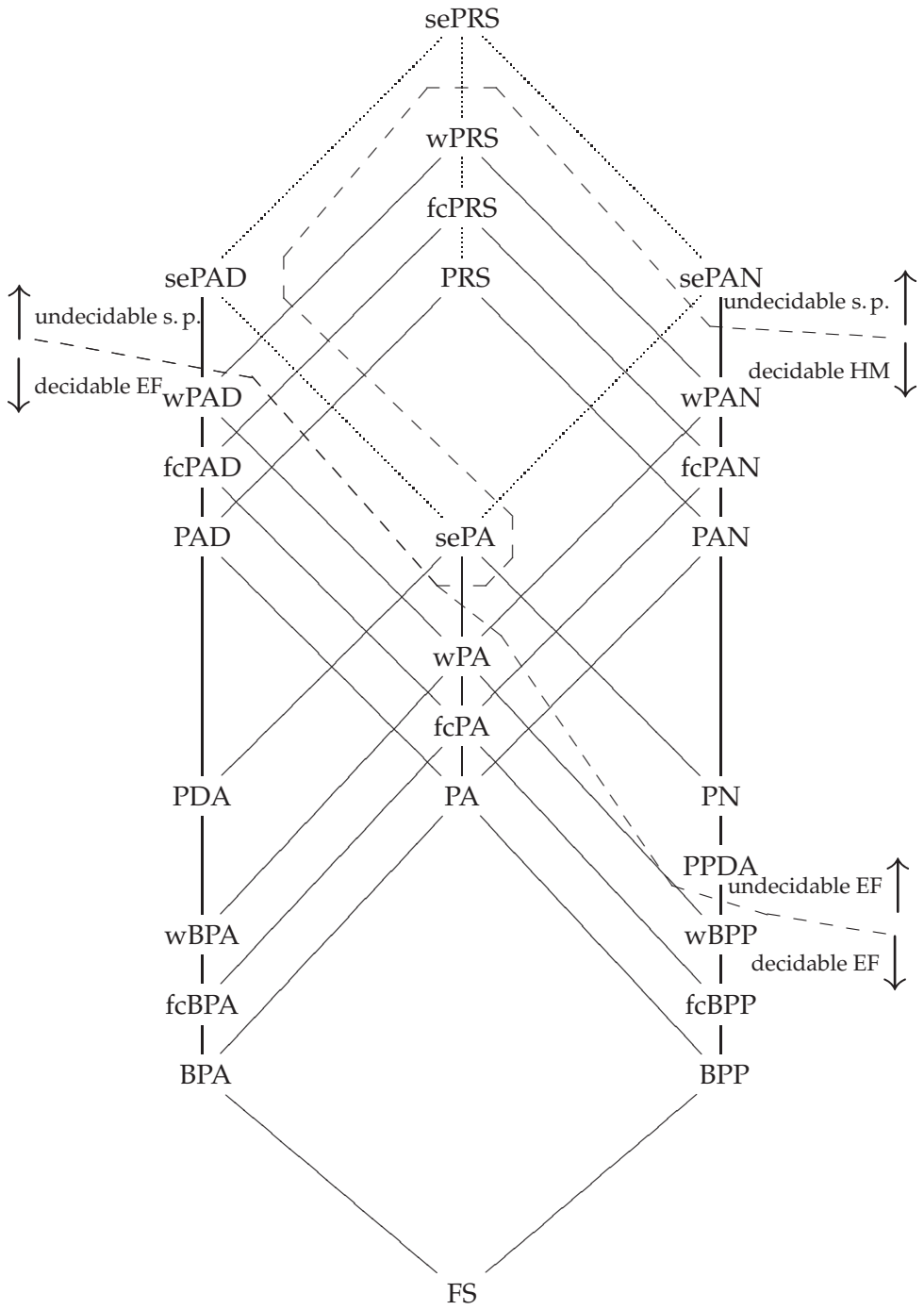


Figure 9.1: The extended PRS-hierarchy with (un)decidability boundaries of EF properties.

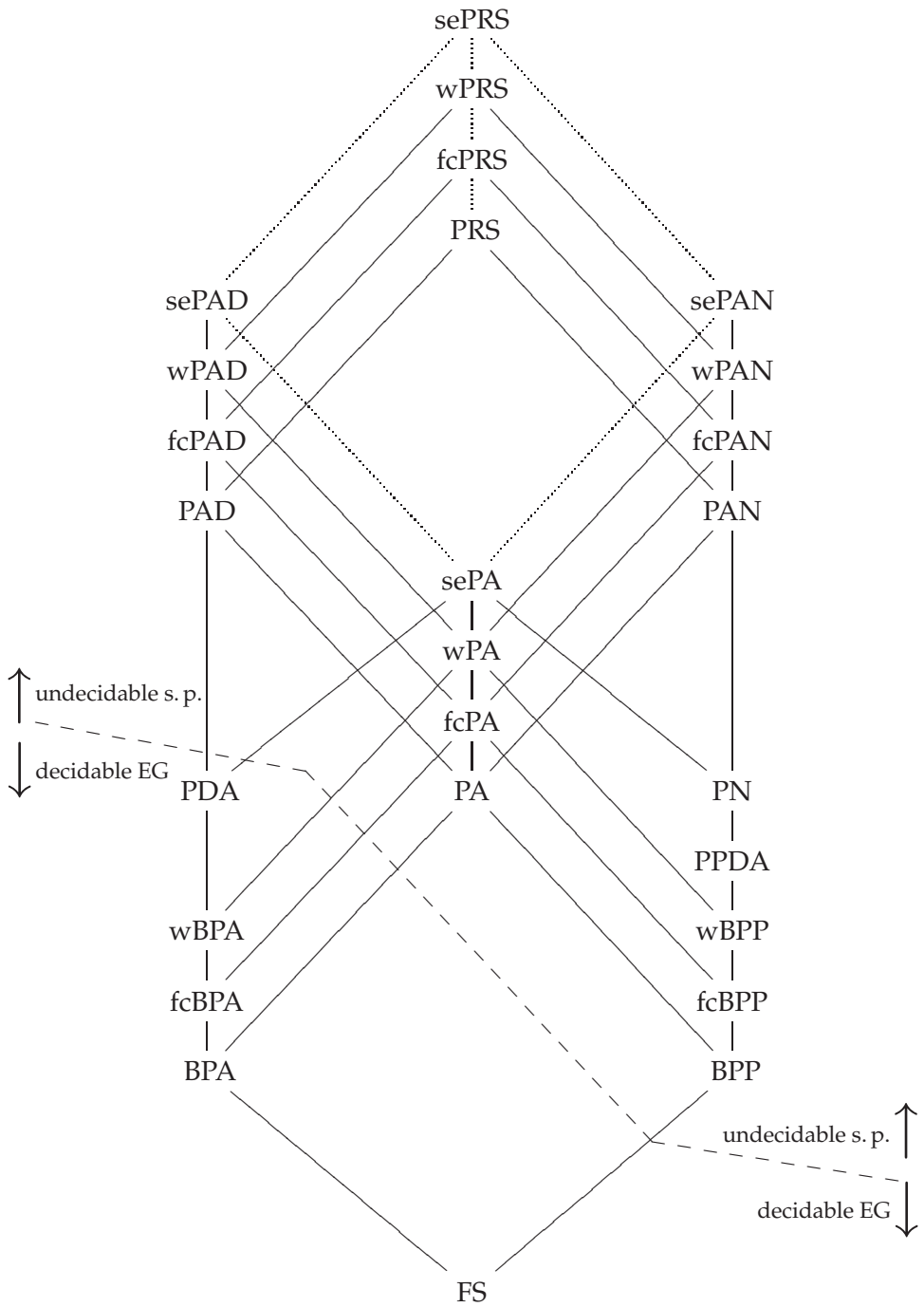


Figure 9.2: The extended PRS-hierarchy with (un)decidability boundaries of EG properties.

Chapter 10

Linear Time Logic

In this chapter we establish decidability boundaries of the model checking problem for infinite-state systems of the extended PRS-hierarchy and properties described by basic fragments of action-based *Linear Temporal Logic* (LTL). It is known that the problem for general LTL properties is decidable for Petri nets and for pushdown processes, while it is undecidable for PA processes. As our main result, we show that the problem is decidable for wPRS if we consider properties defined by formulae only with the *strict eventually* and *strict always* modalities. Later on we extend the result also with past modalities and show that the model checking problem is decidable even for wPRS and LTL fragment based on modalities *strict eventually*, *strict always*, *eventually in the strict past* and *always in the strict past*. Moreover, we show that the problem remains undecidable for PA processes even with respect to the LTL fragment with modality *until* and to the LTL fragment based on modalities *next* and *infinitely often* as well.

10.1 Motivation

Recall that concerning the model checking problem, a broad overview of (un)decidability results for subclasses of PRS and various temporal logics can be found in [May98]. Here we focus exclusively on *Linear Temporal Logic* (LTL). It is known that LTL model checking of PDA is EXPTIME-complete [BEM97]. LTL model checking of PN is also decidable, but at least as hard as the reachability problem for PN [Esp94] (the reachability problem is EXPSPACE-hard [May84, Lip76] and no primitive recursive upper bound is known). If we consider only infinite runs, then the problem for PN is EXPSPACE-complete [Hab97, May98].

Conversely, LTL model checking is undecidable for all classes subsuming PA [BH96, May98]. So far, there are only two positive results for these classes. Bouajjani and Habermehl [BH96] have identified a fragment called *simple PLTL*$_\Box$ for which model checking of infinite runs is decidable for PA

(strictly speaking, simple PLTL$_\Box$ is not a fragment of LTL as it can express also some non-regular properties, while LTL cannot). Only recently, it has been demonstrated that model checking of infinite runs is decidable for PRS and the fragment of LTL capturing exactly fairness properties [Boz05].

In this chapter, we completely locate decidability boundaries of the model checking problem for all subclasses of sePRS and all *basic LTL fragments* (see Figure 10.2), where a basic LTL fragment is a set of all formulae containing only a given subset of standard modalities. The boundaries are depicted in Figure 10.3.

To locate the boundaries, we show the following results.

1. We introduce a new LTL fragment $\mathcal{A}$ and prove that every formula of the basic fragment LTL($\mathsf{F_s}, \mathsf{G_s}$) (i.e. the fragment with modalities *strict eventually* and *strict always* only) can be effectively translated into $\mathcal{A}$. As LTL($\mathsf{F_s}, \mathsf{G_s}$) is closed under negation, we can also translate LTL($\mathsf{F_s}, \mathsf{G_s}$) formulae into negated formulae of $\mathcal{A}$.

2. We show that model checking (of both finite and infinite runs) of wPRS against negated formulae of $\mathcal{A}$ is decidable. We employ the result of [Boz05] and our results presented in Chapter 8 (published in [KŘS04a]) and Chapter 9 (published in [KŘS05]). The proof reduces the problem to LTL model checking problems for PDA and PN. Thus, we get decidability of model checking for wPRS against LTL($\mathsf{F_s}, \mathsf{G_s}$). Note that LTL($\mathsf{F_s}, \mathsf{G_s}$) is strictly more expressive than the *Lamport logic* (i.e. the basic fragment with modalities *eventually* and *always*), which is again strictly more expressive than the mentioned fragment of fairness properties and also more expressive than the *regular* part of simple PLTL$_\Box$.

3. We extend the proof technique, described in the two previous items, with past modalities and show that the model checking problem stays decidable even for wPRS and LTL($\mathsf{F_s}, \mathsf{P_s}$) (i.e. the fragment with modalities *strict eventually*, *strict always*, *eventually in the strict past*, and *always in the strict past*). Let us mention that the expressive power of the fragment LTL($\mathsf{F_s}, \mathsf{P_s}$) semantically coincides with formulae of First-Order Monadic Logic of Order containing at most 2 variables and no successor predicate ($\mathsf{FO}^2[<]$) [EVW02]. Therefore, we also positively solve the model checking problem for wPRS and $\mathsf{FO}^2[<]$. Moreover, we note that First-Order Monadic Logic of Order containing at most 2 variables (FO^2) coincides with an LTL($\mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y}$) fragment [EVW02]. Due to our results mentioned in the next item, we conclude that FO^2 model checking problem is undecidable even for the PA class. For the sake of completeness, we note that First-Order Monadic Logic of Order containing at most 3 variables (FO^3) coincides with the set of all LTL formulae [Kam68, GPSS80].

4. We demonstrate that the model checking problem remains undecidable for PA systems even if we consider the basic fragment with modality *until* or the basic fragment with modalities *next* and *infinitely often* (which is strictly less expressive than the one with *next* and *eventually*).

The chapter is organised as follows. The next section recalls basic definitions. Sections 10.3, 10.4, 10.5, and 10.6 correspond, respectively, to the four results listed above. The Section 10.7 summarises all (un)decidability results on model checking of LTL fragments and the classes of the extended PRS-hierarchy. The last section discusses other potential applications of our results.

10.2 Definitions of the Studied Problems

Before we formulate the model checking problems examined in this chapter, we introduce some notions related to the discussed systems and LTL fragments.

Additional System Definitions

Let us recall that a state pt is *terminal*, written $pt \not\rightarrow_\Delta$, if there is no transition $pt \xrightarrow{a}_\Delta p't'$ for any state $p't'$ and any action a. In this chapter we consider only systems where the initial term is a single constant (see Remark 2.9). Moreover, we always consider only systems where the initial state is not terminal. This restriction do not cause any weakening of our results. It is easy to determine that the initial state is terminal and deciding model checking problem is easy for this case, too.

Definition 10.1. *A (finite or infinite) sequence*

$$\sigma = p_1t_1 \xrightarrow{a_1}_\Delta p_2t_2 \xrightarrow{a_2}_\Delta \dots \xrightarrow{a_n}_\Delta p_{n+1}t_{n+1} \left(\xrightarrow{a_{n+1}}_\Delta \dots \right)$$

is called a derivation over the word $u = a_1a_2 \dots a_n(a_{n+1} \dots)$ in Δ. *A derivation in Δ is called a* run of Δ *if it starts in the initial state p_0X_0 and it is either infinite, or its last state is terminal.*

Finite derivations are also denoted as $p_1t_1 \xrightarrow{u}{}^*_\Delta p_{n+1}t_{n+1}$, infinite ones as $p_1t_1 \longrightarrow^\omega_\Delta$. Further, $L(\Delta)$ denotes the set of words u such that there is a run of Δ over u.

Definition 10.2. *For technical reasons, we define a normal form of wPRS systems. A rewrite rule is* parallel *or* sequential *if it has one of the following forms:*[1]

[1]Note that, due to technical reasons, the parallel rules of this definition are not as restrictive as those of Definition 8.1.

Parallel rules: $p\,(X_1 \| X_2 \| \ldots \| X_n) \overset{a}{\hookrightarrow} q\,(Y_1 \| Y_2 \| \ldots \| Y_m)$

Sequential rules: $pX \overset{a}{\hookrightarrow} qY.Z \quad pX.Y \overset{a}{\hookrightarrow} qZ \quad pX \overset{a}{\hookrightarrow} qY \quad pX \overset{a}{\hookrightarrow} q\varepsilon$

where $X, Y, X_i, Y_j, Z \in Const$, $p, q \in M$, $n > 0$, $m \geq 0$, *and* $a \in Act$. *A rule is called* trivial *if it is both parallel and sequential (i.e. it has the form* $pX \overset{a}{\hookrightarrow} qY$ *or* $pX \overset{a}{\hookrightarrow} q\varepsilon$*). A wPRS* Δ *is in* normal form *if it has only parallel and sequential rewrite rules.*

Linear Temporal Logic

The Linear Temporal Logic (LTL) [Pnu77] can be (equivalently) defined with use of various sets of temporal operators. In the following definition, we employ just two common future temporal operators, namely *next* and *until*, and two common past temporal operators, namely *previously* and *since*.

Definition 10.3. *Let* $Act = \{a, b, \cdots\}$ *be a countably infinite set of atomic actions. The syntax of* Linear Temporal Logic *(LTL) is defined as*

$$\varphi \ ::= \ tt \ \mid \ a \ \mid \ \neg\varphi \ \mid \ \varphi \wedge \varphi \ \mid \ \mathsf{X}\varphi \ \mid \ \varphi \,\mathsf{U}\, \varphi \ \mid \ \mathsf{Y}\varphi \ \mid \ \varphi \,\mathsf{S}\, \varphi,$$

where a *ranges over* Act, X *is called* next, U *is called* until, Y *is called* previously, *and* S *is called* since.

Other boolean connectives $\vee$, $\Rightarrow$, and $\Leftrightarrow$ are derived operators defined in the very standard way. Later on, we also define other modalities as abbreviations of expressions based on the defined four modalities.

The logic is interpreted over infinite as well as nonempty finite words of actions. Given a word u, by $|u|$ we denote the length of u (for infinite words we put $|u| = \infty$). An empty word ε has a zero length. A concatenation of a finite word v and a word u is denoted by $v.u$ or vu. For all $0 \leq i < |u|$ by $u(i)$ we denote the $(i{+}1)^{th}$ letter of u, i.e. $u = u(0)u(1)u(2)\ldots u(|u|-1)$ if u is finite and $u = u(0)u(1)u(2)\ldots$ otherwise. Further, for all $0 \leq i < |u|$ by u_i we denote the i^{th} suffix of u, i.e. $u_i = u(i)u(i+1)u(i+2)\ldots u(|u|-1)$ if u is finite and $u_i = u(i)u(i+1)u(i+2)\ldots$ otherwise. A *pointed word* is a pair (u, i) of a nonempty word u and a *position* $0 \leq i < |u|$ in this word.

The semantics of LTL formulae is defined inductively in the following definition.

Definition 10.4. *Let* φ *be an LTL formula and* (u, i) *be a pointed word. We define that a formula* φ *is* valid *for the position* i *in the word* u*, written* $(u, i) \models \varphi$*, by induction on the structure of* u.

$$
\begin{array}{lll}
(u,i) \models tt & & \\
(u,i) \models a & \textit{iff} & u(i) = a \\
(u,i) \models \neg\varphi & \textit{iff} & (u,i) \not\models \varphi \\
(u,i) \models \varphi_1 \wedge \varphi_2 & \textit{iff} & (u,i) \models \varphi_1 \ \wedge \ (u,i) \models \varphi_2 \\
(u,i) \models \mathsf{X}\varphi & \textit{iff} & i+1 < |u| \ \wedge \ (u,i+1) \models \varphi \\
(u,i) \models \varphi_1 \,\mathsf{U}\, \varphi_2 & \textit{iff} & \exists k.\, (i \leq k < |u| \ \wedge \ (u,k) \models \varphi_2 \ \wedge \\
 & & \qquad \wedge \ \forall j.\, (i \leq j < k \ \Rightarrow \ (u,j) \models \varphi_1)) \\
(u,i) \models \mathsf{Y}\varphi & \textit{iff} & 0 < i \ \wedge \ (u,i-1) \models \varphi \\
(u,i) \models \varphi_1 \,\mathsf{S}\, \varphi_2 & \textit{iff} & \exists k.\, (0 \leq k \leq i \ \wedge \ (u,k) \models \varphi_2 \ \wedge \\
 & & \qquad \wedge \ \forall j.\, (k < j \leq i \ \Rightarrow \ (u,j) \models \varphi_1))
\end{array}
$$

We say that a nonempty word u *satisfies* φ, written $u \models \varphi$, if and only if $(u,0) \models \varphi$. Given a set of words L, we write $L \models \varphi$ if $u \models \varphi$ holds for all $u \in L$. We say that a derivation (or run) σ over a word u satisfies φ, written $\sigma \models \varphi$, whenever $u \models \varphi$.

Considering all pointed words or just the pointed words of the form $(u,0)$ yields different notions of equivalence.

Definition 10.5. *Let φ and ψ be LTL formulae. The formulae are said to be* (initially) equivalent*, written $\varphi \equiv_i \psi$, if for all words u we have*

$$u \models \varphi \ \textit{ if and only if } \ u \models \psi.$$

The formulae are said to be globally equivalent*, written $\varphi \equiv_g \psi$, if for all words u and for all positions $0 \leq i < |u|$ we have*

$$(u,i) \models \varphi \ \textit{ if and only if } \ (u,i) \models \psi.$$

One can readily confirm that if two formulae are globally equivalent then they are also initially equivalent.

Remark 10.6. *Let us mention that there is no difference between initial and global versions of equivalence as far as we consider only so called future modalities like until and next. Indeed, for all pointed words (u,i) and all formulae φ containing these future modalities only, it holds that*

$$(u,i) \models \varphi \ \textit{ if and only if } \ u_i \models \varphi$$

and hence the initial and global equivalences coincide. Contrary, the difference between initial and global versions arises as soon as past modalities are introduced.

modality	meaning	name
X	see Definition 10.4	*next*
U	see Definition 10.4	*until*
F	$\mathsf{F}\varphi \equiv_g tt\,\mathsf{U}\,\varphi$	*eventually*
G	$\mathsf{G}\varphi \equiv_g \neg\mathsf{F}\neg\varphi$	*always*
$\mathsf{F_s}$	$\mathsf{F_s}\varphi \equiv_g \mathsf{XF}\varphi$	*strict eventually*
$\mathsf{G_s}$	$\mathsf{G_s}\varphi \equiv_g \neg\mathsf{F_s}\neg\varphi$	*strict always*
$\overset{\infty}{\mathsf{F}}$	$\overset{\infty}{\mathsf{F}}\varphi \equiv_g \mathsf{GF}\varphi$	*infinitely often*
$\overset{\infty}{\mathsf{G}}$	$\overset{\infty}{\mathsf{G}}\varphi \equiv_g \neg\overset{\infty}{\mathsf{F}}\neg\varphi$	*almost always*
Y	see Definition 10.4	*previously*
S	see Definition 10.4	*since*
P	$\mathsf{P}\varphi \equiv_g tt\,\mathsf{S}\,\varphi$	*eventually in the past*
H	$\mathsf{H}\varphi \equiv_g \neg\mathsf{P}\neg\varphi$	*always in the past*
$\mathsf{P_s}$	$\mathsf{P_s}\varphi \equiv_g \mathsf{YP}\varphi$	*eventually in the strict past*
$\mathsf{H_s}$	$\mathsf{H_s}\varphi \equiv_g \neg\mathsf{P_s}\neg\varphi$	*always in the strict past*
I	$\mathsf{I}\varphi \equiv_g \mathsf{HP}\varphi$	*initially*

Figure 10.1: LTL modalities overview

In Figure 10.1 we enrich the syntax of LTL with other modalities. For the sake of completeness, the presented list of modalities includes also the operators which has been already defined.

Note that $\mathsf{F}\varphi$ is equivalent to $\varphi \vee \mathsf{F_s}\varphi$ but $\mathsf{F_s}\varphi$ cannot be expressed with F as the only modality. Thus $\mathsf{F_s}$ is "stronger" than F. The same relations hold between $\mathsf{G_s}$ and G, $\mathsf{P_s}$ and P, and $\mathsf{H_s}$ and H.

For a set $\{O_1, \ldots, O_n\}$ of modalities, let LTL$(O_1, \ldots, O_n)$ denote the LTL fragment containing all formulae with modalities $O_1, \ldots, O_n$ only. Such a fragment is called *basic*. Figure 10.2 shows an expressiveness hierarchy of all the basic LTL fragments studied here. Indeed, every basic LTL fragment using standard[2] modalities is (initially) equivalent to one of the fragments in the hierarchy. For example, any LTL(S, Y) formula is initially equivalent to a formula without any modality. Hence, every LTL$(\mathsf{U}, \mathsf{X}, \mathsf{S}, \mathsf{Y})$ formula φ can be effectively translated into a formula of LTL(U, X) that is

[2]By standard modalities we mean the ones defined here and also other commonly used modalities like *strict until*, *release*, *weak until*, etc. However, it is well possible that one can define a new modality such that there is a basic fragment not equivalent to any of the fragments in the hierarchy.

initially equivalent. To sum up, every LTL formula with arbitrary modalities is initially equivalent to a formula of LTL(U, X). Let us also note that, e.g. LTL($\mathsf{F_s}$) $\equiv$ LTL($\mathsf{F_s}, \mathsf{G_s}$) even with respect to global equivalence. Further, the hierarchy of Figure 10.2 is strict with respect to initial equivalence. For detailed information about expressiveness of LTL modalities and LTL fragments we refer to [Str04].

Model Checking Problem

Let $\mathcal{F}$ be an LTL fragment and $\mathcal{C}$ be a class of systems.

Problem: *Model checking problem for $\mathcal{F}$ and $\mathcal{C}$*
Instance: A formula $\varphi \in \mathcal{F}$ and a system $\Delta \in \mathcal{C}$
Question: Is the system Δ a model of the formula φ, i.e. $L(\Delta) \models \varphi$?

We also mention a problem called *model checking of infinite runs*, where $L(\Delta) \cap Act^{\omega} \models \varphi$ is examined, and a problem called *model checking of finite runs*, where $L(\Delta) \cap Act^{*} \models \varphi$ is examined.

Using this model checking problem definition we study behaviour of the initial state only. In context of LTL with past modalities, it is natural to consider also an extend model checking problem reflecting behaviour of a given (non-initial) state. We call this problem a *pointed* model checking problem. Before the formal definition of the pointed model checking problem, we present some auxiliary definitions.

Definition 10.7. *Let Δ be a wPRS with the initial state p_0t_0 and pt be a reachable nonterminal state of Δ. We define $L(pt, \Delta)$ to be a set of all pointed words (u, i) such that there is a (finite or infinite) run over the word u*

$$p_0t_0 \xrightarrow{u(0)}_{\Delta} p_1t_1 \xrightarrow{u(1)}_{\Delta} \cdots \xrightarrow{u(n)}_{\Delta} p_{n+1}t_{n+1} \left(\xrightarrow{u(n+1)}_{\Delta} \cdots \right)$$

and $pt = p_it_i$. Further, we say that a set of pointed words $L(pt, \Delta)$ satisfies *an LTL formula φ, written $L(pt, \Delta) \models \varphi$, if and only if $(u, i) \models \varphi$ for all $(u, i) \in L(pt, \Delta)$.*

Note that every (u, i) of the previous definition is a pointed word, because the state pt is not terminal. As pt is reachable in Δ, $L(pt, \Delta)$ is nonempty. Now, the pointed model checking problem for an LTL fragment $\mathcal{F}$ and a system class $\mathcal{C}$ can be formally defined as follows:

Problem: *Pointed model checking problem for $\mathcal{F}$ and $\mathcal{C}$*
Instance: A formula $\varphi \in \mathcal{F}$, a system $\Delta \in \mathcal{C}$, and a reachable nonterminal state pt of Δ
Question: Is the state pt of the system Δ a model of the formula φ, i.e. $L(pt, \Delta) \models \varphi$?

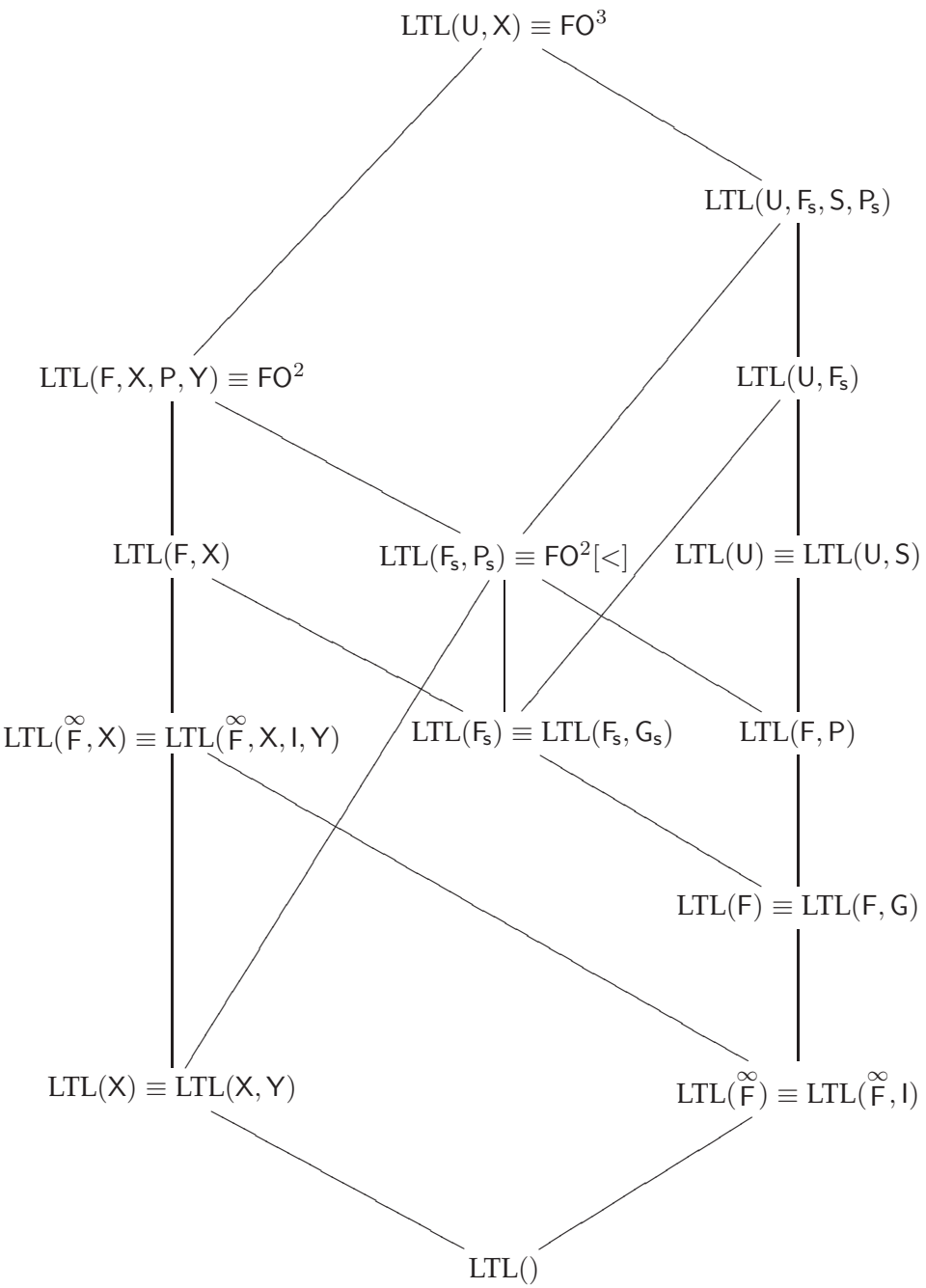


Figure 10.2: The hierarchy of basic LTL fragments with respect to (initial) equivalence.

10.3 Fragment $\mathcal{A}$ and Translation of LTL($\mathsf{F_s}, \mathsf{G_s}$) into $\mathcal{A}$

In this section we introduce a new LTL fragment $\mathcal{A}$ and prove that every formula of the basic fragment LTL($\mathsf{F_s}, \mathsf{G_s}$) can be effectively translated into $\mathcal{A}$. In other words, we show that for each LTL($\mathsf{F_s}, \mathsf{G_s}$) formula one can find an (globally) equivalent formula of $\mathcal{A}$.

Recall that LTL() denotes the fragment of formulae without any modality, i.e. boolean combinations of actions. In the following we use $\varphi_1 \,\mathsf{U}_+\, \varphi_2$ to abbreviate $\varphi_1 \wedge \mathsf{X}(\varphi_1 \,\mathsf{U}\, \varphi_2)$.

Definition 10.8. *Let $\delta = \theta_1 O_1 \theta_2 O_2 \dots \theta_n O_n \theta_{n+1}$, where*

- $n > 0$,
- $\theta_i \in$ *LTL()* *for each $i \leq n+1$,*
- *O_i is either 'U' or 'U_+' or '$\wedge\,\mathsf{X}$' for each $i < n$, and*
- *O_n is '$\wedge\,\mathsf{G_s}$'.*

Further, let $\mathcal{B} \subseteq$ LTL() be a finite set. An α-formula *is defined as*

$$\alpha(\delta, \mathcal{B}) = \big(\theta_1 O_1 (\theta_2 O_2 \dots (\theta_n O_n \theta_{n+1}) \dots)\big) \;\wedge \bigwedge_{\psi \in \mathcal{B}} \mathsf{G_s F_s} \psi$$

The $\mathcal{A}$ fragment *consists of finite disjunctions of α-formulae.*

Hence, a word u satisfies $\alpha(\delta, \mathcal{B})$ if and only if u can be written as

$$u_1 . u_2 . \cdots . u_{n+1}$$

where

- each u_i, for $i = 1, \dots, n+1$, consists only of actions satisfying θ_i and
  - $|u_i| \geq 0$ if $i = n+1$ or O_i is 'U',
  - $|u_i| > 0$ if O_i is 'U_+',
  - $|u_i| = 1$ if O_i is '$\wedge\,\mathsf{X}$' or '$\wedge\,\mathsf{G_s}$', and
- u_{n+1} satisfies $\mathsf{G_s F_s} \psi$ for every $\psi \in \mathcal{B}$.

Lemma 10.9. *A conjunction of α-formulae can be effectively converted into an equivalent disjunction of α-formulae.*

Proof of this lemma is a simple but very technical exercise and we sketch only the basic ideas here.

Proof (sketch). Let $\alpha(\delta_1, \mathcal{B}_1)$ and $\alpha(\delta_2, \mathcal{B}_2)$ be the two disjuncts. Clearly, the set $\mathcal{B}$ of each α-formula of the desired disjunction is $\mathcal{B}_1 \cup \mathcal{B}_2$.

Each δ part (see Definition 10.8) of an α-formula can be represented as a weak finite state automaton[3] accepting exactly those words that models the δ formula.[4] Vice versa, every weak automaton can be represented as a disjunction of δ formulae. In particular, every δ formula stays for one "loop-less" run from the initial state to an accepting state.

Therefore, we construct a synchronous product of two weak automata representing δ_1 and δ_2. It is clear that the resulting automaton is also weak. Hence, there is a disjunction of δ formulae representing the automaton. These δ formulae compose the required disjunction of α-formulae. □

Theorem 10.10. *Every LTL*$(\mathsf{F_s}, \mathsf{G_s})$ *formula can be translated into an equivalent disjunction of α-formulae.*

Proof. As $\mathsf{F_s}$ and $\mathsf{G_s}$ are dual modalities, we can assume that every LTL$(\mathsf{F_s}, \mathsf{G_s})$ formula contains negations only in front of actions. Given an LTL$(\mathsf{F_s}, \mathsf{G_s})$ formula φ, we construct a finite set A_φ of α-formulae such that φ is equivalent to disjunction of formulae in A_φ. Although our proof looks like a proof by induction on the structure of φ, in fact it is done by induction on the length of φ. Thus, if $\varphi \notin \mathrm{LTL}()$, then we assume that for every LTL$(\mathsf{F_s}, \mathsf{G_s})$ formula φ' shorter than φ we can construct the corresponding set $A_{\varphi'}$. In this proof, let p denotes a formula of LTL(). The structure of φ fits into one of the following cases.

- •p **Case** p**:** In this case, φ is equivalent to $p \wedge \mathsf{G_s}tt$. Hence $A_\varphi = \{\alpha(p \wedge \mathsf{G_s}tt, \emptyset)\}$.

- •$\vee$ **Case** $\varphi_1 \vee \varphi_2$**:** Due to induction hypothesis, we can assume that we have sets A_{φ_1} and A_{φ_2}. Clearly, $A_\varphi = A_{\varphi_1} \cup A_{\varphi_2}$.

- •$\wedge$ **Case** $\varphi_1 \wedge \varphi_2$**:** Due to Lemma 10.9, the set A_φ can be constructed from the sets A_{φ_1} and A_{φ_2}.

- •$\mathsf{F_s}$ **Case** $\mathsf{F_s}\varphi_1$**:** As $\mathsf{F_s}(\alpha_1 \vee \alpha_2) \equiv (\mathsf{F_s}\alpha_1) \vee (\mathsf{F_s}\alpha_2)$ and $\mathsf{F_s}(\alpha \wedge \mathsf{G_s}\mathsf{F_s}\phi) \equiv (\mathsf{F_s}\alpha) \wedge (\mathsf{G_s}\mathsf{F_s}\phi)$, we set $A_\varphi = \{\alpha(tt\ \mathsf{U}_+\ \delta, \mathcal{B}) \mid \alpha(\delta, \mathcal{B}) \in A_{\varphi_1}\}$.

- •$\mathsf{G_s}$ **Case** $\mathsf{G_s}\varphi_1$**:** This case is divided into the following subcases according to the structure of φ_1.

  - ∘p **Case** $\mathsf{G_s}p$**:** As $\mathsf{G_s}p$ is equivalent to $tt \wedge \mathsf{G_s}p$, we set $A_\varphi = \{\alpha(tt \wedge \mathsf{G_s}p, \emptyset)\}$.

[3] A weak finite state automaton is a weak FSU supplemented with a specification of accepting states — see Definition 11.3.

[4] In fact, the transformation of δ onto a weak finite state automaton is given in the proof of Theorem 10.11.

$\circ\wedge$ **Case** $\mathsf{G_s}(\varphi_2 \wedge \varphi_3)$**:** As $\mathsf{G_s}(\varphi_2 \wedge \varphi_3) \equiv (\mathsf{G_s}\varphi_2) \wedge (\mathsf{G_s}\varphi_3)$, the set A_φ can be constructed from $A_{\mathsf{G_s}\varphi_2}$ and $A_{\mathsf{G_s}\varphi_3}$ using Lemma 10.9. Note that $A_{\mathsf{G_s}\varphi_2}$ and $A_{\mathsf{G_s}\varphi_3}$ can be constructed because $\mathsf{G_s}\varphi_2$ and $\mathsf{G_s}\varphi_3$ are shorter than $\mathsf{G_s}(\varphi_2 \wedge \varphi_3)$.

$\circ\mathsf{F_s}$ **Case** $\mathsf{G_s}\mathsf{F_s}\varphi_2$**:** This case is again divided into the following subcases.

$-p$ **Case** $\mathsf{G_s}\mathsf{F_s}p$**:** As $p \in$ LTL(), we directly set $A_\varphi = \{\alpha(tt \wedge \mathsf{G_s}tt, \{p\})\}$.

$-\vee$ **Case** $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \vee \varphi_4)$**:** As $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \vee \varphi_4) \equiv (\mathsf{G_s}\mathsf{F_s}\varphi_3) \vee (\mathsf{G_s}\mathsf{F_s}\varphi_4)$, we set $A_\varphi = A_{\mathsf{G_s}\mathsf{F_s}\varphi_3} \cup A_{\mathsf{G_s}\mathsf{F_s}\varphi_4}$.

$-\wedge$ **Case** $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \varphi_4)$**:** This case is also divided into subcases depending on the formulae φ_3 and φ_4.

$*p$ **Case** $\mathsf{G_s}\mathsf{F_s}(p_3 \wedge p_4)$**:** As $p_3 \wedge p_4 \in$ LTL(), this subcase has already been covered by Case $\mathsf{G_s}\mathsf{F_s}p$.

$*\vee$ **Case** $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge (\varphi_5 \vee \varphi_6))$**:** As $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge (\varphi_5 \vee \varphi_6)) \equiv \mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \varphi_5) \vee \mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \varphi_6)$, we set $A_\varphi = A_{\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \varphi_5)} \cup A_{\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \varphi_6)}$.

$*\mathsf{F_s}$ **Case** $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \mathsf{F_s}\varphi_5)$**:** As $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \mathsf{F_s}\varphi_5) \equiv (\mathsf{G_s}\mathsf{F_s}\varphi_3) \wedge (\mathsf{G_s}\mathsf{F_s}\varphi_5)$, the set A_φ can be constructed from $A_{\mathsf{G_s}\mathsf{F_s}\varphi_3}$ and $A_{\mathsf{G_s}\mathsf{F_s}\varphi_5}$ using Lemma 10.9.

$*\mathsf{G_s}$ **Case** $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \mathsf{G_s}\varphi_5)$**:** As $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \mathsf{G_s}\varphi_5) \equiv (\mathsf{G_s}\mathsf{F_s}\varphi_3) \wedge (\mathsf{G_s}\mathsf{F_s}\mathsf{G_s}\varphi_5)$, the set A_φ can be constructed from $A_{\mathsf{G_s}\mathsf{F_s}\varphi_3}$ and $A_{\mathsf{G_s}\mathsf{F_s}\mathsf{G_s}\varphi_5}$ using Lemma 10.9.

$-\mathsf{F_s}$ **Case** $\mathsf{G_s}\mathsf{F_s}\mathsf{F_s}\varphi_3$**:** As $\mathsf{G_s}\mathsf{F_s}\mathsf{F_s}\varphi_3 \equiv \mathsf{G_s}\mathsf{F_s}\varphi_3$, we set $A_\varphi = A_{\mathsf{G_s}\mathsf{F_s}\varphi_3}$.

$-\mathsf{G_s}$ **Case** $\mathsf{G_s}\mathsf{F_s}\mathsf{G_s}\varphi_3$**:** A word u satisfies $\mathsf{G_s}\mathsf{F_s}\mathsf{G_s}\varphi_3$ iff $|u| = 1$ or u is an infinite word satisfying $\mathsf{F_s}\mathsf{G_s}\varphi_3$. Note that $\mathsf{G_s}\neg tt$ is satisfied only by finite words of length one. Further, a word u satisfies $(\mathsf{F_s}tt) \wedge (\mathsf{G_s}\mathsf{F_s}tt)$ iff u is infinite. Thus, $\mathsf{G_s}\mathsf{F_s}\mathsf{G_s}\varphi_3 \equiv (\mathsf{G_s}\neg tt) \vee \varphi'$ where $\varphi' = (\mathsf{F_s}tt) \wedge (\mathsf{G_s}\mathsf{F_s}tt) \wedge (\mathsf{F_s}\mathsf{G_s}\varphi_3)$. Hence, $A_\varphi = A_{\mathsf{G_s}\neg tt} \cup A_{\varphi'}$ where $A_{\varphi'}$ is constructed from $A_{\mathsf{F_s}tt}$, $A_{\mathsf{G_s}\mathsf{F_s}tt}$, and $A_{\mathsf{F_s}\mathsf{G_s}\varphi_3}$ using Lemma 10.9.

$\circ\vee$ **Case** $\mathsf{G_s}(\varphi_2 \vee \varphi_3)$**:** According to the structure of φ_2 and φ_3, there are the following subcases.

$\star p$ **Case** $\mathsf{G_s}(p_2 \vee p_3)$**:** As $p_2 \vee p_3 \in$ LTL(), this subcase has already been covered by Case $\mathsf{G_s}p$.

$\star\wedge$ **Case** $\mathsf{G_s}(\varphi_2 \vee (\varphi_4 \wedge \varphi_5))$**:** As $\mathsf{G_s}(\varphi_2 \vee (\varphi_4 \wedge \varphi_5)) \equiv \mathsf{G_s}(\varphi_2 \vee \varphi_4) \wedge \mathsf{G_s}(\varphi_2 \vee \varphi_5)$, the set A_φ can be constructed from $A_{\mathsf{G_s}(\varphi_2 \vee \varphi_4)}$ and $A_{\mathsf{G_s}(\varphi_2 \vee \varphi_5)}$ using Lemma 10.9.

$\star\mathsf{F_s}$ **Case** $\mathsf{G_s}(\varphi_2 \vee \mathsf{F_s}\varphi_4)$**:** It holds that $\mathsf{G_s}(\varphi_2 \vee \mathsf{F_s}\varphi_4) \equiv (\mathsf{G_s}\varphi_2) \vee \mathsf{F_s}(\mathsf{F_s}\varphi_4 \wedge \mathsf{G_s}\varphi_2) \vee \mathsf{G_s}\mathsf{F_s}\varphi_4$. Therefore, the set A_φ can be constructed as $A_{\mathsf{G_s}\varphi_2} \cup \{\alpha(tt\, \mathsf{U}_+\, \delta, \mathcal{B}) \mid \alpha(\delta, \mathcal{B}) \in A_{\mathsf{F_s}\varphi_4 \wedge \mathsf{G_s}\varphi_2}\} \cup$

$A_{\mathsf{G_s F_s}\varphi_4}$, where $A_{\mathsf{F_s}\varphi_4 \wedge \mathsf{G_s}\varphi_2}$ is created from $A_{\mathsf{F_s}\varphi_4}$ and $A_{\mathsf{G_s}\varphi_2}$ due to Lemma 10.9.

$\star\mathsf{G_s}$ **Case** $\mathsf{G_s}(\varphi_2 \vee \mathsf{G_s}\varphi_4)$**:** There are only the following two subcases (the others fit to some of the previous cases).

(i) **Case** $\mathsf{G_s}(\bigvee_{\varphi' \in G} \mathsf{G_s}\varphi')$**:** It holds that $\mathsf{G_s}(\bigvee_{\varphi' \in G} \mathsf{G_s}\varphi') \equiv (\mathsf{G_s}\neg tt) \vee \bigvee_{\varphi' \in G}(\mathsf{X}\mathsf{G_s}\varphi')$. Therefore, the set A_φ can be constructed as $A_{\mathsf{G_s}\neg tt} \cup \bigcup_{\varphi' \in G}\{\alpha(tt \wedge \mathsf{X}\delta, \mathcal{B}) \mid \alpha(\delta, \mathcal{B}) \in A_{\mathsf{G_s}\varphi'}\}$.

(ii) **Case** $\mathsf{G_s}(p_2 \vee \bigvee_{\varphi' \in G} \mathsf{G_s}\varphi')$**:** As $\mathsf{G_s}(p_2 \vee \bigvee_{\varphi' \in G} \mathsf{G_s}\varphi') \equiv (\mathsf{G_s}p_2) \vee \bigvee_{\varphi' \in G}(\mathsf{X}(p_2 \,\mathsf{U}\, \mathsf{G_s}\varphi'))$. Therefore, the set A_φ can be constructed as $A_{\mathsf{G_s}p_2} \cup \bigcup_{\varphi' \in G}\{\alpha(tt \wedge \mathsf{X}p_2 \,\mathsf{U}\, \delta, \mathcal{B}) \mid \alpha(\delta, \mathcal{B}) \in A_{\mathsf{G_s}\varphi'}\}$.

$\circ\mathsf{G_s}$ **Case** $\mathsf{G_s}\mathsf{G_s}\varphi_2$**:** As $\mathsf{G_s}(\mathsf{G_s}\varphi_2) \equiv (\mathsf{G_s}\neg tt) \vee (\mathsf{X}\mathsf{G_s}\varphi_2)$, the set A_φ can be constructed as $A_{\mathsf{G_s}\neg tt} \cup \{\alpha(tt \wedge \mathsf{X}\delta, \mathcal{B}) \mid \alpha(\delta, \mathcal{B}) \in A_{\mathsf{G_s}\varphi_2}\}$.

□

10.4 Model Checking of wPRS against Negated $\mathcal{A}$

This section is devoted to decidability of the model checking problem for the wPRS class and negated formulae of the $\mathcal{A}$ fragment. In fact, we prove decidability of the dual problem, i.e. whether a given wPRS system has a run satisfying a given formula of $\mathcal{A}$. Finite and infinite runs are treated separately.

Theorem 10.11. *The problem whether a given wPRS system has a finite run satisfying a given α-formula is decidable.*

Proof. Let Δ be a wPRS system and $\alpha(\delta, \mathcal{B})$ be an α-formula. Note that a formula $\mathsf{G_s F_s}\psi$ is valid on a finite nonempty word if and only if the length of the word is 1. Therefore, if $\mathcal{B} \neq \emptyset$ then it is easy to check whether there is a finite run of Δ satisfying $\alpha(\delta, \mathcal{B})$. In what follows we assume $\mathcal{B} = \emptyset$.

Let $\delta = \theta_1 O_1 \theta_2 O_2 \dots \theta_n O_n \theta_{n+1}$. We construct a wPRS system Δ' with control states $M(\Delta) \times \{1, 2, \dots, n+1\}$ in the following way.

- For any $1 \leq i \leq n$ and every rule $pt_1 \overset{a}{\hookrightarrow} qt_2$ of Δ such that a satisfies θ_i, we add to Δ' the rule $(p,i)t_1 \overset{a}{\hookrightarrow} (q, i+1)t_2$ and if O_i is U or U_+ then also the rule $(p,i)t_1 \overset{a}{\hookrightarrow} (q,i)t_2$.
- For every $p \in M(\Delta)$, $X \in \mathit{Const}(\Delta)$, and for any $1 \leq i \leq n$ such that $O_i = \mathsf{U}$, we add to Δ' the rule $(p,i)X \overset{e}{\hookrightarrow} (p, i+1)X$, where e is an arbitrary action.
- For every rule $pt_1 \overset{a}{\hookrightarrow} qt_2$ of Δ such that a satisfies θ_{n+1}, we add to Δ' the rule $(p, n+1)t_1 \overset{a}{\hookrightarrow} (q, n+1)t_2$.

- For every rule $pt_1 \overset{a}{\hookrightarrow} qt_2$ of Δ we add to Δ' the rule $(p, n+1)t_1 \overset{a}{\hookrightarrow} (p, n+1)t_1$.

Let p_0X_0 be the initial state of Δ. There is a finite run $p_0X_0 \overset{u}{\longrightarrow}^*_\Delta qt$ satisfying $\alpha(\delta, \emptyset)$ if and only if there is a finite run $(p_0, 1)X_0 \overset{v}{\longrightarrow}^*_{\Delta'} (q, n+1)t$. Hence, we need to decide whether there exists a state of the form $(q, n+1)t$ that is terminal and reachable from $(p_0, 1)X_0$. To do that, for every $p \in M(\Delta)$ we add to Δ' the rule $(p, n+1)Z \overset{end}{\hookrightarrow} (p, n+1)\varepsilon$, where $end \notin Act(\Delta)$ is a fresh action and $Z \notin Const(\Delta)$ is a fresh process constant. Now, it holds that Δ has a finite run satisfying $\alpha(\delta, \emptyset)$ if and only if there exists a state of Δ', which is reachable from $(p_0, 1)(X_0 \| Z)$ and the only enabled action in this state is end. This last condition on the state can be expressed by formula $\varphi = \langle end \rangle tt \wedge \bigwedge_{a \in Act(\Delta)} \neg \langle a \rangle tt$ of the Hennessy–Milner logic. As reachability of a state satisfying a given Hennessy–Milner formula is decidable for wPRS (see Chapter 9 for details), we are done. □

The problem for infinite runs is more complicated. In order to solve it, we introduce more terminology and notation. First we define *β-formulae* and regular languages called *γ-languages*. Let $w = a_1O_1a_2O_2 \dots a_nO_n$, where $n \geq 0$, $a_1, \dots, a_n \in Act$ are pairwise distinct actions and each O_i is either 'U_+' or '$\wedge\mathsf{X}$'. Further, let $B \subseteq Act \setminus \{a_1, \dots, a_n\}$ be a nonempty finite set of actions and $C \subseteq B$. A β-formula $\beta(w, B, C)$ and γ-language $\gamma(w, C)$ are defined as

$$\beta(w, B, C) = \big(a_1O_1(a_2O_2 \dots (a_nO_n \mathsf{G} \bigvee_{b \in B} b) \dots)\big) \wedge \bigwedge_{b \in C} \mathsf{GF}b \wedge \bigwedge_{b \in B \setminus C} (\mathsf{F}b \wedge \neg \mathsf{GF}b)$$

$$\gamma(w, C) = a_1^{o_1}.a_2^{o_2}.\cdots.a_n^{o_n}.L,$$

$$\text{where } o_i = \begin{cases} + & \text{if } O_i = \mathsf{U}_+ \\ 1 & \text{if } O_i = \wedge\mathsf{X} \end{cases} \quad \text{and } L = \begin{cases} \{\varepsilon\} & \text{if } C = \emptyset \\ \bigcap_{b \in C} C^*.b.C^* & \text{otherwise} \end{cases}$$

Roughly speaking, a β-formula is a more restrictive version of an α-formula and in context of β-formulae we consider infinite words only. Contrary to δ of an α-formula, w of a β-formula employs actions rather than LTL() formulae. While a tail of an infinite word satisfying an α-formula is specified by θ_{n+1}, in the definition of β-formulae we use a set B containing exactly all the actions of the tail and its subset C of exactly all actions occurring infinitely many times in the tail.

Remark 10.12. *Note that an infinite word satisfies a formula $\beta(w, B, C)$ if and only if it can be divided into a prefix $u \in \gamma(w, B)$ and a suffix $v \in C^\omega$ such that v contains infinitely many occurrences of every $c \in C$.*

Let w, B, C be defined as above. We say that a finite derivation σ over a word u *satisfies* $\gamma(w, C)$ if and only if $u \in \gamma(w, C)$. We write $(w', B') \sqsubseteq$

(w, B) whenever $B' \subseteq B$ and $w' = a_{i_1} O_{i_1} a_{i_2} O_{i_2} \dots a_{i_k} O_{i_k}$ for some $1 \leq i_1 < i_2 < \dots < i_k \leq n$. Moreover, we write $(w', B', C') \sqsubseteq (w, B, C)$ whenever $(w', B') \sqsubseteq (w, B)$, B' is nonempty, and $C' \subseteq C \cap B'$.

Remark 10.13. *If u is an infinite word satisfying $\beta(w, B, C)$ and v is an infinite* subword *of u (i.e. it arises from u by omitting some letters), then there is exactly one triple $(w', B', C') \sqsubseteq (w, B, C)$ such that $v \models \beta(w', B', C')$. Further, for each finite subword v of u, there is exactly one pair (w', B') such that $(w', B') \sqsubseteq (w, B)$ and $v \in \gamma(w', B')$.*

Given a PRS in normal form, by $tri(\Delta)$, $par(\Delta)$, and $seq(\Delta)$ we denote the system Δ restricted to trivial, parallel, and sequential rules, respectively. A derivation in $tri(\Delta)$ is called a *trivial* derivation in Δ. In the following we write simply tri, par, seq as Δ is always clearly determined by the context.

Definition 10.14. *Let Δ be a PRS in normal form and $\beta(w, B, C)$ be a β-formula. The PRS Δ is in* flat (w, B, C)-form *if and only if for each $X, Y \in Const(\Delta)$, each $(w', B', C') \sqsubseteq (w, B, C)$, and each $B'' \subseteq B$, the following conditions hold:*

1. *If there is a finite derivation $X \xrightarrow{u}{}^* Y$ satisfying $\gamma(w', B'')$, then there is also a finite derivation $X \xrightarrow{v}{}^*_{tri} Y$ satisfying $\gamma(w', B'')$.*

2. *If there is a term t and a finite derivation $X \xrightarrow{u}{}^* t$ satisfying $\gamma(w', B'')$, then there is also a process constant Z and a finite derivation $X \xrightarrow{v}{}^*_{tri} Z$ satisfying $\gamma(w', B'')$.*

3. *If $w' = \varepsilon$ and there is an infinite derivation $X \xrightarrow{u} {}^\omega$ satisfying $\beta(w', B', C')$, then there is also an infinite derivation $X \xrightarrow{v}{}^\omega_{tri}$ satisfying $\beta(w', B', C')$.*

4. *If there is an infinite derivation $X \xrightarrow{u}{}^\omega_{par}$ satisfying $\beta(w', B', C')$, then there is also an infinite derivation $X \xrightarrow{v}{}^\omega_{tri}$ satisfying $\beta(w', B', C')$;*

5. *If there is an infinite derivation $X \xrightarrow{u}{}^\omega_{seq}$ satisfying $\beta(w', B', C')$, then there is also an infinite derivation $X \xrightarrow{v}{}^\omega_{tri}$ satisfying $\beta(w', B', C')$.*

Roughly speaking, a system is in flat (w, B, C)-form if for every derivation of the form given in the definition there is an "equivalent" trivial derivation.

Proposition 10.15 (Fairness problem [Boz05]**).** *For $X, Y \in Const(\Delta)$ and $K, K_\omega \subseteq Act(\Delta)$ it is decidable whether there is an* infinite *derivation of the form $X \xrightarrow{u}{}^\omega$ such that all actions of K occur in u and all actions of K_ω occur infinitely many times in u.*

All conditions of the Definition 10.14 can be checked due to the following lemma, Proposition 10.15 , and decidability of LTL model checking for PDA and PN. Lemma 10.17 says that every PRS in normal form can be transformed into an "equivalent" flat system. Finally, Lemma 10.20 says that if a PRS system in flat (w, B, C)-form has an infinite derivation satisfying $\beta(w, B, C)$, then it has also a trivial infinite derivation satisfying $\beta(w, B, C)$. Note that it is easy to check whether such a trivial derivation exists.

Lemma 10.16. *Given a γ-language $\gamma(w, C)$, a PRS system Δ, and constants X, Y, the following problems are decidable:*
(i) Is there any derivation $X \xrightarrow{u}{}^* Y$ satisfying $\gamma(w, C)$?
(ii) Is there any derivation $X \xrightarrow{u}{}^* t$ such that t is a term and $u \in \gamma(w, C)$?

Proof. Both problems can be reduced to the reachability problem for wPRS (i.e. to decide whether given states p_1t_1, p_2t_2 of a given wPRS system Δ' satisfy $p_1t_1 \xrightarrow{v}{}^*_{\Delta'} p_2t_2$ for some v), which is known to be decidable (Theorem 8.8).

(i) Let $w = a_1O_1 \dots a_nO_n$. We construct a wPRS Δ' with the set of control states $\{1, 2, \dots n\} \cup 2^C$. We use $(n+1)$ as another name for the control state $\emptyset$ (from 2^C). The set of rewrite rules is defined as follows.

- For every $1 \leq i \leq n$ and every rule $t_1 \overset{a_i}{\hookrightarrow} t_2$ of Δ, we add to Δ' the rule $it_1 \overset{a_i}{\hookrightarrow} (i+1)t_2$ and if $O_i = \mathsf{U}_+$ then also the rule $it_1 \overset{a_i}{\hookrightarrow} it_2$.
- For every $b \in C$, every $D \subseteq C$, and every rule $t_1 \overset{b}{\hookrightarrow} t_2$ of Δ, we add to Δ' the rule $Dt_1 \overset{b}{\hookrightarrow} (D \cup \{b\})t_2$.

Obviously, a word $u \in Act^*$ satisfies $1X \xrightarrow{u}{}^*_{\Delta'} CY$ if and only if it satisfies both $X \xrightarrow{u}{}^*_{\Delta} Y$ and $u \in \gamma(w, C)$. As we can decide whether $1X \xrightarrow{u}{}^*_{\Delta'} CY$ holds for some u, we can decide Problem (i).

(ii) We construct a wPRS Δ' as in the previous case. Moreover, for every $Z \in Const(\Delta)$ we add to Δ' the rule $CZ \overset{e}{\hookrightarrow} C\varepsilon$. It is easy to see that if a word $u \in \gamma(w, C)$ satisfies $X \xrightarrow{u}{}^*_{\Delta} t$ for some t, then $1X \xrightarrow{ue^m}{}^*_{\Delta'} C\varepsilon$ holds for some $m \geq 0$. Conversely, if $1X \xrightarrow{v}{}^*_{\Delta'} C\varepsilon$ holds for some v, then some prefix u of v satisfies both $u \in \gamma(w, C)$ and $X \xrightarrow{u}{}^*_{\Delta} t$ for some t. As we can decide whether $1X \xrightarrow{v}{}^*_{\Delta'} C\varepsilon$ holds for some v, we can decide Problem (ii).

□

The proof of the following lemma contains an algorithmic core of this section.

Lemma 10.17. *Let Δ be a PRS in normal form and $\beta(w, B, C)$ be a β-formula. One can construct a PRS Δ' in flat (w, B, C)-form such that, for each $(w', B', C') \sqsubseteq (w, B, C)$ and each $X \in Const(\Delta)$, is holds that*

in Δ' there is an infinite derivation starting from X and satisfying $\beta(w', B', C')$

if and only if

in Δ there is an infinite derivation starting from X and satisfying $\beta(w', B', C')$.

Proof. In order to obtain Δ', we describe an algorithm extending a PRS Δ with trivial rewrite rules according to Conditions 1–5 of Definition 10.14.

All the conditions of Definition 10.14 can be checked for each $X, Y \in Const(\Delta)$, each $(w', B', C') \sqsubseteq (w, B, C)$, and each $B'' \subseteq B$. For Conditions 1 and 2, this follows from Lemma 10.16. For Condition 3, a problem whether there is an infinite derivation $X \xrightarrow{u}{}^{\omega}$ satisfying $\beta(\varepsilon, B', C')$ is a special case of the *fairness problem* - see Proposition 10.15, which is decidable due to [Boz05]. Finally, Conditions 4 and 5 can be checked due to decidability of LTL model checking for PDA and PN.

Hence we can check if the conditions are satisfied. If there is a derivation which violates some of the conditions, we add some trivial rules to ensure the existence of a trivial derivation required by the respective condition. This process of adding new trivial rules is described in what follows.

Let us assume that Condition 3 (or 4 or 5) is not satisfied, i.e. there exists an infinite derivation $X \xrightarrow{u}{}^{\omega}$ (or $X \xrightarrow{u}{}^{\omega}_{par}$ or $X \xrightarrow{u}{}^{\omega}_{seq}$ respectively) satisfying $\beta(w', B', C')$ for some $(w', B', C') \sqsubseteq (w, B, C)$ and violating the condition. Remark 10.12 implies that C' is nonempty and there is a finite derivation $X \xrightarrow{v}{}^{*}_{\Delta} t$ satisfying $\gamma(w', B')$. Hence, there exists an ordering of $B' = \{b_1, b_2, \ldots, b_m\}$ such that

(*) for each $1 \leq j \leq m$, there is a finite derivation in Δ starting from X and satisfying $\gamma(w', \{b_1, \ldots, b_j\})$.

Such an ordering can be effectively computed using Lemma 10.16. Further, let $w' = a_1 O_1 a_2 O_2 \ldots a_n O_n$ and let $C' = \{c_1, c_2, \ldots, c_k\}$. Then, we add the trivial rule $Z_{i-1} \overset{a_i}{\hookrightarrow} Z_i$ for each $1 \leq i \leq n$, the trivial rule $Z_{n+j-1} \overset{b_j}{\hookrightarrow} Z_{n+j}$ for each $1 \leq j \leq m$, and the trivial rule $Z_{n+m+j-1} \overset{c_j}{\hookrightarrow} Z_{n+m+j}$ for each $1 \leq j \leq k$, where $Z_0 = X$, $Z_1, \ldots, Z_{n+m+k-1}$ are fresh process constants, and $Z_{n+m+k} = Z_{n+m}$. These added rules form an infinite derivation using only trivial rules, starting from X, and satisfying $\beta(w', B', C')$.

Similarly, if there are X, Y, and $\gamma(w', B'')$ with $w' = a_1 O_1 a_2 O_2 \ldots a_n O_n$ such that Condition 1 or 2 of Definition 10.14 is violated, then we first compute an ordering $\{b_1, \ldots, b_m\}$ of B'' satisfying (*), and then we add the trivial rule $Z_{i-1} \overset{a_i}{\hookrightarrow} Z_i$ for each $1 \leq i \leq n$, and the trivial rule $Z_{n+j-1} \overset{b_j}{\hookrightarrow} Z_{n+j}$

for each $1 \leq j \leq m$, where $Z_0 = X$ and $Z_1, \ldots, Z_{n+m}$ are fresh process constants (with exception of Z_{n+m} which is Y in the case of Condition 1). The added trivial rules generate derivation $X \stackrel{a_1 \ldots a_n b_1 \ldots b_m}{\longrightarrow^*} Z_{n+m}$ satisfying $\gamma(w', B'')$. Note that Conditions 1 and 2 are always satisfied if $n = m = 0$ as $\gamma(\varepsilon, \emptyset) = \{\varepsilon\}$.

Let Δ'' be the PRS Δ extended with the new rules. The condition (*) ensures that, for each $X \in Const(\Delta)$ and each $(w', B', C') \sqsubseteq (w, B, C)$, Δ'' is equivalent to Δ with respect to the existence of an infinite derivation starting from X and satisfying $\beta(w', B', C')$. If Δ'' is not in flat (w, B, C)-form, then the algorithm repeats the procedure described above on the system Δ'' with the difference that X and Y range over the constants of the original system Δ. The algorithm eventually terminates as the number of iterations is bounded by the number of pairs of process constants X, Y of Δ, times the number of triples (w', B', C') satisfying $(w', B', C') \sqsubseteq (w, B, C)$, and times the number of subsets $B'' \subseteq B$.

We claim that the resulting PRS Δ' is in flat (w, B, C)-form. By the construction, Δ' satisfies all conditions of Definition 10.14 for the process constants of the original system Δ. For the added constants, it is sufficient to observe that any derivation in Δ', starting from such a constant, is either trivial or it has a trivial prefix leading to a constant of Δ. Hence, Δ' is the desired PRS system. □

Definition 10.18 (Subderivation)**.** *Let Δ be a PRS in normal form and σ_1 be a (finite or infinite) derivation $s_1 \stackrel{a_1}{\longrightarrow} s_2 \stackrel{a_2}{\longrightarrow} \ldots$, where $s_1 \stackrel{a_1}{\longrightarrow} s_2$ has the form $X \stackrel{a_1}{\longrightarrow} Y.Z$ and, for each $i \geq 2$, if s_i is not the last state of the derivation, then it has the form $s_i = t_i.Z$ with $t_i \neq \varepsilon$. Then σ_1 is called a* subderivation *of a derivation σ if σ has a suffix σ' satisfying the following:*

1. *every transition step in σ' is of the form $s_i \| t' \stackrel{a_i}{\longrightarrow} s_{i+1} \| t'$ or $s_i \| t' \stackrel{b}{\longrightarrow} s_i \| t''$, where $t' \stackrel{b}{\longrightarrow} t''$,*
2. *in σ', if we replace every step of the form $s_i \| t' \stackrel{a_i}{\longrightarrow} s_{i+1} \| t'$ by step $s_i \stackrel{a_i}{\longrightarrow} s_{i+1}$ and we skip every step of the form $s_i \| t' \stackrel{b}{\longrightarrow} s_i \| t''$, we get precisely σ_1.*

Further, if σ_1 and σ are finite, the last term of σ_1 is a process constant, and σ is a prefix of a derivation σ', then σ_1 is also a subderivation *of σ'.*

Remark 10.19. *Let Δ be a PRS in normal form and σ be a derivation of Δ having a suffix σ' of the form $\sigma' = X \| t \stackrel{a}{\longrightarrow} (Y.Z) \| t \stackrel{u}{\longrightarrow^\omega}$. Then, there is a subderivation of σ whose first transition step $X \stackrel{a}{\longrightarrow} Y.Z$ corresponds to the first transition step of σ'.*

Intuitively, a subderivation captures the behaviour of the subterm $Y.Z$ since its emergence until its eventual reduction to a term without any se-

quential composition. Due to the normal form of Δ, the subterm $Y.Z$ behaves independently on the rest of the term (as long as it contains a sequential composition).

Lemma 10.20. *Let Δ be a PRS in flat (w, B, C)-form. Then, the following condition holds for each $X \in Const(\Delta)$ and each $(w', B', C') \sqsubseteq (w, B, C)$:*
If there is an infinite derivation $X \xrightarrow{u} \omega$ satisfying $\beta(w', B', C')$, then there is also an infinite derivation $X \xrightarrow{v}_{tri} \omega$ satisfying $\beta(w', B', C')$.

Proof. In the following two paragraphs, we provide a sketch of the proof. The full proof follows.

Given an infinite derivation σ satisfying a formula $\beta(\sigma) = \beta(w', B', C')$ where $(w', B', C') \sqsubseteq (w, B, C)$, by *trivial equivalent* of σ we mean an infinite trivial derivation starting in the same term as σ and satisfying $\beta(\sigma)$. Similarly, given a finite derivation σ satisfying some $\gamma(\sigma) = \gamma(w', B')$ where $(w', B') \sqsubseteq (w, B)$, by *trivial equivalent* of σ we mean a finite trivial derivation σ' such that σ' starts in the same term as σ, it satisfies $\gamma(\sigma)$, and if the last term of σ is a process constant, then the last term of σ' is the same process constant.

The lemma is proven by contradiction. We assume that there exist some infinite derivations violating the condition of the lemma. Let σ be one of these derivations such that the number of transition steps of σ generated by sequential non-trivial rules with actions $a \notin B$ is minimal (note that this number is always finite as we consider derivations satisfying $\beta(w', B', C')$ for some $(w', B', C') \sqsubseteq (w, B, C)$). First, we prove that every subderivation of σ has a trivial equivalent. Then we replace all subderivations of σ by the corresponding trivial equivalents. This step is technically nontrivial because σ may have infinitely many subderivations. By the replacement we obtain an infinite derivation σ' satisfying $\beta(\sigma)$ and starting in the same process constant as σ. Moreover, σ' has no subderivations and hence it does not contain any sequential operator. Flat (w, B, C)-form of Δ (Condition 4) implies that σ' has a trivial equivalent. This is also a trivial equivalent of σ which means that σ does not violate the condition of our lemma.

In this proof, by β-formula we always mean a formula of the form $\beta(w', B', C')$ where $(w', B', C') \sqsubseteq (w, B, C)$. We also consider only infinite derivations satisfying some of these β-formulae. Remark 10.13 implies that such an infinite derivation σ satisfies exactly one β-formula. We denote this β-formula by $\beta(\sigma)$. Further, by $SEQ(\sigma)$ we denote the number of transition steps $t_i \xrightarrow{a} t_{i+1}$ of σ generated by a sequential non-trivial rule and such that $a \notin B$. Note that $SEQ(\sigma)$ is always finite due to the restrictions on considered infinite derivations. Given an infinite derivation σ, by its *trivial equivalent* we mean an infinite trivial derivation starting in the same term as σ and satisfying $\beta(\sigma)$.

Similarly, we consider only finite derivations satisfying some $\gamma(w', B')$ where $(w', B') \sqsubseteq (w, B)$. Remark 10.13 implies that such a finite derivation σ satisfies exactly one γ-language, which is denoted by $\gamma(\sigma)$. Given a finite derivation σ, by its *trivial equivalent* we mean a finite trivial derivation σ' such that σ' starts in the same term as σ, it satisfies $\gamma(\sigma)$, and if the last term of σ is a process constant, then the last term of σ' is the same process constant.

Using the introduced terminology, the lemma says that every infinite derivation starting in a process constant has a trivial equivalent. For the sake of contradiction, we assume that the lemma does not hold. Let Σ be the set of infinite derivations violating the lemma and let $k = \min\{SEQ(\sigma) \mid \sigma \in \Sigma\}$.

First of all, we prove two claims.

Claim 1. Let σ be an infinite derivation satisfying $SEQ(\sigma) \leq k$. Then every subderivation of σ has a trivial equivalent.

Proof of the claim: For finite subderivations, the existence of trivial equivalents follows directly from the flat (w, B, C)-form of Δ (Conditions 1 and 2). Let σ_1 be an infinite subderivation of σ. It has the form

$$\sigma_1 = X \xrightarrow{a}_{seq} Y.Z \xrightarrow{b_1} t_1.Z \xrightarrow{b_2} t_2.Z \xrightarrow{b_3} \dots$$

where $t_1, t_2, \dots$ are nonempty terms. There are two cases:

- If $a \in B$, then $\beta(\sigma_1)$ has the form $\beta(\varepsilon, B', C')$. Hence, σ_1 has a trivial equivalent due to the flat (w, B, C)-form of Δ (Condition 3).

- If $a \notin B$, then the first step $X \xrightarrow{a}_{seq} Y.Z$ of σ_1 is counted in $SEQ(\sigma_1)$ and the corresponding step $X \| t' \xrightarrow{a}_{seq} Y.Z \| t'$ of σ is counted in $SEQ(\sigma)$. Hence, $0 < SEQ(\sigma)$. Let σ_2 be the derivation

  $$\sigma_2 = Y \xrightarrow{b_1} t_1 \xrightarrow{b_2} t_2 \xrightarrow{b_3} \dots$$

  As $SEQ(\sigma_2) < SEQ(\sigma_1) \leq k$, the definition of k implies that σ_2 has a trivial equivalent

  $$\sigma_2' = Y \xrightarrow{c_1}_{tri} Y_1 \xrightarrow{c_2}_{tri} Y_2 \xrightarrow{c_3}_{tri} .$$

  Further, as σ_2' satisfies $\beta(\sigma_2)$, the derivation

  $$\sigma_1' = X \xrightarrow{a}_{seq} Y.Z \xrightarrow{c_1}_{tri} Y_1.Z \xrightarrow{c_2}_{tri} Y_2.Z \xrightarrow{c_3}_{tri} \dots$$

  satisfies $\beta(\sigma_1)$. Moreover, the flat (w, B, C)-form of Δ (Condition 5) implies that σ_1' has a trivial equivalent. Obviously, it is also a trivial equivalent of σ_1. □

Claim 2. Let σ be an infinite derivation such that $SEQ(\sigma) \leq k$, it starts in a parallel term p, and it satisfies a formula $\beta(w', B', C')$. Then there is an infinite derivation $p \xrightarrow{u}{}^*_{par} p' \xrightarrow{v}{}^\omega$ such that p' is a parallel term, $u \in \gamma(w', B')$, and v satisfies $\beta(\varepsilon, C', C')$.

Proof of the claim: Remark 10.12 implies that σ can be written as $p \xrightarrow{u_1}{}^* t \xrightarrow{u_2}{}^\omega$ where $p \xrightarrow{u_1}{}^* t$ is the *minimal* prefix of σ satisfying $\gamma(w', B')$ and such that $t \xrightarrow{u_2}{}^\omega$ satisfies $\beta(\varepsilon, C', C')$. Let $\widetilde{SEQ}(\sigma)$ denote the number of transition steps in the prefix $p \xrightarrow{u_1}{}^* t$ generated by sequential non-trivial rules (note that $\widetilde{SEQ}(\sigma) \geq SEQ(\sigma)$). We prove the claim by induction on $\widetilde{SEQ}(\sigma)$. The base case $\widetilde{SEQ}(\sigma) = 0$ is obvious. Now, assume that $\widetilde{SEQ}(\sigma) > 0$. Since p is parallel term and Δ is in normal form, the first transition step of $p \xrightarrow{u_1}{}^* t$ counted in $\widetilde{SEQ}(\sigma)$ has the form $Y \| p' \xrightarrow{a} (W.Z) \| p'$ and it corresponds to the first transition step $Y \xrightarrow{a} W.Z$ of a subderivation σ_1. In σ, we replace the subderivation σ_1 with its trivial equivalent (whose existence is guaranteed by Claim 1) and we obtain a new derivation σ'' starting from p, satisfying $\beta(\sigma)$ and such that $\widetilde{SEQ}(\sigma'') < \widetilde{SEQ}(\sigma)$. Hence, the second claim directly follows from the induction hypothesis. In the following, we describe the replacement of such a subderivation.

Let $\sigma_1 = Y \xrightarrow{u}{}^\omega$ and $\sigma_1' = Y \xrightarrow{v}{}^\omega_{tri}$ be its trivial equivalent. Let $\beta(\sigma_1) = \beta(c_1 O_1 c_2 O_2 \dots c_n O_n, B'', C'')$. Then $u, v \in c_1^+ c_2^+ \dots c_n^+ . B^\omega$. Recall that $c_1, c_2, \dots, c_n$ are pairwise distinct and $B \subseteq Act \setminus \{c_1, \dots, c_n\}$. Intuitively, for every $1 \leq i \leq n$, we replace the first transition step of σ_1 labelled with c_i by the sequence of transition steps of σ_1' labelled with c_i, and then we cancel the other transition steps of σ_1 labelled with c_i.[5] Further, the first transition step of σ_1 labelled with an action of B is replaced with the minimal prefix of the remaining part of σ_1' satisfying $\gamma(\varepsilon, B'')$. Finally, the remaining transition steps of σ_1 are orderly replaced with the remaining transition steps of σ_1'. The case when σ_1 and its trivial equivalent σ_1' are finite is similar.

It is easy to see that the described replacement operation preserves the fulfilment of $\beta(\sigma)$ and the obtained derivation σ'' satisfies $\widetilde{SEQ}(\sigma'') < \widetilde{SEQ}(\sigma)$. □

With this claim, we can easily reach a contradiction. Let $\sigma = X \xrightarrow{u}{}^\omega$ be an infinite derivation such that $SEQ(\sigma) = k$ and it has no trivial equivalent.

[5] By replacement of a transition step $s_1 \xrightarrow{a} s_2$ of σ_1 by a sequence $Y_1 \xrightarrow{v'}{}^*_{tri} Y_2$ of transition steps of σ_1' we mean that the corresponding transition step $s_1 \| t' \xrightarrow{a} s_2 \| t'$ of σ is replaced by $Y_1 \| t' \xrightarrow{v'}{}^*_{tri} Y_2 \| t'$, and all immediately succeeding steps $s_2 \| t'' \xrightarrow{b} s_2 \| t'''$ of σ are replaced by $Y_2 \| t'' \xrightarrow{b} Y_2 \| t'''$. Further, by cancellation of a transition step $s_1 \xrightarrow{c_i} s_2$ of σ_1 we mean that the corresponding transition step $s_1 \| t' \xrightarrow{c_i} s_2 \| t'$ of σ is replaced by $Y_2 \| t'$, where Y_2 is the last process constant of σ_1' such that a transition under c_i leads to Y_2, and all immediately succeeding steps $s_2 \| t'' \xrightarrow{b} s_2 \| t'''$ of σ are replaced by $Y_2 \| t'' \xrightarrow{b} Y_2 \| t'''$.

Further, let $\beta(\sigma) = (w', B', C')$. Note that C' is nonempty. Claim 2 says that there is a derivation $X \xrightarrow{u_1}{}^*_{par}\, p_1 \xrightarrow{v_1}{}^{\omega}$ where p_1 is a parallel term, $u_1 \in \gamma(w', B')$, and v_1 satisfies $\beta(\varepsilon, C', C')$. Applying the second claim on the suffix $p_1 \xrightarrow{v_1}{}^{\omega}$, we get a derivation $p_1 \xrightarrow{u_2}{}^*_{par}\, p_2 \xrightarrow{v_2}{}^{\omega}$ where p_2 is a parallel term, $u_2 \in \gamma(\varepsilon, C')$, and v_2 satisfies $\beta(\varepsilon, C', C')$. Iterating this argument, we get a sequence $(p_i \xrightarrow{u_{i+1}}{}^*_{par}\, p_{i+1})_{i\in\mathbb{N}}$ of derivations satisfying $\gamma(\varepsilon, C')$. These derivations are nonempty as C' is nonempty. Let us consider the derivation

$$\sigma' = X \xrightarrow{u_1}{}^*_{par}\, p_1 \xrightarrow{u_2}{}^*_{par}\, p_2 \xrightarrow{u_3}{}^*_{par}\, p_3 \xrightarrow{u_4}{}^*_{par}\, \cdots$$

Flat (w, B, C)-form of Δ (Condition 4) implies that σ' has a trivial equivalent. However, this is also a trivial equivalent of σ as both σ, σ' start with X and σ' satisfies $\beta(\sigma)$. This is a contradiction. □

Theorem 10.21. *The problem whether a given PRS Δ in normal form has an infinite run satisfying a given formula $\beta(w, B, C)$ is decidable.*

Proof. Due to Lemma 10.17 and Lemma 10.20, the problem can be reduced to the problem whether there is an infinite derivation $X \xrightarrow{v}{}^{\omega}_{tri}$ satisfying $\beta(w, B, C)$. This problem corresponds to LTL model checking of finite-state systems, which is decidable. □

The following three theorems show that Theorem 10.21 holds even for wPRS and α-formulae.

Theorem 10.22. *The problem whether a given PRS Δ in normal form has an infinite run satisfying a given α-formula is decidable.*

Proof. Let Δ be a PRS in normal form and $\alpha(\theta_1 O_1 \dots \theta_n O_n \xi, \mathcal{B})$ be an α-formula. For every θ_i and every rule $t_1 \overset{b}{\hookrightarrow} t_2$ such that b satisfies θ_i, we add a rule $t_1 \overset{a_i}{\hookrightarrow} t_2$, where a_i is a fresh action corresponding to θ_i. Similarly, for every $\psi \in \mathcal{B} \cup \{\xi\}$ and every rule $t_1 \overset{b}{\hookrightarrow} t_2$ such that b satisfies $\psi \wedge \xi$, we add a rule $t_1 \overset{a_\psi}{\hookrightarrow} t_2$, where a_ψ is a fresh action. Let Δ' be the resulting PRS system. Note that Δ' is also in normal form. Obviously, Δ has an infinite run satisfying the original α-formula if and only if Δ' has an infinite run satisfying $\alpha(a_1 O_1 \dots a_n O_n (a_\xi \vee \bigvee_{b\in C} b), C)$, where $C = \{a_\psi \mid \psi \in \mathcal{B}\}$. It is an easy exercise to show that this new α-formula can be effectively transformed into a disjunction of β-formulae which is equivalent with respect to infinite words. Hence, the problem is decidable due to Theorem 10.21. □

Theorem 10.23. *The problem whether a given PRS Δ has an infinite run satisfying a given α-formula is decidable.*

Proof. Let Δ be a PRS, $\alpha(\delta, \mathcal{B})$ be an α-formula, and $e \notin Act(\Delta)$ be a fresh action. First we describe our modification of the standard algorithm [May00] that transforms Δ into a PRS in normal form.

If Δ is not in normal form, then there exists a rule r which is neither parallel nor sequential; r has one of the following forms:

1. $r = t \overset{a}{\hookrightarrow} t_1 \| t_2$ (resp., $r = t_1 \| t_2 \overset{a}{\hookrightarrow} t$) where t or t_1 or t_2 is not a parallel term. Let $Z_1, Z_2, Z \notin Const(\Delta)$ be fresh process constants. We replace r with the rules $t \overset{e}{\hookrightarrow} Z$, $Z \overset{a}{\hookrightarrow} Z_1 \| Z_2$, $Z_1 \overset{e}{\hookrightarrow} t_1$, and $Z_2 \overset{e}{\hookrightarrow} t_2$ (resp., $t_1 \overset{e}{\hookrightarrow} Z_1$, $t_2 \overset{e}{\hookrightarrow} Z_2$, $Z_1 \| Z_2 \overset{a}{\hookrightarrow} Z$, and $Z \overset{e}{\hookrightarrow} t$).

2. $r = t \overset{a}{\hookrightarrow} t_1.(t_2 \| t_3)$ (resp., $r = t_1.(t_2 \| t_3) \overset{a}{\hookrightarrow} t$). Let $Z \notin Const(\Delta)$ be a fresh process constant. We modify Δ in two steps. First, we replace $t_2 \| t_3$ by Z in left-hand and right-hand sides of all rules of Δ. Then, we add the rules $Z \overset{e}{\hookrightarrow} t_2 \| t_3$ and $t_2 \| t_3 \overset{e}{\hookrightarrow} Z$.

3. $r = t_1 \overset{a}{\hookrightarrow} t_2.X$ (resp., $r = t_2.X \overset{a}{\hookrightarrow} t_1$) where t_1 or t_2 is not a process constant. Let $Z_1, Z_2 \notin Const(\Delta)$ be fresh process constants. We replace r with the rules $t_1 \overset{e}{\hookrightarrow} Z_1$, $Z_1 \overset{a}{\hookrightarrow} Z_2.X$, and $Z_2 \overset{e}{\hookrightarrow} t_2$ (resp., $t_2 \overset{e}{\hookrightarrow} Z_2$, $Z_2.X \overset{a}{\hookrightarrow} Z_1$, and $Z_1 \overset{e}{\hookrightarrow} t_1$).

After a finite number of applications of this procedure (with the same action e), we obtain a PRS Δ' in normal form.

We define a formula $\alpha(\delta', \mathcal{B}')$, where $\mathcal{B}' = \mathcal{B} \cup \{\bigvee_{a \in Act(\Delta)} a\}$ and δ' arises from $\delta = \theta_1 O_1 \dots \theta_n O_n \xi$ by the following substitution for every $1 \leq i \leq n$.

- If O_i is U, then replace the pair $\theta_i \, \mathsf{U}$ by the pair $(e \vee \theta_i) \, \mathsf{U}$.
- If O_i is U_+, then replace the pair $\theta_i \, \mathsf{U}_+$ by the sequence $(e \vee \theta_i) \, \mathsf{U} \, \theta_i \, \mathsf{U}_+$.
- If O_i is $\wedge \mathsf{X}$, then replace the pair $\theta_i \wedge \mathsf{X}$ by the sequence $e \, \mathsf{U} \, \theta_i \wedge \mathsf{X}$.
- $\theta_n O_n = \theta_n \wedge \mathsf{G_s}$ is replaced by the sequence $e \, \mathsf{U} \, \theta_n \wedge \mathsf{G_s}$.
- ξ is replaced by $(\xi \vee e)$.

Let us note that the construction of $\mathcal{B}'$ ensures that any word with a suffix e^ω does not satisfy $\alpha(\delta', \mathcal{B}')$. Observe that $u' \models \alpha(\delta', \mathcal{B}')$ if and only if $u \models \alpha(\delta, \mathcal{B})$, where u is obtained from u' by eliminating all occurrences of action e.

Clearly, Δ has an infinite run satisfying $\alpha(\delta, \mathcal{B})$ if and only if Δ' has an infinite run satisfying $\alpha(\delta', \mathcal{B}')$. As Δ' is in normal form, we can now apply Theorem 10.22. □

Theorem 10.24. *The problem whether a given wPRS system has an infinite run satisfying a given α-formula is decidable.*

Proof. Let Δ be a wPRS with initial state p_0X_0 and $\alpha(\delta, \mathcal{B})$ be an α-formula. We construct a PRS Δ' with initial state X_0 which can simulate Δ. We also define set of formulae recognising correct simulations.

The system Δ' is very similar to Δ. We only change actions of rules to hold information about control states in the rules and then we remove all control states. More precisely, for every rule of the form $pt_1 \overset{a}{\hookrightarrow} pt_2$ of Δ we add to Δ' the rule $t_1 \overset{a_{[p]}}{\hookrightarrow} t_2$, and for every rule of the form $pt_1 \overset{a}{\hookrightarrow} qt_2$ of Δ we add to Δ' the rule $t_1 \overset{a_{[p<q]}}{\hookrightarrow} t_2$.

Further, we modify the formula $\alpha(\delta, \mathcal{B})$ such that every occurrence of each action a is replaced by $\bigvee_{q \in M(\Delta)}(a_{[q]} \vee \bigvee_{p<q} a_{[p<q]})$. Let $\alpha(\delta', \mathcal{B}')$ be the resulting formula.

Moreover, for every sequence $p_1 < p_2 < \ldots < p_k$ of control states of $M(\Delta)$ such that $p_1 = p_0$, we define an α-formula

$$\varphi_{[p_1<p_2<\ldots<p_k]} = \alpha(\theta_{[p_1]} \,\mathsf{U}\, \theta_{[p_1<p_2]} \wedge \mathsf{X}\, \theta_{[p_2]} \,\mathsf{U}\, \theta_{[p_2<p_3]} \wedge \mathsf{X} \ldots \theta_{[p_{k-1}<p_k]} \wedge \mathsf{G_s}\theta_{[p_k]}, \emptyset)$$

where $\theta_{[p_i]} = \bigvee_{a \in Act(\Delta)} a_{[p_i]}$ and $\theta_{[p_i<p_j]} = \bigvee_{a \in Act(\Delta)} a_{[p_i<p_j]}$.

It is easy to see that there is an infinite run of Δ satisfying $\alpha(\delta, \mathcal{B})$ if and only if there is an infinite run of Δ' satisfying $\alpha(\delta', \mathcal{B}')$ and $\varphi_{[p_1<p_2<\ldots<p_k]}$ for some sequence $p_1 < p_2 < \ldots < p_k$. As the number of such sequences is finite and each $\varphi_{[p_1<p_2<\ldots<p_k]}$ is an α-formula, Theorem 10.23 and Lemma 10.9 imply that the considered problem is decidable. □

As LTL($\mathsf{F_s}, \mathsf{G_s}$) is closed under negation, Theorem 10.10, Theorem 10.11, and Theorem 10.24 give us the following.

Corollary 10.25. *The model checking problem for wPRS and LTL($\mathsf{F_s}, \mathsf{G_s}$) is decidable.*

This problem is EXPSPACE-hard due to EXPSPACE-hardness of the model checking problem for LTL(F, G) for PN [Hab97]. Our decidability proof does not provide any primitive recursive upper bound as it employs LTL model checking for PN, for which no primitive recursive upper bound is known.

10.5 Model Checking LTL($\mathsf{F_s}, \mathsf{P_s}$)

In this section we prove decidability of the model checking problem for wPRS and LTL($\mathsf{F_s}, \mathsf{P_s}$). We introduce a new LTL fragment $\mathcal{PA}$ and prove that every formula of the basic fragment LTL($\mathsf{F_s}, \mathsf{P_s}$) can be effectively translated into $\mathcal{PA}$. In other words, we show that for each LTL($\mathsf{F_s}, \mathsf{P_s}$) formula one can find a globally equivalent formula of $\mathcal{PA}$.

Recall that LTL() denotes the fragment of formulae without any modality, i.e. boolean combinations of actions. In the following we keep using

$\varphi_1 \mathsf{U}_+ \varphi_2$ to abbreviate $\varphi_1 \wedge \mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)$ and, moreover, we use $\varphi_1 \mathsf{S}_+ \varphi_2$ to abbreviate $\varphi_1 \wedge \mathsf{Y}(\varphi_1 \mathsf{S} \varphi_2)$. Now, we can define a past variant of the $\mathcal{A}$ fragment called $\mathcal{PA}$.

Definition 10.26. *Let $\eta = \iota_1 P_1 \iota_2 P_2 \dots \iota_m P_m \iota_{m+1}$, where*

- $m > 0$,
- $\iota_j \in LTL()$ *for each* $j \leq m+1$,
- P_j *is either '*S*' or '*S_+*' or '*$\wedge \mathsf{Y}$*' for each* $j < m$, *and*
- P_m *is '*$\wedge \mathsf{H}_\mathsf{s}$*'.*

Similarly, let $\delta = \theta_1 O_1 \theta_2 O_2 \dots \theta_n O_n \theta_{n+1}$, where

- $n > 0$,
- $\theta_i \in LTL()$ *for each* $i \leq n+1$,
- O_i *is either '*U*' or '*U_+*' or '*$\wedge \mathsf{X}$*' for each* $i < n$, *and*
- O_n *is '*$\wedge \mathsf{G}_\mathsf{s}$*'.*

Further, let $\mathcal{B} \subseteq LTL()$ be a finite set. A $P\alpha$-formula *is defined as*

$$P\alpha(\eta, \delta, \mathcal{B}) = \big(\iota_1 P_1 (\iota_2 P_2 \dots (\iota_m P_m \iota_{m+1}) \dots)\big) \wedge \\ \wedge \big(\theta_1 O_1 (\theta_2 O_2 \dots (\theta_n O_n \theta_{n+1}) \dots)\big) \wedge \bigwedge_{\psi \in \mathcal{B}} \mathsf{G}_\mathsf{s}\mathsf{F}_\mathsf{s}\psi$$

The $\mathcal{PA}$ fragment *consists of finite disjunctions of $P\alpha$-formulae.*

Intuitively, $P\alpha$-formula $P\alpha(\eta, \delta, \mathcal{B})$ is an $\alpha(\delta, \mathcal{B})$ extended with a new conjunct that describes a past part. Hence, η is a past counterpart of δ. There is no past counter part to $\wedge_{\psi \in \mathcal{B}} \mathsf{G}_\mathsf{s}\mathsf{F}_\mathsf{s}\psi$ as every history is finite — it begin in the initial state.

Therefore, a pointed word (u, k) satisfies $P\alpha(\eta, \delta, \mathcal{B})$ if and only if (u, k) satisfies $\alpha(\delta, \mathcal{B})$ and $u(0) \dots u(k-1)u(k)$ can be written as

$$v_{m+1}.v_{m-1}.\cdots.v_2.v_1$$

where each v_i, for $i = 1, \dots, m+1$, consists only of actions satisfying ι_i and

- $|v_i| \geq 0$ if $i = m+1$ or P_i is 'S',
- $|v_i| > 0$ if P_i is 'S_+',
- $|v_i| = 1$ if P_i is '$\wedge \mathsf{Y}$' or '$\wedge \mathsf{H}_\mathsf{s}$'.

In the following lemmata we provide a list of operation under which the $\mathcal{PA}$ fragment is closed. Proofs of the lemmata are easy but sometimes very technical exercises, hence we sketch their basic ideas only.

Lemma 10.27. *A conjunction of Pα-formulae can be effectively converted into a globally equivalent disjunction of Pα-formulae.*

The proof is constructed in a similar way as the proof of Lemma 10.9.

Lemma 10.28. *Let φ be a Pα-formula. A formula $\mathsf{X}\varphi$ can be effectively converted into a globally equivalent disjunction of Pα-formulae.*

Proof (sketch). We go through all possibilities reflecting how to bite one "step" out of η and append it to the beginning of δ using $\wedge\mathsf{X}$. □

Lemma 10.29. *Let φ be a Pα-formula. A formula $\mathsf{Y}\varphi$ can be effectively converted into a globally equivalent disjunction of Pα-formulae.*

The proof is constructed in a similar way as the proof of Lemma 10.28.

Lemma 10.30. *Let φ be a Pα-formula and $p \in LTL()$. A formula $p\,\mathsf{U}\,\varphi$ can be effectively converted into a globally equivalent disjunction of Pα-formulae.*

Proof (sketch). Contrary to the proof of Lemma 10.28, an arbitrary number of "steps" can be "bitten" out of η here. Hence, the bitten parts are not connected only by $\wedge\mathsf{X}$ but reflect the operators in the bitten part. Moreover, all bitten ι formulae of η are extended with $\wedge p$ before appending them to δ. □

Lemma 10.31. *Let φ be a Pα-formula and $p \in LTL()$. A formula $p\,\mathsf{S}\,\varphi$ can be effectively converted into a globally equivalent disjunction of Pα-formulae.*

The proof is constructed in a similar way as the proof of Lemma 10.30.

Lemma 10.32. *Let φ be a Pα-formula. A formula $\mathsf{F_s}\varphi$ can be effectively converted into a globally equivalent disjunction of Pα-formulae.*

Proof. As $\mathsf{F_s}\varphi \equiv_g \mathsf{X}(true\,\mathsf{U}\,\varphi)$, the proof directly follows from Lemma 10.30 and Lemma 10.28. □

Lemma 10.33. *Let φ be a Pα-formula. A formula $\mathsf{P_s}(\varphi)$ can be effectively converted into a globally equivalent disjunction of Pα-formulae.*

Proof. As $\mathsf{P_s}\varphi \equiv_g \mathsf{Y}(true\,\mathsf{S}\,\varphi)$, the proof directly follows from Lemma 10.31 and Lemma 10.29. □

Theorem 10.34. *Every LTL($\mathsf{F_s}, \mathsf{P_s}$) formula can be translated into a globally equivalent disjunction of Pα-formulae.*

Proof. As $\mathsf{F_s}, \mathsf{G_s}$ and $\mathsf{P_s}, \mathsf{H_s}$ are dual modalities, we can assume that every LTL($\mathsf{F_s}, \mathsf{P_s}$) formula contains negations only in front of actions. Given an LTL($\mathsf{F_s}, \mathsf{P_s}$) formula φ, we construct a finite set A_φ of Pα-formulae such that φ is globally equivalent to disjunction of formulae in A_φ. Again, our proof

looks like a proof by induction on the structure of φ, however it is done by induction on the length of φ. Thus, if $\varphi \notin \text{LTL}()$, then we assume that for every LTL($\mathsf{F_s}, \mathsf{P_s}$) formula φ' shorter than φ we can construct the corresponding set $A_{\varphi'}$. In this proof, let p represent a formula of LTL(). The structure of φ fits into one of the following cases.

- $\bullet p$ **Case** p**:** In this case, φ is equivalent to $p \wedge \mathsf{G_s}tt$. Hence $A_\varphi = \{P\alpha(tt \wedge \mathsf{H_s}tt, p \wedge \mathsf{G_s}tt, \emptyset)\}$.
- $\bullet\vee$ **Case** $\varphi_1 \vee \varphi_2$**:** Due to induction hypothesis, we can assume that we have sets A_{φ_1} and A_{φ_2}. Clearly, $A_\varphi = A_{\varphi_1} \cup A_{\varphi_2}$.
- $\bullet\wedge$ **Case** $\varphi_1 \wedge \varphi_2$**:** Due to Lemma 10.27, the set A_φ can be constructed from the sets A_{φ_1} and A_{φ_2}.
- $\bullet\mathsf{F_s}$ **Case** $\mathsf{F_s}\varphi_1$**:** Due to Lemma 10.32, the set A_φ can be constructed from the sets A_{φ_1}.
- $\bullet\mathsf{P_s}$ **Case** $\mathsf{P_s}\varphi_1$**:** Due to Lemma 10.33, the set A_φ can be constructed from the sets A_{φ_1}.
- $\bullet\mathsf{G_s}$ **Case** $\mathsf{G_s}\varphi_1$**:** This case is divided into the following subcases according to the structure of φ_1.
  - $\circ p$ **Case** $\mathsf{G_s}p$**:** As $\mathsf{G_s}p$ is equivalent to $tt \wedge \mathsf{G_s}p$, we set $A_\varphi = \{P\alpha(tt \wedge \mathsf{H_s}tt, tt \wedge \mathsf{G_s}p, \emptyset)\}$.
  - $\circ\wedge$ **Case** $\mathsf{G_s}(\varphi_2 \wedge \varphi_3)$**:** As $\mathsf{G_s}(\varphi_2 \wedge \varphi_3) \equiv (\mathsf{G_s}\varphi_2) \wedge (\mathsf{G_s}\varphi_3)$, the set A_φ can be constructed from $A_{\mathsf{G_s}\varphi_2}$ and $A_{\mathsf{G_s}\varphi_3}$ using Lemma 10.27. Note that $A_{\mathsf{G_s}\varphi_2}$ and $A_{\mathsf{G_s}\varphi_3}$ can be constructed because $\mathsf{G_s}\varphi_2$ and $\mathsf{G_s}\varphi_3$ are shorter than $\mathsf{G_s}(\varphi_2 \wedge \varphi_3)$.
  - $\circ\mathsf{F_s}$ **Case** $\mathsf{G_s}\mathsf{F_s}\varphi_2$**:** This case is again divided into the following subcases.
    - $-p$ **Case** $\mathsf{G_s}\mathsf{F_s}p$**:** As $p \in \text{LTL}()$, we directly set $A_\varphi = \{P\alpha(tt \wedge \mathsf{H_s}tt, tt \wedge \mathsf{G_s}tt, \{p\})\}$.
    - $-\vee$ **Case** $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \vee \varphi_4)$**:** As $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \vee \varphi_4) \equiv (\mathsf{G_s}\mathsf{F_s}\varphi_3) \vee (\mathsf{G_s}\mathsf{F_s}\varphi_4)$, we set $A_\varphi = A_{\mathsf{G_s}\mathsf{F_s}\varphi_3} \cup A_{\mathsf{G_s}\mathsf{F_s}\varphi_4}$.
    - $-\wedge$ **Case** $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \varphi_4)$**:** This case is also divided into subcases depending on the formulae φ_3 and φ_4.
      - $*p$ **Case** $\mathsf{G_s}\mathsf{F_s}(p_3 \wedge p_4)$**:** As $p_3 \wedge p_4 \in \text{LTL}()$, this subcase has already been covered by Case $\mathsf{G_s}\mathsf{F_s}p$.
      - $*\vee$ **Case** $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge (\varphi_5 \vee \varphi_6))$**:** As $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge (\varphi_5 \vee \varphi_6)) \equiv \mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \varphi_5) \vee \mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \varphi_6)$, we set $A_\varphi = A_{\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \varphi_5)} \cup A_{\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \varphi_6)}$.
      - $*\mathsf{F_s}$ **Case** $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \mathsf{F_s}\varphi_5)$**:** As $\mathsf{G_s}\mathsf{F_s}(\varphi_3 \wedge \mathsf{F_s}\varphi_5) \equiv (\mathsf{G_s}\mathsf{F_s}\varphi_3) \wedge (\mathsf{G_s}\mathsf{F_s}\varphi_5)$, the set A_φ can be constructed from $A_{\mathsf{G_s}\mathsf{F_s}\varphi_3}$ and $A_{\mathsf{G_s}\mathsf{F_s}\varphi_5}$ using Lemma 10.27.

$*\mathsf{P_s}$ **Case** $\mathsf{G_s F_s}(\varphi_3 \wedge \mathsf{P_s}\varphi_5)$**:** As $\mathsf{G_s F_s}(\varphi_3 \wedge \mathsf{P_s}\varphi_5) \equiv (\mathsf{G_s F_s}\varphi_3) \wedge (\mathsf{G_s F_s P_s}\varphi_5)$, the set A_φ can be constructed from $A_{\mathsf{G_s F_s}\varphi_3}$ and $A_{\mathsf{G_s F_s P_s}\varphi_5}$ using Lemma 10.27.

$*\mathsf{G_s}$ **Case** $\mathsf{G_s F_s}(\varphi_3 \wedge \mathsf{G_s}\varphi_5)$**:** As $\mathsf{G_s F_s}(\varphi_3 \wedge \mathsf{G_s}\varphi_5) \equiv (\mathsf{G_s F_s}\varphi_3) \wedge (\mathsf{G_s F_s G_s}\varphi_5)$, the set A_φ can be constructed from $A_{\mathsf{G_s F_s}\varphi_3}$ and $A_{\mathsf{G_s F_s G_s}\varphi_5}$ using Lemma 10.27.

$*\mathsf{H_s}$ **Case** $\mathsf{G_s F_s}(\varphi_3 \wedge \mathsf{H_s}\varphi_5)$**:** As $\mathsf{G_s F_s}(\varphi_3 \wedge \mathsf{H_s}\varphi_5) \equiv (\mathsf{G_s F_s}\varphi_3) \wedge (\mathsf{G_s F_s H_s}\varphi_5)$, the set A_φ can be constructed from $A_{\mathsf{G_s F_s}\varphi_3}$ and $A_{\mathsf{G_s F_s H_s}\varphi_5}$ using Lemma 10.27.

$-\mathsf{F_s}$ **Case** $\mathsf{G_s F_s F_s}\varphi_3$**:** As $\mathsf{G_s F_s F_s}\varphi_3 \equiv \mathsf{G_s F_s}\varphi_3$, we set $A_\varphi = A_{\mathsf{G_s F_s}\varphi_3}$.

$-\mathsf{P_s}$ **Case** $\mathsf{G_s F_s P_s}\varphi_3$**:** A pointed word (u, i) satisfies $\mathsf{G_s F_s P_s}\varphi_3$ iff $i = |u| - 1$ or u is an infinite word satisfying $\mathsf{F}\varphi_3$. Note that $\mathsf{G_s}\neg tt$ is satisfied only by finite words at their last position. Further, a word u satisfies $(\mathsf{F_s}tt) \wedge (\mathsf{G_s F_s}tt)$ iff u is infinite. Thus, $\mathsf{G_s F_s P_s}\varphi_3 \equiv (\mathsf{G_s}\neg tt) \vee \varphi'$ where $\varphi' = (\mathsf{F_s}tt) \wedge (\mathsf{G_s F_s}tt) \wedge (\varphi_3 \vee \mathsf{P_s}\varphi_3 \vee \mathsf{F_s}\varphi_3)$. Hence, $A_\varphi = A_{\mathsf{G_s}\neg tt} \cup A_{\varphi'}$ where $A_{\varphi'}$ is constructed from $A_{\mathsf{F_s}tt}$, $A_{\mathsf{G_s F_s}tt}$, and $A_{\varphi_3} \cup A_{\mathsf{P_s}\varphi_3} \cup A_{\mathsf{F_s}\varphi_3}$ using Lemma 10.27.

$-\mathsf{G_s}$ **Case** $\mathsf{G_s F_s G_s}\varphi_3$**:** A pointed word (u, i) satisfies $\mathsf{G_s F_s G_s}\varphi_3$ iff $i = |u| - 1$ or u is an infinite word satisfying $\mathsf{F_s G_s}\varphi_3$. Thus, $\mathsf{G_s F_s G_s}\varphi_3 \equiv (\mathsf{G_s}\neg tt) \vee \varphi'$ where $\varphi' = (\mathsf{F_s}tt) \wedge (\mathsf{G_s F_s}tt) \wedge (\mathsf{F_s G_s}\varphi_3)$. Hence, $A_\varphi = A_{\mathsf{G_s}\neg tt} \cup A_{\varphi'}$ where $A_{\varphi'}$ is constructed from $A_{\mathsf{F_s}tt}$, $A_{\mathsf{G_s F_s}tt}$, and $A_{\mathsf{F_s G_s}\varphi_3}$ using Lemma 10.27.

$-\mathsf{H_s}$ **Case** $\mathsf{G_s F_s H_s}\varphi_3$**:** A pointed word (u, i) satisfies $\mathsf{G_s F_s H_s}\varphi_3$ iff $i = |u| - 1$ or u is an infinite word satisfying $\mathsf{G}\varphi_3$. Thus, $\mathsf{G_s F_s H_s}\varphi_3 \equiv (\mathsf{G_s}\neg tt) \vee \varphi'$ where $\varphi' = (\mathsf{F_s}tt) \wedge (\mathsf{G_s F_s}tt) \wedge (\varphi_3 \wedge \mathsf{H_s}\varphi_3 \wedge \mathsf{G_s}\varphi_3)$. Hence, $A_\varphi = A_{\mathsf{G_s}\neg tt} \cup A_{\varphi'}$ where $A_{\varphi'}$ is constructed from $A_{\mathsf{F_s}tt}$, $A_{\mathsf{G_s F_s}tt}$, A_{φ_3}, $A_{\mathsf{H_s}\varphi_3}$, and $A_{\mathsf{G_s}\varphi_3}$ using Lemma 10.27.

$\circ\mathsf{P_s}$ **Case** $\mathsf{G_s P_s}\varphi_2$**:** A pointed word (u, i) satisfies $\mathsf{G_s P_s}\varphi_2$ iff $i = |u| - 1$ or (u, i) satisfies $\mathsf{P}\varphi_2$. Hence, $A_\varphi = A_{\mathsf{G_s}\neg tt} \cup A_{\varphi_2} \cup A_{\mathsf{P_s}\varphi_2}$.

$\circ\vee$ **Case** $\mathsf{G_s}(\varphi_2 \vee \varphi_3)$**:** According to the structure of φ_2 and φ_3, there are the following subcases.

$\star p$ **Case** $\mathsf{G_s}(p_2 \vee p_3)$**:** As $p_2 \vee p_3 \in \mathrm{LTL}()$, this subcase has already been covered by Case $\mathsf{G_s}p$.

$\star\wedge$ **Case** $\mathsf{G_s}(\varphi_2 \vee (\varphi_4 \wedge \varphi_5))$**:** As $\mathsf{G_s}(\varphi_2 \vee (\varphi_4 \wedge \varphi_5)) \equiv \mathsf{G_s}(\varphi_2 \vee \varphi_4) \wedge \mathsf{G_s}(\varphi_2 \vee \varphi_5)$, the set A_φ can be constructed from $A_{\mathsf{G_s}(\varphi_2 \vee \varphi_4)}$ and $A_{\mathsf{G_s}(\varphi_2 \vee \varphi_5)}$ using Lemma 10.27.

$\star\mathsf{F_s}$ **Case** $\mathsf{G_s}(\varphi_2 \vee \mathsf{F_s}\varphi_4)$**:** It holds that $\mathsf{G_s}(\varphi_2 \vee \mathsf{F_s}\varphi_4) \equiv (\mathsf{G_s}\varphi_2) \vee \mathsf{F_s}(\mathsf{F_s}\varphi_4 \wedge \mathsf{G_s}\varphi_2) \vee \mathsf{G_s F_s}\varphi_4$. Therefore, the set A_φ can be constructed as $A_{\mathsf{G_s}\varphi_2} \cup A_{\mathsf{F_s}(\mathsf{F_s}\varphi_4 \wedge \mathsf{G_s}\varphi_2)} \cup A_{\mathsf{G_s F_s}\varphi_4}$,

where $A_{\mathsf{F}_s(\mathsf{F}_s\varphi_4 \wedge \mathsf{G}_s\varphi_2)}$ is created from $A_{\mathsf{F}_s\varphi_4}$ and $A_{\mathsf{G}_s\varphi_2}$ due to Lemma 10.27 and Lemma 10.32.

$\star\mathsf{H}_s$ **Case** $\mathsf{G}_s(\varphi_2 \vee \mathsf{H}_s\varphi_4)$**:** As $\mathsf{G}_s(\varphi_2 \vee \mathsf{H}_s\varphi_4) \equiv (\mathsf{G}_s\varphi_2) \vee \mathsf{F}_s(\mathsf{H}_s\varphi_4 \wedge \mathsf{G}_s\varphi_2) \vee \mathsf{G}_s\mathsf{H}_s\varphi_4$. Hence, $A_\varphi = A_{\mathsf{G}_s\varphi_2} \cup A_{\mathsf{F}_s(\mathsf{H}_s\varphi_4 \wedge \mathsf{G}_s\varphi_2)} \cup A_{(\mathsf{G}_s\mathsf{H}_s\varphi_4)}$ where $A_{\mathsf{F}_s(\mathsf{H}_s\varphi_4 \wedge \mathsf{G}_s\varphi_2)}$ can be created from $A_{\mathsf{H}_s\varphi_4}$ and $A_{\mathsf{G}_s\varphi_2}$ using Lemma 10.27 and Lemma 10.32.

$\star\mathsf{G}_s, \mathsf{P}_s$ **Case** $\mathsf{G}_s(\varphi_2 \vee \mathsf{G}_s\varphi_4 \vee \mathsf{P}_s\varphi_5)$**:** There are only the following five subcases (the others fit to some of the previous cases).

(i) **Case** $\mathsf{G}_s(\bigvee_{\varphi' \in G} \mathsf{G}_s\varphi')$**:** It holds that $\mathsf{G}_s(\bigvee_{\varphi' \in G} \mathsf{G}_s\varphi') \equiv (\mathsf{G}_s\neg tt) \vee \bigvee_{\varphi' \in G}(\mathsf{X}\mathsf{G}_s\varphi')$. Therefore, the set A_φ can be constructed as $A_{\mathsf{G}_s\neg tt} \cup \bigcup_{\varphi' \in G} A_{\mathsf{X}\mathsf{G}_s\varphi'}$ where each $A_{\mathsf{X}\mathsf{G}_s\varphi'}$ is created from $A_{\mathsf{G}_s\varphi'}$ using Lemma 10.28.

(ii) **Case** $\mathsf{G}_s(p_2 \vee \bigvee_{\varphi' \in G} \mathsf{G}_s\varphi')$**:** As $\mathsf{G}_s(p_2 \vee \bigvee_{\varphi' \in G} \mathsf{G}_s\varphi') \equiv (\mathsf{G}_s p_2) \vee \bigvee_{\varphi' \in G}(\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{G}_s\varphi')))$. Therefore, the set A_φ can be constructed as $A_{\mathsf{G}_s p_2} \cup \bigcup_{\varphi' \in G} A_{\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{G}_s\varphi'))}$ where each $A_{\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{G}_s\varphi'))}$ is created from $A_{\mathsf{G}_s\varphi'}$ using Lemma 10.30 and Lemma 10.28.

(iii) **Case** $\mathsf{G}_s(\bigvee_{\varphi'' \in P} \mathsf{P}_s\varphi'')$**:** It holds that $\mathsf{G}_s(\bigvee_{\varphi'' \in P} \mathsf{P}_s\varphi'') \equiv (\mathsf{G}_s\neg tt) \vee \bigvee_{\varphi'' \in P}(\mathsf{X}\mathsf{P}_s\varphi'')$. Therefore, the set A_φ can be constructed as $A_{\mathsf{G}_s\neg tt} \cup \bigcup_{\varphi'' \in P} A_{\mathsf{X}\mathsf{P}_s\varphi''}$ where each $A_{\mathsf{X}\mathsf{P}_s\varphi''}$ is created from $A_{\mathsf{P}_s\varphi''}$ using Lemma 10.28.

(iv) **Case** $\mathsf{G}_s(p_2 \vee \bigvee_{\varphi'' \in P} \mathsf{P}_s\varphi'')$**:** As $\mathsf{G}_s(p_2 \vee \bigvee_{\varphi'' \in P} \mathsf{P}_s\varphi'') \equiv (\mathsf{G}_s p_2) \vee \bigvee_{\varphi'' \in P}(\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{P}_s\varphi'')))$. Therefore, the set A_φ can be constructed as $A_{\mathsf{G}_s p_2} \cup \bigcup_{\varphi'' \in P} A_{\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{P}_s\varphi''))}$ where each $A_{\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{P}_s\varphi''))}$ is created from $A_{\mathsf{P}_s\varphi''}$ using Lemma 10.30 and Lemma 10.28.

(v) **Case** $\mathsf{G}_s(p_2 \vee \bigvee_{\varphi' \in G} \mathsf{G}_s\varphi' \vee \bigvee_{\varphi'' \in P} \mathsf{P}_s\varphi'')$**:** As $\mathsf{G}_s(p_2 \vee \bigvee_{\varphi' \in G} \mathsf{G}_s\varphi' \vee \bigvee_{\varphi'' \in P} \mathsf{G}_s\varphi'') \equiv (\mathsf{G}_s p_2) \vee \bigvee_{\varphi' \in G}(\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{G}_s\varphi'))) \vee \bigvee_{\varphi'' \in P}(\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{P}_s\varphi'')))$. Therefore, the set A_φ can be constructed as $A_{\mathsf{G}_s p_2} \cup \bigcup_{\varphi' \in G} A_{\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{G}_s\varphi'))} \cup \bigcup_{\varphi'' \in P} A_{\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{P}_s\varphi''))}$ where each $A_{\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{G}_s\varphi'))}$ is created from $A_{\mathsf{G}_s\varphi'}$ and each $A_{\mathsf{X}(p_2 \,\mathsf{U}\, (\mathsf{P}_s\varphi''))}$ is created from $A_{\mathsf{P}_s\varphi''}$ using Lemma 10.30 and Lemma 10.28.

$\circ\mathsf{G}_s$ **Case** $\mathsf{G}_s\mathsf{G}_s\varphi_2$**:** As $\mathsf{G}_s(\mathsf{G}_s\varphi_2) \equiv (\mathsf{G}_s\neg tt) \vee (\mathsf{X}\mathsf{G}_s\varphi_2)$, the set A_φ can be constructed as $A_{\mathsf{G}_s\neg tt} \cup A_{\mathsf{X}\mathsf{G}_s\varphi_2}$ where $A_{\mathsf{X}\mathsf{G}_s\varphi_2}$ is created from $A_{\mathsf{G}_s\varphi_2}$ using Lemma 10.28.

$\circ\mathsf{H}_s$ **Case** $\mathsf{G}_s\mathsf{H}_s\varphi_2$**:** A pointed word (u, i) satisfies $\mathsf{G}_s(\mathsf{H}_s\varphi_2)$ iff $i = |u| - 1$ or $(u, |u| - 1)$ satisfies $\mathsf{H}_s\varphi_2$ or u is infinite and all its positions satisfy φ_2. Hence, $A_\varphi = A_{\mathsf{G}_s\neg tt} \cup A_{\mathsf{F}_s((\mathsf{G}_s\neg tt) \wedge (\mathsf{H}_s\varphi_2))} \cup A_{(\mathsf{H}_s\varphi_2) \wedge \varphi_2 \wedge (\mathsf{G}_s\varphi_2)}$ where $A_{\mathsf{F}_s((\mathsf{G}_s\neg tt) \wedge (\mathsf{H}_s\varphi_2))}$ and $A_{(\mathsf{H}_s\varphi_2) \wedge \varphi_2 \wedge (\mathsf{G}_s\varphi_2)}$ is created from $A_{\mathsf{G}_s\neg tt}$, $A_{\mathsf{H}_s\varphi_2}$, A_{φ_2}, and $A_{\mathsf{G}_s\varphi_2}$ using Lemma 10.27 and Lemma 10.32.

- $\bullet\mathsf{H_s}$ **Case** $\mathsf{H_s}\varphi_1$**:** This case is divided into the following subcases according to the structure of φ_1.
  - $\circ p$ **Case** $\mathsf{H_s}p$**:** As $\mathsf{H_s}p$ is equivalent to $tt \wedge \mathsf{H_s}p$, we set $A_\varphi = \{P\alpha(tt \wedge \mathsf{H_s}p, tt \wedge \mathsf{G_s}tt, \emptyset)\}$.
  - $\circ\wedge$ **Case** $\mathsf{H_s}(\varphi_2 \wedge \varphi_3)$**:** As $\mathsf{H_s}(\varphi_2 \wedge \varphi_3) \equiv (\mathsf{H_s}\varphi_2) \wedge (\mathsf{H_s}\varphi_3)$, the set A_φ can be constructed from $A_{\mathsf{H_s}\varphi_2}$ and $A_{\mathsf{H_s}\varphi_3}$ using Lemma 10.27.
  - $\circ\mathsf{F_s}$ **Case** $\mathsf{H_s}\mathsf{F_s}\varphi_2$**:** A pointed word (u, i) satisfies $\mathsf{H_s}\mathsf{F_s}\varphi_2$ iff $i = 0$ or (u, i) satisfies $\mathsf{F}\varphi_2$. Note that $\mathsf{H_s}\neg tt$ is satisfied by (u, i) only if $i = 0$. Therefore, $A_\varphi = A_{\mathsf{H_s}\neg tt} \cup A_{\varphi_2} \cup A_{\mathsf{F_s}\varphi_2}$.
  - $\circ\mathsf{P_s}$ **Case** $\mathsf{H_s}\mathsf{P_s}\varphi_2$**:** Every run has to start in the initial state, and so, every history is finite. Hence, a pointed word (u, i) satisfies $\mathsf{H_s}\mathsf{P_s}\varphi_2$ iff $i = 0$. Therefore, $A_\varphi = A_{\mathsf{H_s}\neg tt}$.
  - $\circ\vee$ **Case** $\mathsf{H_s}(\varphi_2 \vee \varphi_3)$**:** According to the structure of φ_2 and φ_3, there are the following subcases.
    - $\star p$ **Case** $\mathsf{H_s}(p_2 \vee p_3)$**:** As $p_2 \vee p_3 \in$ LTL(), this subcase has already been covered by Case $\mathsf{H_s}p$.
    - $\star\wedge$ **Case** $\mathsf{H_s}(\varphi_2 \vee (\varphi_4 \wedge \varphi_5))$**:** As $\mathsf{H_s}(\varphi_2 \vee (\varphi_4 \wedge \varphi_5)) \equiv \mathsf{H_s}(\varphi_2 \vee \varphi_4) \wedge \mathsf{H_s}(\varphi_2 \vee \varphi_5)$, the set A_φ can be constructed from $A_{\mathsf{H_s}(\varphi_2 \vee \varphi_4)}$ and $A_{\mathsf{H_s}(\varphi_2 \vee \varphi_5)}$ using Lemma 10.27.
    - $\star\mathsf{P_s}$ **Case** $\mathsf{H_s}(\varphi_2 \vee \mathsf{P_s}\varphi_4)$**:** It holds that $\mathsf{H_s}(\varphi_2 \vee \mathsf{P_s}\varphi_4) \equiv (\mathsf{H_s}\varphi_2) \vee \mathsf{P_s}(\mathsf{P_s}\varphi_4 \wedge \mathsf{H_s}\varphi_2)$. Therefore, the set A_φ can be constructed as $A_{\mathsf{H_s}\varphi_2} \cup A_{\mathsf{P_s}(\mathsf{P_s}\varphi_4 \wedge \mathsf{H_s}\varphi_2)}$, where $A_{\mathsf{P_s}(\mathsf{P_s}\varphi_4 \wedge \mathsf{H_s}\varphi_2)}$ is created from $A_{\mathsf{P_s}\varphi_4}$ and $A_{\mathsf{H_s}\varphi_2}$ due to Lemma 10.27 and Lemma 10.33.
    - $\star\mathsf{G_s}$ **Case** $\mathsf{H_s}(\varphi_2 \vee \mathsf{G_s}\varphi_4)$**:** As $\mathsf{H_s}(\varphi_2 \vee \mathsf{G_s}\varphi_4) \equiv (\mathsf{H_s}\varphi_2) \vee \mathsf{P_s}(\mathsf{G_s}\varphi_4 \wedge \mathsf{H_s}\varphi_2)$. Hence, A_φ is constructed as $A_{\mathsf{H_s}\varphi_2} \cup A_{\mathsf{P_s}(\mathsf{G_s}\varphi_4 \wedge \mathsf{H_s}\varphi_2)}$ where $A_{\mathsf{P_s}(\mathsf{G_s}\varphi_4 \wedge \mathsf{H_s}\varphi_2)}$ is created from $A_{\mathsf{G_s}\varphi_4}$ and $A_{\mathsf{H_s}\varphi_2)}$ using Lemma 10.27 and Lemma 10.33.
    - $\star\mathsf{F_s}, \mathsf{H_s}$ **Case** $\mathsf{H_s}(\varphi_2 \vee \mathsf{F_s}\varphi_4 \vee \mathsf{H_s}\varphi_5)$**:** There are only the following five subcases (the others fit to some of the previous cases).
      - (i) **Case** $\mathsf{H_s}(\bigvee_{\varphi' \in F} \mathsf{F_s}\varphi')$**:** It holds that $\mathsf{H_s}(\bigvee_{\varphi' \in F} \mathsf{F_s}\varphi') \equiv (\mathsf{H_s}\neg tt) \vee \bigvee_{\varphi' \in F}(\mathsf{Y}\mathsf{F_s}\varphi')$. Therefore, the set A_φ can be constructed as $A_{\mathsf{H_s}\neg tt} \cup \bigcup_{\varphi' \in F} A_{\mathsf{Y}\mathsf{F_s}\varphi'}$ where each $A_{\mathsf{Y}\mathsf{F_s}\varphi'}$ is created from $A_{\mathsf{F_s}\varphi'}$ using Lemma 10.29.
      - (ii) **Case** $\mathsf{H_s}(p_2 \vee \bigvee_{\varphi' \in F} \mathsf{F_s}\varphi')$**:** As $\mathsf{H_s}(p_2 \vee \bigvee_{\varphi' \in F} \mathsf{F_s}\varphi') \equiv (\mathsf{H_s}p_2) \vee \bigvee_{\varphi' \in F}(\mathsf{Y}(p_2\,\mathsf{S}\,(\mathsf{F_s}\varphi')))$. Therefore, the set A_φ can be constructed as $A_{\mathsf{H_s}p_2} \cup \bigcup_{\varphi' \in F} A_{\mathsf{Y}(p_2\,\mathsf{S}\,(\mathsf{F_s}\varphi'))}$ where each $A_{\mathsf{Y}(p_2\,\mathsf{S}\,(\mathsf{F_s}\varphi'))}$ is created from $A_{\mathsf{F_s}\varphi'}$ using Lemma 10.31 and Lemma 10.29.
      - (iii) **Case** $\mathsf{H_s}(\bigvee_{\varphi'' \in H} \mathsf{H_s}\varphi'')$**:** It holds that $\mathsf{H_s}(\bigvee_{\varphi'' \in H} \mathsf{H_s}\varphi'') \equiv (\mathsf{H_s}\neg tt) \vee \bigvee_{\varphi'' \in H}(\mathsf{Y}\mathsf{H_s}\varphi'')$. Therefore, the set A_φ can be constructed as $A_{\mathsf{H_s}\neg tt} \cup \bigcup_{\varphi'' \in H} A_{\mathsf{Y}\mathsf{H_s}\varphi''}$ where each $A_{\mathsf{Y}\mathsf{H_s}\varphi''}$ is created from $A_{\mathsf{H_s}\varphi''}$ using Lemma 10.29.

(iv) **Case** $\mathsf{H_s}(p_2 \vee \bigvee_{\varphi'' \in H} \mathsf{H_s}\varphi'')$**:** As $\mathsf{H_s}(p_2 \vee \bigvee_{\varphi'' \in H} \mathsf{H_s}\varphi'') \equiv (\mathsf{H_s}p_2) \vee \bigvee_{\varphi'' \in H}(\mathsf{Y}(p_2 \,\mathsf{S}\, (\mathsf{H_s}\varphi'')))$. Therefore, the set A_φ can be constructed as $A_{\mathsf{H_s}p_2} \cup \bigcup_{\varphi'' \in H} A_{\mathsf{Y}(p_2 \,\mathsf{S}\, (\mathsf{H_s}\varphi''))}$ where each $A_{\mathsf{Y}(p_2 \,\mathsf{S}\, (\mathsf{H_s}\varphi''))}$ is created from $A_{\mathsf{H_s}\varphi''}$ using Lemma 10.29 and Lemma 10.31.

(v) **Case** $\mathsf{H_s}(p_2 \vee \bigvee_{\varphi' \in F} \mathsf{F_s}\varphi' \vee \bigvee_{\varphi'' \in H} \mathsf{H_s}\varphi'')$**:** As $\mathsf{H_s}(p_2 \vee \bigvee_{\varphi' \in F} \mathsf{F_s}\varphi' \vee \bigvee_{\varphi'' \in H} \mathsf{H_s}\varphi'') \equiv (\mathsf{H_s}p_2) \vee \bigvee_{\varphi' \in F}(\mathsf{Y}(p_2 \,\mathsf{S}\, (\mathsf{F_s}\varphi'))) \vee \bigvee_{\varphi'' \in H}(\mathsf{Y}(p_2 \,\mathsf{S}\, (\mathsf{H_s}\varphi'')))$. Therefore, the set A_φ can be constructed as $A_{\mathsf{H_s}p_2} \cup \bigcup_{\varphi' \in F} A_{\mathsf{Y}(p_2 \,\mathsf{S}\, (\mathsf{F_s}\varphi'))} \cup \bigcup_{\varphi'' \in H} A_{\mathsf{Y}(p_2 \,\mathsf{S}\, (\mathsf{H_s}\varphi''))}$ where each $A_{\mathsf{Y}(p_2 \,\mathsf{S}\, (\mathsf{F_s}\varphi'))}$ is created from $A_{\mathsf{F_s}\varphi'}$ and each $A_{\mathsf{Y}(p_2 \,\mathsf{S}\, (\mathsf{H_s}\varphi''))}$ is created from $A_{\mathsf{H_s}\varphi''}$ using Lemma 10.31 and Lemma 10.29.

$\circ\mathsf{G_s}$ **Case** $\mathsf{H_s}\mathsf{G_s}\varphi_2$**:** A pointed word (u, i) satisfies $\mathsf{H_s}(\mathsf{G_s}\varphi_2)$ iff $i = 0$ or $(u, 0)$ satisfies $\mathsf{G_s}\varphi_2$. Hence, $A_\varphi = A_{\mathsf{H_s}\neg tt} \cup A_{\mathsf{P_s}((\mathsf{H_s}\neg tt) \wedge (\mathsf{G_s}\varphi_2))}$ where $A_{\mathsf{P_s}((\mathsf{H_s}\neg tt) \wedge (\mathsf{G_s}\varphi_2))}$ is created from $A_{\mathsf{H_s}\neg tt}$ and $A_{\mathsf{G_s}\varphi_2}$ using Lemma 10.27 and Lemma 10.33.

$\circ\mathsf{H_s}$ **Case** $\mathsf{H_s}\mathsf{H_s}\varphi_2$**:** As $\mathsf{H_s}(\mathsf{H_s}\varphi_2) \equiv (\mathsf{H_s}\neg tt) \vee (\mathsf{Y}\mathsf{H_s}\varphi_2)$, the set A_φ can be constructed as $A_{\mathsf{H_s}\neg tt} \cup A_{\mathsf{Y}\mathsf{H_s}\varphi_2}$ where $A_{\mathsf{Y}\mathsf{H_s}\varphi_2}$ is created from $A_{\mathsf{H_s}\varphi_2}$ using Lemma 10.29.

□

In other words, we have shown that $LTL(\mathsf{F_s}, \mathsf{P_s})$ is a semantic subset (with respect to global equivalence) of every formalism that is able to express

- p where $p \in \mathrm{LTL}()$,
- $\mathsf{G_s}p$ where $p \in \mathrm{LTL}()$,
- $\mathsf{H_s}p$ where $p \in \mathrm{LTL}()$, and
- $\mathsf{G_s}\mathsf{F_s}p$ where $p \in \mathrm{LTL}()$

and is closed under

- $\vee$ operation,
- $\wedge$ operation,
- X application,
- $p\,\mathsf{U}$ application where $p \in \mathrm{LTL}()$,
- Y application, and
- $p\,\mathsf{S}$ application where $p \in \mathrm{LTL}()$.

Now, using Theorem 10.11 and Theorem 10.24 we can easily proof decidability of the model checking problem for wPRS and negated formulae of the $\mathcal{PA}$ fragment. In fact, we prove decidability of the dual problem, i.e. whether a given wPRS system has a run satisfying a given formula of $\mathcal{PA}$.

Theorem 10.35. *The problem whether a given wPRS system has a run satisfying a given $P\alpha$-formula is decidable.*

Proof. A run over a word u satisfies a formula φ if and only if $(u, 0) \models \varphi$. Moreover, $(u, 0) \models P\alpha(\eta, \delta, \mathcal{B})$ if and only if $(u(0), 0) \models \eta$ and $(u, 0) \models \alpha(\delta, \mathcal{B})$. Due to Theorem 10.11 and Theorem 10.24, we can check whether $(u, 0) \models \alpha(\delta, \mathcal{B})$. Therefore, it remains to show how to check $(u(0), 0) \models \eta$. Let $\eta = \iota_1 P_1 \iota_2 P_2 \dots \iota_m P_m \iota_{m+1}$. It follows from Definition 10.4 that $(u(0), 0) \models \eta$ if and only if $(u(0), 0) \models \iota_m$ and $P_i = \mathsf{S}$ for all $i < m$. □

As LTL($\mathsf{F_s}, \mathsf{P_s}$) is closed under negation, Theorem 10.34 and Theorem 10.35 give us the following.

Corollary 10.36. *The model checking problem for wPRS and LTL($\mathsf{F_s}, \mathsf{P_s}$) is decidable.*

Moreover, we can show that the pointed model checking problem is decidable for wPRS and LTL($\mathsf{F_s}, \mathsf{P_s}$) as well. Also in this case we solve the dual problem, i.e. whether a given wPRS system has a pointed run satisfying a given formula of $\mathcal{PA}$.

Theorem 10.37. *Let Δ be a wPRS and pt be a reachable nonterminal state of Δ. The problem whether $L(pt, \Delta)$ includes a pointed word (u, i) satisfying a given $P\alpha$-formula is decidable.*

Proof. Let $\Delta = (M, \geq, R, p_0, t_0)$ be a wPRS and pt be a reachable nonterminal state of Δ. We construct a wPRS $\Delta' = (M, \geq, R', p_0, t_0.X)$ where $X \notin Const(\Delta)$ is a fresh process constant, $f \notin Act(\Delta)$ is a fresh action,

$$R' = R \cup \{(p(t.X) \xhookrightarrow{a} pX_a), (pX_a \xhookrightarrow{f} pY_a), (pY_a \xhookrightarrow{a} p't') \mid pt \xrightarrow{a}_{\Delta} p't'\},$$

and $X_a, Y_a \notin Const(\Delta)$ are fresh process constants for each $a \in Act(\Delta)$.

It is easy to see that (u, i) is in $L(pt, \Delta)$ if and only if

$$u(0)u(1) \dots u(i-1)u(i).f.u(i).u(i+1) \dots$$

is in $L(\Delta')$. Hence, for a given $P\alpha$-formula $\varphi = P\alpha(\eta, \delta, \mathcal{B})$ we construct a $P\alpha$-formula $\varphi' = P\alpha(\eta, tt \wedge \mathsf{X}f \wedge \mathsf{X}\delta, \mathcal{B})$. We get that

$$L(pt, \Delta) \models P\alpha(\eta, \delta, \mathcal{B}) \quad \iff \quad L(\Delta') \models \mathsf{F}(P\alpha(\eta, tt \wedge \mathsf{X}f \wedge \mathsf{X}\delta, \mathcal{B}))$$

and due to Lemma 10.30 and Theorem 10.35 the proof is done. □

As LTL($\mathsf{F_s}, \mathsf{P_s}$) is closed under negation and Theorem 10.34 works with global equivalence, Theorem 10.37 give us the following.

Corollary 10.38. *The pointed model checking problem is decidable for wPRS and LTL*($\mathsf{F_s}, \mathsf{P_s}$).

10.6 Undecidability Results

In this section we prove that the model checking problem is undecidable for the PA class and the fragments LTL(U) and LTL($\overset{\infty}{\mathsf{F}}, X$), respectively. The undecidability proofs are based on reduction from the non-halting problem for Minsky 2-counter machines, which is known to be undecidable [Min67]. See Subsection 7.3.2 for the Minsky 2-counter machine definition.

Theorem 10.39. *The model checking problem for PA and LTL*(U) *is undecidable.*

Proof. Given a machine $N = l_1 : i_1,\ l_2 : i_2,\ \ldots,\ l_{n-1} : i_{n-1},\ l_n : \texttt{halt}$, we construct a PA system Δ_N with the initial term $D_1 \| D_2 \| H$. In what follows we construct sets of rewrite rules emulating instructions of the machine N. Moreover, for each instruction, we define an LTL formula describing a correct simulation of the instruction in the PA system.

Increment: $l_i : \texttt{c}_k\texttt{:=c}_k\texttt{+1; goto } l_r$

To each such an increment instruction of the machine N we add to Δ the following rules:

$$D_k \overset{l_i}{\hookrightarrow} S_k.D_k \qquad C_k \overset{l_i}{\hookrightarrow} S_k.C_k \qquad S_k \overset{inc_i}{\hookrightarrow} C_k.S_k$$

The correctness formula ψ_i of this instruction is as follows:

$$(l_i \Longrightarrow (l_i \,\mathsf{U}\, inc_i)) \ \wedge\ (inc_i \Longrightarrow (inc_i \,\mathsf{U}\, l_r)).$$

Intuitively, it says that every l_i action is followed by the inc_i action that has to be followed by the l_r action.

Test-and-decrement:

$$l_i\texttt{: if c}_k\texttt{>0 then c}_k\texttt{:=c}_k\texttt{-1; goto } l_r \texttt{ else goto } l_s$$

To each such a test-and-decrement instruction of the machine N we add to Δ the following rules:

$$D_k \overset{l_i}{\hookrightarrow} E_k \qquad E_k \overset{zero_i}{\hookrightarrow} D_k \qquad C_k \overset{l_i}{\hookrightarrow} \varepsilon \qquad S_k \overset{dec_i}{\hookrightarrow} \varepsilon$$

The correctness formula ψ_i of this test-and-decrement instruction is as follows:

$$(l_i \Longrightarrow (l_i \,\mathsf{U}\, (dec_i \vee zero_i))) \wedge (dec_i \Longrightarrow (dec_i \,\mathsf{U}\, l_r)) \wedge (zero_i \Longrightarrow (zero_i \,\mathsf{U}\, l_s)).$$

Intuitively, it says that every l_i action is followed by the dec_i action or the $zero_i$ action. Moreover, the two last conjuncts express that the actions dec_i and $zero_i$ has to be followed by the l_r and l_s, respectively.

Halt: l_n : `halt`

The halt instruction is translated into the following rule:

$$H \overset{l_n}{\hookrightarrow} H$$

Finally, we define a general formula φ as follows:

$$\varphi = l_1 \;\wedge\; ((\bigwedge_{1 \leq i < n} \psi_i) \,\mathsf{U}\, l_n).$$

It is easy to see that machine N halts if and only if the system Δ_N has a run satisfying φ. In other words, the machine N does not halt if and only if $L(\Delta_N) \models \neg\varphi$. □

Theorem 10.40. *The model checking problem for PA and LTL*$(\overset{\infty}{\mathsf{F}}, \mathsf{X})$ *is undecidable.*

Proof. Given a machine $N = l_1 : i_1,\ l_2 : i_2,\ \ldots,\ l_{n-1} : i_{n-1},\ l_n :$ `halt`, we construct a PA system Δ_N with the initial term $D_1 \| D_2 \| H$. In what follows we construct sets of rewrite rules emulating instructions of the machine N. Moreover, for each instruction we define an LTL formula describing a correct simulation of the instruction in the PA system.

Increment: l_i : `c`$_k$`:=c`$_k$`+1; goto` l_r

To each such an increment instruction of the machine N we add to Δ the following rules:

$$D_k \overset{inc_i}{\hookrightarrow} C_k.D_k \qquad\qquad C_k \overset{inc_i}{\hookrightarrow} C_k.C_k$$

The correctness formula ψ_i of this instruction is as follows:

$$(l_i \Longrightarrow \mathsf{X} inc_i) \;\wedge\; (inc_i \Longrightarrow \mathsf{X} l_r).$$

Intuitively, it says that every l_i action is followed by the inc_i action that has to be followed by the l_r action.

Test-and-decrement:

l_i: `if` c_k`>0 then` c_k`:=` c_k`-1; goto` l_r `else goto` l_s

To each such a test-and-decrement instruction of the machine N we add to Δ the following rules:

$$D_k \overset{zero_i}{\hookrightarrow} D_k \qquad\qquad C_k \overset{dec_i}{\hookrightarrow} \varepsilon$$

The correctness formula ψ_i of this test-and-decrement instruction is as follows:

$$(l_i \Longrightarrow \mathsf{X}(dec_i \vee zero_i)) \wedge (dec_i \Longrightarrow \mathsf{X}l_r) \wedge (zero_i \Longrightarrow \mathsf{X}l_s).$$

Intuitively, it says that every l_i action is followed by the dec_i action or the $zero_i$ action. Moreover, the two last conjuncts express that the actions dec_i and $zero_i$ has to be followed by the l_r and l_s, respectively.

Halt: l_n : `halt`

The halt instruction is translated into the following rules.

$$H \overset{halt}{\hookrightarrow} H \qquad\qquad H \overset{l_i}{\hookrightarrow} H \text{ for every } 1 \leq i \leq n$$

Restart:

Additionally, we also add the rules enabling to reset the counters.

$$C_1 \overset{del_1}{\hookrightarrow} \varepsilon \qquad C_2 \overset{del_2}{\hookrightarrow} \varepsilon \qquad D_1 \overset{reset_1}{\hookrightarrow} D_1 \qquad D_2 \overset{reset_2}{\hookrightarrow} D_2$$

As in the previous proof, we define a formula ψ which describes a correct step of the constructed PA system when simulating machine N.

$$\psi = \bigwedge_{1 \leq i < n} \psi_i \wedge (l_n \Longrightarrow \mathsf{X}halt)$$

Moreover, we define a formula ρ describing a correct step of resetting counters and restarting the simulation.

$$\begin{aligned} \rho \;=\; & (halt \Longrightarrow \mathsf{X}(del_1 \vee reset_1)) \wedge (del_1 \Longrightarrow \mathsf{X}(del_1 \vee reset_1)) \\ & \wedge\; (reset_1 \Longrightarrow \mathsf{X}(del_2 \vee reset_2)) \wedge (del_2 \Longrightarrow \mathsf{X}(del_2 \vee reset_2)) \\ & \wedge\; (reset_2 \Longrightarrow \mathsf{X}l_1) \end{aligned}$$

The formula $\varphi = \overset{\infty}{\mathsf{G}}(\psi \wedge \rho) \wedge \overset{\infty}{\mathsf{F}} halt$ says that, at some point, the $halt$ action occurs, both counters are reset, a correct simulation is started, and whenever the simulation ends (with the action $halt$), this sequence of events is performed again. Moreover, note that φ is satisfied only if the action $halt$ appears infinitely often. Hence, there is a run of Δ_N satisfying φ if and only if N halts. In other words, the machine N does not halt if and only if $L(\Delta_N) \models \neg\varphi$. □

Remark 10.41. *In the proofs of the previous two theorems, the PA systems constructed have only infinite runs. This means that model checking of* infinite runs *remains undecidable for PA and both LTL*$(\overset{\infty}{\mathsf{F}}, \mathsf{X})$ *and LTL*(U).

Remark 10.42. *It can be easily shown that model checking of* finite runs *for PA and LTL*(U) *is undecidable as well. To that end, it suffices to replace the rule* $H \overset{l_n}{\hookrightarrow} H$ *by the rule* $H \overset{l_n}{\hookrightarrow} \varepsilon$ *in the proof of Theorem 10.39.*

Remark 10.43. *As all the systems of the extended PRS-hierarchy are finitely branching, the number of states reachable in a given finite number of steps is finite. Therefore, it can be easily shown that the model checking problem is decidable for LTL*(X) *and sePRS.*

Remark 10.44. *However, note that model checking of* finite runs *against LTL*$(\overset{\infty}{\mathsf{F}}, \mathsf{X})$ *is decidable, even for wPRS. The proof is based on the observation that a nonempty finite run satisfies* $\overset{\infty}{\mathsf{F}}\varphi$ *if and only if the last action of the run satisfies* φ*. The same holds for a formula* $\overset{\infty}{\mathsf{G}}\varphi$*. Hence, if we restrict only to nonempty finite runs, the modalities* $\overset{\infty}{\mathsf{F}}, \overset{\infty}{\mathsf{G}}$ *are equivalent. The observation also implies that* $\overset{\infty}{\mathsf{F}}\neg\varphi$ *is equivalent to* $\neg\overset{\infty}{\mathsf{F}}\varphi$*,* $\overset{\infty}{\mathsf{F}}(\varphi_1 \wedge \varphi_2)$ *is equivalent to* $(\overset{\infty}{\mathsf{F}}\varphi_1) \wedge (\overset{\infty}{\mathsf{F}}\varphi_2)$*,* $\overset{\infty}{\mathsf{F}}\overset{\infty}{\mathsf{F}}\varphi$ *is equivalent to* $\overset{\infty}{\mathsf{F}}\varphi$*, and that* $\overset{\infty}{\mathsf{F}}\mathsf{X}\varphi$ *never holds. Now it is easy to see that every LTL*$(\overset{\infty}{\mathsf{F}}, \mathsf{X})$ *formula can describe only a bounded prefix of a finite run (using the modality* X*) and the last action of the run. Thus, decidability of model checking of finite runs against LTL*$(\overset{\infty}{\mathsf{F}}, \mathsf{X})$ *follows from Remark 10.43 and decidability of the* reachability Hennessy-Milner property *problem [KŘS05].*

Remark 10.45. *Last but not least, let us note that model checking of* finite runs *against LTL*(F, X) *and PA is undecidable. To prove this, we can use the proof of Theorem 10.40 where the rule* $H \overset{halt}{\hookrightarrow} H$ *is replaced by* $H \overset{halt}{\hookrightarrow} \varepsilon$*, the "reset" rules are needless, and the formula* φ *is set to be* $l_1 \wedge \mathsf{G}\psi \wedge \mathsf{F} halt$*.*

10.7 Summary

The (un)decidability borderlines of the model checking problems discussed (and depicted in Figure 10.3 and Figure 10.4) are direct consequences of the following results:

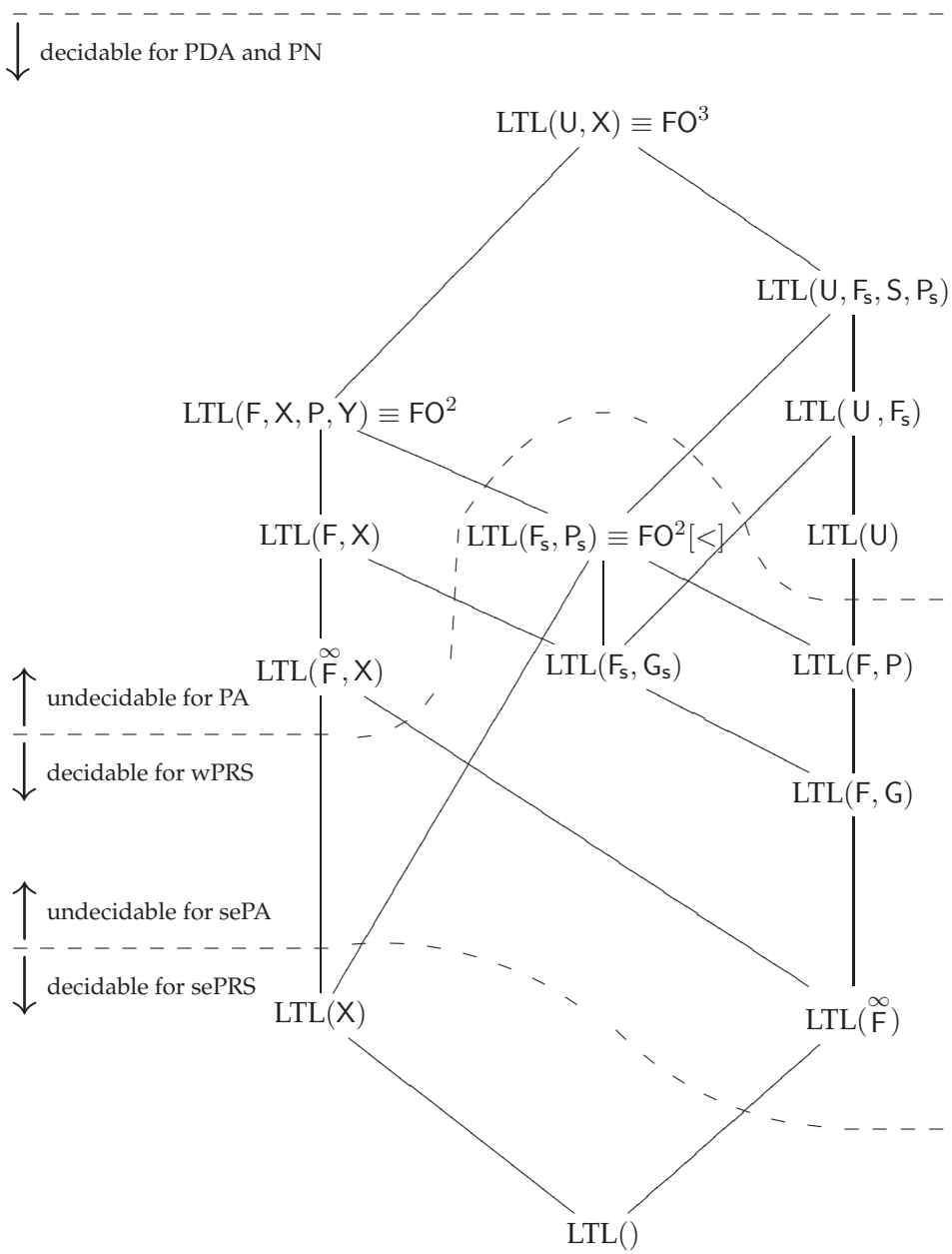


Figure 10.3: The hierarchy of basic LTL fragments with model checking (un)decidability boundaries.

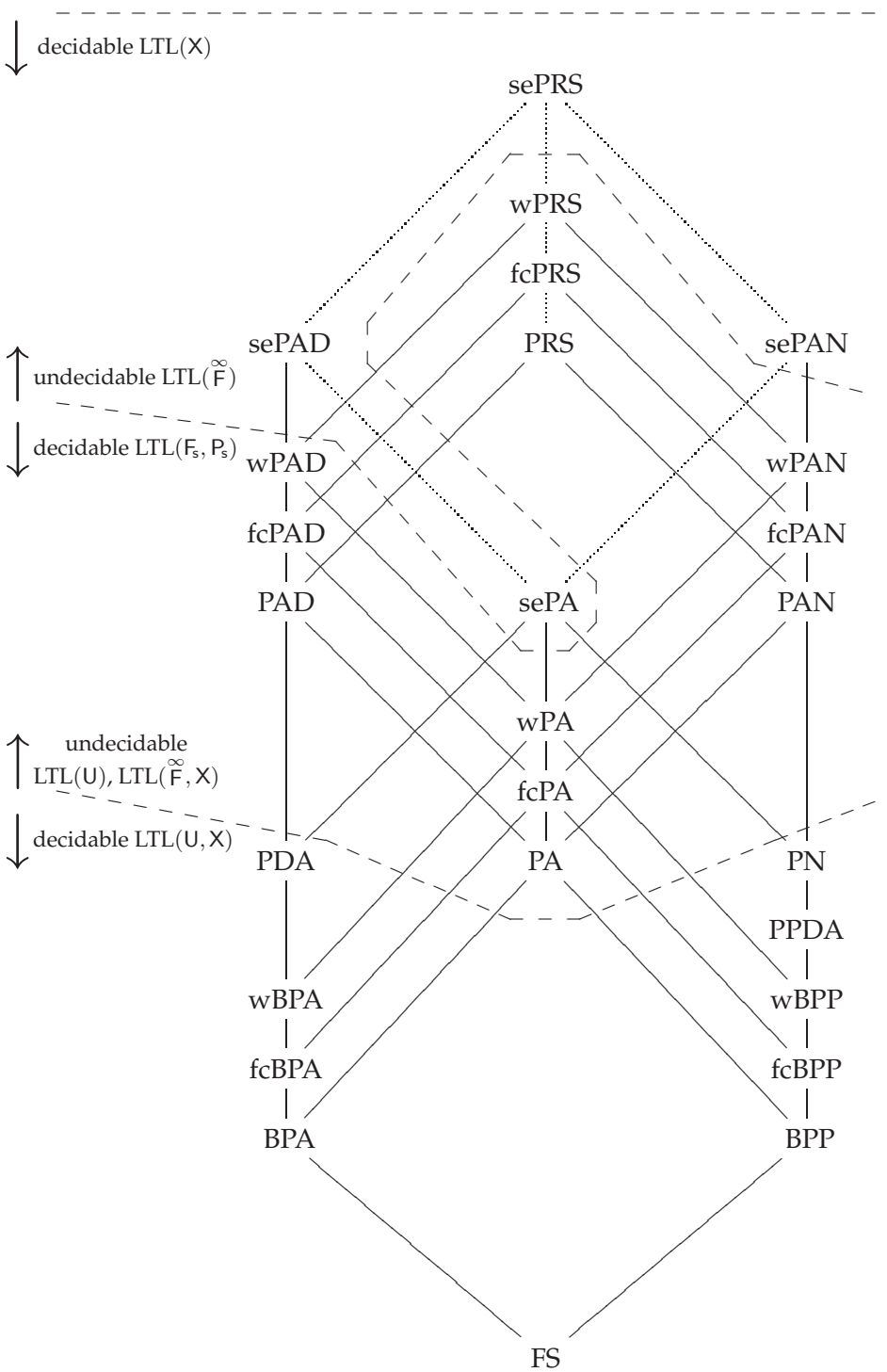


Figure 10.4: The extended PRS-hierarchy with (un)decidability boundaries of the discussed fragments of LTL.

- model checking for *sePRS and LTL*(X) *is decidable* (Remark 10.43),
- model checking for *sePA and LTL*$(\overset{\infty}{\mathsf{F}})$ *is undecidable* ([BEH95]),
- model checking for *wPRS and LTL*$(\mathsf{F}_s, \mathsf{P}_s)$ *is decidable* (Corollary 10.36),
- model checking for *PA and LTL*(U) *is undecidable* (Theorem 10.39),
- model checking for *PA and LTL*$(\overset{\infty}{\mathsf{F}}, \mathsf{X})$ *is undecidable* (Theorem 10.40),
- model checking for *PDA and LTL*(U, X) *is decidable* ([BEM97]), and
- model checking for *PN and LTL*(U, X) *is decidable* ([Esp94]).

We note that the (un)decidability borderlines remain the same even if we restrict the model checking problem on infinite runs (see Remark 10.41). The model checking problem of finite runs differs from its infinite counterpart only in the case of PA systems and LTL$(\overset{\infty}{\mathsf{F}}, \mathsf{X})$ for which it is decidable (see Remark 10.42, Remark 10.44, and Remark 10.45).

10.8 Conclusion

We have established the decidability boundaries of model checking of wPRS classes and basic fragments of LTL (see Figure 10.3). Namely, we have shown that the model checking problem of wPRS against LTL$(\mathsf{F}_s, \mathsf{P}_s)$ is decidable, while the same problem for the PA class and the fragments LTL(U) and LTL$(\overset{\infty}{\mathsf{F}}, \mathsf{X})$ respectively are undecidable. To that end, only two positive results on LTL model checking of PA (and classes subsuming PA) have been published: decidability of model checking of infinite runs for PRS and LTL fragment of fairness properties [Boz05] and decidability of the same problem for PA and *simple PLTL*$_\Box$ [BH96]. Note that the fairness fragment and the regular part of simple PLTL$_\Box$ are strictly less expressive than LTL(F, G) (also known as Lamport logic), which is again strictly less expressive than LTL$(\mathsf{F}_s, \mathsf{G}_s)$.

It is also worth mentioning that our proof techniques differ from those used in [Boz05] and [BH96]. The decidability proof for LTL$(\mathsf{F}_s, \mathsf{G}_s)$ is based on the auxiliary result saying that model checking for wPRS and negated $\mathcal{A}$ fragment is decidable. Moreover, this technique was also successfully adapted to show decidability of model checking for wPRS and LTL$(\mathsf{F}_s, \mathsf{P}_s)$, see Section 10.5. We also emphasise that our positive result for LTL$(\mathsf{F}_s, \mathsf{P}_s)$ deals with both finite and infinite runs, and with wPRS rather than PRS or PA only. Moreover, we show also decidability of the pointed model checking problem for wPRS and LTL$(\mathsf{F}_s, \mathsf{P}_s)$.

The fragment LTL($\mathsf{F}_s, \mathsf{P}_s$) semantically coincides with formulae of First-Order Monadic Logic of Order containing at most 2 variables and no successor predicate ($\mathsf{FO}^2[<]$) [EVW02]. Therefore, our results positively answer also the model checking question for wPRS and $\mathsf{FO}^2[<]$. Let us moreover note that First-Order Monadic Logic of Order containing at most 2 variables (FO^2) coincides with an LTL($\mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y}$) fragment [EVW02]. Due to our undecidability result for model checking PA and LTL($\overset{\infty}{\mathsf{F}}, \mathsf{X}$), we conclude that FO^2 model checking problem is undecidable even for the PA systems. For the sake of completeness we note that First-Order Monadic Logic of Order containing at most 3 variables (FO^3) coincides with the set of all LTL formulae [Kam68, GPSS80]. Hence, the FO^3 model checking problem is decidable PDA and PN systems but it stays undecidable for PA systems.

Chapter 11

$\mathrm{LTL}^{\mathrm{det}}$ Model Checking

This chapter is devoted to model checking problems for $\mathrm{LTL}^{\mathrm{det}}$ and systems of the extended PRS-hierarchy. $\mathrm{LTL}^{\mathrm{det}}$ [Mai00] is a common fragment of CTL and LTL. Using some results of the previous chapter we show that the model checking problem for wPRS and $\mathrm{LTL}^{\mathrm{det}}$ is decidable.

11.1 Motivation

$\mathrm{LTL}^{\mathrm{det}}$ was introduced by Maidl [Mai00] as a syntactic restriction of LTL. The same paper shows that on the semantic level $\mathrm{LTL}^{\mathrm{det}}$ coincides with a logic of all formulae that can be expressed in both LTL and CTL. Moreover, the author also proves that $\mathrm{LTL}^{\mathrm{det}}$ formulae describe exactly those languages the negations of which can be represented by 1-weak Büchi automata (every loop in its transition system is a self loop).

In [BH96], it was shown that model checking problem for the PA class and a logic called *simple* $PLTL_{\Box}$ is decidable. In the same paper, Bouajjani and Habermehl also show that simple $\mathrm{PLTL}_{\Box}$ can express all complements of *simple ω-regular languages* – how they call languages recognised by 1-weak Büchi automata. Therefore, combining the results of [BH96] and [Mai00] we can deduce decidability of the model checking problem for $\mathrm{LTL}^{\mathrm{det}}$ and PA.

As $\mathrm{LTL}^{\mathrm{det}}$ can express some reachability formulae, e.g. $(\neg halt)\,\mathsf{U}\,halt$, the $\mathrm{LTL}^{\mathrm{det}}$ model checking problem is undecidable for all Turing powerful classes, i.e. sePA and all its superclasses.

Using some of the ideas of [Mai00] and decidability of the model checking problem for $\mathrm{LTL}(\mathsf{F}_s, \mathsf{G}_s)$ and wPRS, we show that the model checking problem for wPRS and $\mathrm{LTL}^{\mathrm{det}}$ is decidable. Note that $\mathrm{LTL}^{\mathrm{det}}$ is semantically incomparable with $\mathrm{LTL}(\mathsf{F}_s, \mathsf{G}_s)$.

11.2 Definition of the Studied Problem

To define LTL^{det}, we use the (common) LTL modalities X, U, F, G, etc. as they have been defined in the previous chapter (see Definition 10.4 and Figure 10.1). In addition, we introduce another LTL modality W.

Definition 11.1. Weak until *modality* W *is defined using* U *and* G *as*

$$\varphi \,\mathsf{W}\, \psi = \mathsf{G}\varphi \;\vee\; \varphi \,\mathsf{U}\, \psi \,.$$

Now, we can define the LTL fragment LTL^{det} [Mai00].

Definition 11.2. *Let* $Act = \{a, b, \cdots\}$ *be a countably infinite set of atomic actions. The syntax of* LTL^{det} formula *is defined as follows.*

$$\begin{array}{rl} \varphi \;::=\; & p \mid \varphi_1 \wedge \varphi_2 \mid (p \wedge \varphi_1) \vee (\neg p \wedge \varphi_2) \mid \mathsf{X}\varphi_1 \mid \\ & (p \wedge \varphi_1)\,\mathsf{U}\,(\neg p \wedge \varphi_2) \mid (p \wedge \varphi_1)\,\mathsf{W}\,(\neg p \wedge \varphi_2), \end{array}$$

where

$$p \;::=\; tt \mid a \mid \neg p_1 \mid p_1 \wedge p_2,$$

and a *ranges over* Act*.*

The semantics of LTL^{det} formulae is interpreted over both finite and infinite words of actions in the same way as it was defined in Definition 10.4 in the previous chapter.

Weak Automaton

In this chapter, we make use of Büchi automata to represent LTL formulae. We define an automaton as follows.

Definition 11.3. *An* automaton[1] $A = (Q, \Sigma, q_0, R, F)$ *over the alphabet* Σ *consists of:*

- *the finite set of states* Q*,*
- *the initial state* $q_0 \in Q$*,*
- *the transition relation* $R \subseteq Q \times \Sigma \times Q$*, and*
- *the set of accepting state* $F \subseteq Q$*.*

A weak automaton *is an automaton such that there is a partial ordering* $\geq$ *on the states of* Q *such that every* q *and* q' *for which* $(q, q') \in R$*, we have* $q \geq q'$*.*

[1] An automaton differs from (LTS generated by) FS class just by having accepting states. The same is true for a weak automaton and a weak FSU.

A *run* of A over a word $u = u(0)u(1)u(2)\ldots \in \Sigma^* \cup \Sigma^\omega$ is a sequence $\sigma = q(0)q(1)q(2)\ldots \in Q^+ \cup Q^\omega$ such that $q(0) = q_0$ and for all $i \geq 0, (q(i), u(i), q(i+1)) \in R$. An infinite run $q(0)q(1)q(2)\ldots$ is *successful* if and only if $\{q \in Q \mid q = q(i)$ for infinitely many i $\} \cap F \neq \emptyset$. A finite run $q(0)q(1)q(2)\ldots q(n)$ is *successful* if and only if $q(n) \in F$.

Let us note, that the 1-weak Büchi automaton of [Mai00], we have mentioned in Section 11.1, is the weak automaton of our definition using Büchi acceptance condition. As we consider the automaton on both finite and infinite runs, we call it simply a weak automaton.

Remark 11.4. *A run σ of an automaton $A = (Q, \Sigma, q_0, R, F)$ is successful if and only if $\sigma \models \mathsf{GF}(\bigvee_{q \in F} q)$ where the atomic actions ranges over Q.*

An automaton A *accepts* a word u if there is a successful run of A over u. An automaton $A = (Q, \Sigma, q_0, R, F)$ *represents* an LTL formula φ *over* an alphabet Σ if A accepts exactly the words over Σ that satisfy φ, i.e. all models of φ over Σ.

LTL$^{\text{det}}$ Model Checking Problem

Let $\mathcal{C}$ be a class of systems.

Problem:	*LTL^{det} model checking problem for $\mathcal{C}$*
Instance:	An LTL$^{\text{det}}$ formula φ and a system $\Delta \in \mathcal{C}$
Question:	Is the system Δ a model of the formula φ, i.e. $L(\Delta) \models \varphi$?

11.3 Proof Construction

A basic idea how to prove decidability of the model checking problem for wPRS and LTL$^{\text{det}}$ is as follows. In [Mai00] it has been shown that a negation of a given LTL$^{\text{det}}$ formula can be represented by a weak automaton. In Lemma 11.5 we refine the proof a bit to hold for finite runs as well. Having an automaton representing a negation of a given formula, we construct an synchronous product of the automaton and a given wPRS. As a formula representing the automaton is weak, the product is a weakly extended PRS as well. Now we focus on runs that are successful for the formula representing automaton (a component of the product for now). Due to the construction given in the proof of Lemma 11.5, there is such a run if and only if there is a run in the given wPRS violating the given formula, i.e. the given wPRS does not model the formula (Lemma 11.6). Due to Remark 11.4, we can use the decidability result for the LTL($\mathsf{F_s}, \mathsf{G_s}$) fragment for checking existence of such a successful run and prove the main theorem of this section (Theorem 11.7).

Lemma 11.5. *Let φ be an LTL^{det} formula and Σ be a finite alphabet of actions. There is a weak automaton $A_{\neg\varphi}$ representing $\neg\varphi$ over Σ.*

Proof. The proof is based on induction on structure of φ. We construct an automaton $A_{\neg\varphi} = (Q, \Sigma, q, R, F)$ for every case of structure type of φ, assuming that automata for its proper subformulae can be constructed.

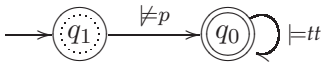
Case p: We construct

$$A_{\neg p} = (\{q_1, q_0\}, \Sigma, q_1, R, F),$$

where

- $R = \{(q_1, a, q_0) \mid a \in \Sigma \wedge a \not\models p\} \cup \{(q_0, a, q_0) \mid a \in \Sigma\}$.
- If $\varepsilon \models \neg p$ then $F = \{q_1, q_0\}$; otherwise, $F = \{q_0\}$.

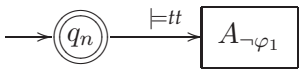
The automaton $A_{\neg p}$ can be depicted as follows.



Case $\mathsf{X}\varphi_1$: Let $A_{\neg\varphi_1} = (Q', \Sigma, q', R', F')$ and $n = |Q'|$. We construct

$$A_{\neg(\mathsf{X}\varphi_1)} = (Q' \cup \{q_n\}, \Sigma, q_n, R, F' \cup \{q_n\}),$$

where $R = \{(q_n, a, q') \mid a \in \Sigma\} \cup R'$ and q_n is fresh. The automaton $A_{\neg(\mathsf{X}\varphi_1)}$ can be depicted as follows.



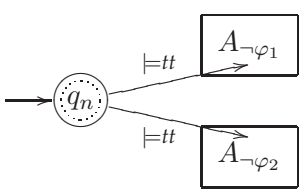
Case $\varphi_1 \wedge \varphi_2$: Let $A_{\neg\varphi_1} = (Q', \Sigma, q', R', F')$, $A_{\neg\varphi_2} = (Q'', \Sigma, q'', R'', F'')$, $Q'' = \{q_{|Q'|+|Q''|-1}, \ldots, q_{|Q'|}\}$, and $n = |Q'| + |Q''|$. We construct

$$A_{\neg(\varphi_1 \wedge \varphi_2)} = (Q' \cup Q'' \cup \{q_n\}, \Sigma, q_n, R, F),$$

where

- $R = \{(q_n, a, q) \mid (q', a, q) \in R' \vee (q'', a, q) \in R''\} \cup R' \cup R''$.
- If $q' \in F'$ or $q'' \in F''$ then $F = \{q_n\} \cup F' \cup F''$;
  otherwise, $F = F' \cup F''$.

The automaton $A_{\neg(\varphi_1 \wedge \varphi_2)}$ can be depicted as follows.



Case $(p \wedge \varphi_1) \vee (\neg p \wedge \varphi_2)$: As $(p \wedge \varphi_1) \vee (\neg p \wedge \varphi_2) = (\neg p \vee \varphi_1) \wedge (p \vee \varphi_2)$, we can partially adopt the previous construction. Therefore, we get

$$\neg((p \wedge \varphi_1) \vee (\neg p \wedge \varphi_2)) = (p \wedge \neg\varphi_1) \vee (\neg p \wedge \neg\varphi_2)$$

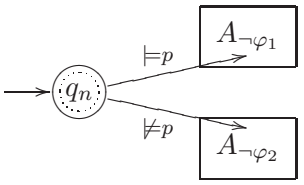
and the construction required is now quite straightforward. Let $A_{\neg\varphi_1} = (Q', \Sigma, q', R', F')$, $A_{\neg\varphi_2} = (Q'', \Sigma, q'', R'', F'')$, $Q'' = \{q_{|Q'|+|Q''|-1}, \ldots, q_{|Q'|}\}$, and n be equal to $|Q'| + |Q''|$. We construct

$$A_{\neg((p\wedge\varphi_1)\vee(\neg p\wedge\varphi_2))} = (Q' \cup Q'' \cup \{q_n\}, \Sigma, q_n, R, F),$$

where

- $R = R' \cup \{(q_n, a, q) \mid a \not\models p \wedge (q', a, q) \in R'\} \cup$
  $R'' \cup \{(q_n, a, q) \mid a \models p \wedge (q'', a, q) \in R''\}$.
- If $q' \in F' \wedge \varepsilon \models p$ or $q'' \in F'' \wedge \varepsilon \models \neg p$ then $F = \{q_n\} \cup F' \cup F''$; otherwise, $F = F' \cup F''$.

The automaton $A_{\neg((p\wedge\varphi_1)\vee(\neg p\wedge\varphi_2))}$ can be depicted as follows.



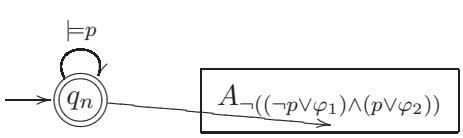
Case $(p \wedge \varphi_1)\,\mathsf{U}\,(\neg p \wedge \varphi_2)$: As $\neg((p\wedge\varphi_1)\,\mathsf{U}\,(\neg p\wedge\varphi_2)) = p\,\mathsf{W}\,((p\wedge\neg\varphi_1)\vee(\neg p\wedge\neg\varphi_2)) = p\,\mathsf{W}\,(\neg((\neg p \vee \varphi_1) \wedge (p \vee \varphi_2)))$, the construction can be done as follows. Applying the previous constructions we obtain $A_{\neg((\neg p\vee\varphi_1)\wedge(p\vee\varphi_2))} = (Q', \Sigma, q', R', F')$. Let $n = |Q'|$. We construct

$$A_{\neg(p\wedge\varphi_1)\,\mathsf{U}\,(\neg p\wedge\varphi_2)} = (Q' \cup \{q_n\}, \Sigma, q_n, R, F),$$

where

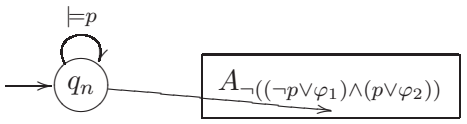
- $R = \{(q_n, a, q_n) \mid a \in \Sigma \wedge a \models p\} \cup \{(q_n, a, q) \mid (q', a, q) \in R'\} \cup R'$ and
- $F = \{q_n\} \cup F'$.

The automaton $A_{\neg((p\wedge\varphi_1)\,\mathsf{U}\,(\neg p\wedge\varphi_2))}$ can be depicted as follows.



Case $(p \wedge \varphi_1)\,\mathsf{W}\,(\neg p \wedge \varphi_2)$: As $\neg((p \wedge \varphi_1)\,\mathsf{W}\,(\neg p \wedge \varphi_2)) = p\,\mathsf{U}\,((p \wedge \neg\varphi_1) \vee (\neg p \wedge \neg\varphi_2)) = p\,\mathsf{U}\,(\neg((\neg p \vee \varphi_1) \wedge (p \vee \varphi_2)))$, the construction is similar to

the case $(p \wedge \varphi_1)\,\mathsf{U}\,(\neg p \wedge \varphi_2)$. Here, $A_{\neg\varphi}$ is the same as in the previous case, except that $F = F'$. Hence, the automaton $A_{\neg((p\wedge\varphi_1)\,\mathsf{W}\,(\neg p\wedge\varphi_2))}$ can be depicted as follows.



□

Lemma 11.6. *Let Δ be a wPRS and φ be an LTL^{det} formula. We can construct a wPRS Δ' and a set of actions $\mathcal{F} \subseteq Act(\Delta')$ such that*

$$\Delta \models \neg\varphi \quad \textit{if and only if} \quad \Delta' \models \bigvee_{a\in\mathcal{F}} \mathsf{GF}a$$

.

Proof. Let $\Delta = (M, \sqsupseteq, R, p_0, X_0)$. Due to Lemma 11.5, a weak automaton $A_{\neg\varphi} = (Q, Act(\Delta), q_{|Q|-1}, R'', F'')$ representing the negation of the given LTL$^{\text{det}}$ formula φ over $Act(\Delta)$ can be constructed. In the following, we construct a wPRS Δ' representing a synchronous product of the given wPRS Δ and the weak automaton $A_{\neg\varphi}$. The product is synchronised on the labels of actions. For the purpose of looking up the successful runs, we label actions of the product by the states of $A_{\neg\varphi}$.

We set $\mathcal{F}$ to be F'' and $\Delta' = (M', \sqsupseteq', R', p', X_0)$, where

- $M' = \{p'\} \cup ((Q \cup \{q'\}) \times M)$ where p' and q' are fresh,
- $p' \sqsupseteq' (q_i, p) \sqsupseteq' (q_j, r) \sqsupseteq' (q', p) \sqsupseteq' (q', r)$ where $i \geq j$ and $p \sqsupseteq r$,
- $Const(\Delta') = Const(\Delta)$, and
- $R' = \{p'X_0 \overset{q_{|Q|-1}}{\hookrightarrow} (q_{|Q|-1}, p_0)X_0)\} \cup$
  $\{(q_i, p)t_1 \overset{q'}{\hookrightarrow} (q', r)t_2 \mid q_i \in Q \wedge pt_1 \overset{a}{\hookrightarrow} rt_2 \in R\} \cup$
  $\{(q_i, p)t_1 \overset{q_j}{\hookrightarrow} (q_j, r)t_2) \mid (q_i, a, q_j) \in R'' \wedge pt_1 \overset{a}{\hookrightarrow} rt_2 \in R\}.$

From the construction, it follows that there is a run violating φ in Δ if and only if there is a run in Δ' that is successful for the component representing the formula. Due to Remark 11.4, the lemma holds. □

Theorem 11.7. *The model checking problem for wPRS and LTL^{det} is decidable.*

Proof. The theorem follows directly from Lemma 11.6 and decidability of the LTL($\mathsf{F_s}, \mathsf{G_s}$) model checking problem for wPRS (see Corollary 10.25). □

11.4 Summary

It was easily deducible from [BH96] and [Mai00], that the model checking problem for PA and the common fragment of CTL and LTL called LTL^{det} is decidable. In this chapter, we have shown that the LTL^{det} model checking problem (of both *infinite runs* and *finite runs*) is decidable also for wPRS. Our proof use some ideas of [Mai00] and decidability of the model checking problem for $\text{LTL}(\mathsf{F_s}, \mathsf{G_s})$ and wPRS (see Corollary 10.25). As LTL^{det} can express some reachability formulae e.g. $(\neg halt)\,\mathsf{U}\,halt$, the LTL^{det} model checking problem is undecidable for all Turing powerful classes, i.e. sePA and all its superclasses. Therefore, the (un)decidability boundaries of the LTL^{det} model checking problem are established for the extended PRS-hierarchy, see Figure 11.1.

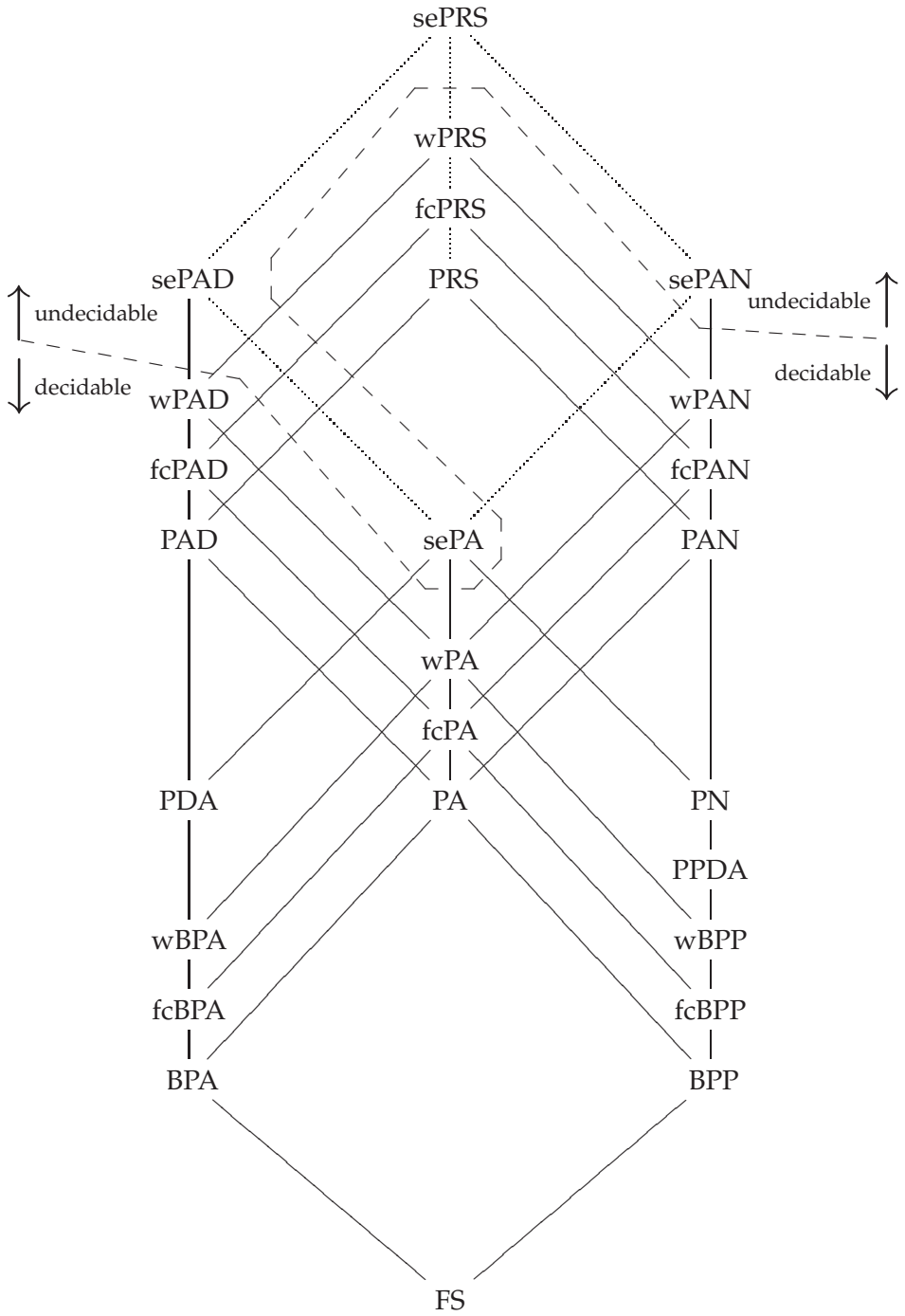


Figure 11.1: The extended PRS-hierarchy with (un)decidability boundaries of LTL^{det} model checking problem.

Chapter 12

Conclusion and Future Work

This thesis provides an overview of extensions of Process Rewrite System formalism (see Chapters 2–5). We construct a hierarchy of this formalisms with respect to the strong bisimulation equivalence. Within this hierarchy we examined various (un)decidability questions of interesting verification problems such as bisimulation equivalence (see Chapters 6 and 7) and model checking problems (see Chapters 8–11). On the one hand, we can say that all the borderlines related to all the discussed model checking problems are established for now. On the other hand, a lot of questions remain open in the area of equivalence checking, namely deciding bisimulation equivalence, for example strong bisimulation problem for fcBPP, wBPP, PA, fcPA, wPA, PAD, fcPAD, and wPAD (see Section 6.1) and weak bisimulation problem for BPA and BPP (see Section 7.5). Moreover, in this field there is a lot of important open problems that has not been objected in this thesis, for example strong/weak bisimilarity with finite state systems and strong/weak regularity as well as upper and lower complexity bounds of all the problems mentioned above.

Bibliography

[BAPM83] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20(3):207–226, 1983.

[BCMS01] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elsevier Science Publishers, 2001.

[BCS95] O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, volume 969 of *Lecture Notes in Computer Science*, pages 423–433. Springer-Verlag, 1995.

[BCS96] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 247–262. Springer-Verlag, 1996.

[BE97] O. Burkart and J. Esparza. More infinite results. *Electronic Notes in Theoretical Computer Science*, 5, 1997.

[BEH95] A. Bouajjani, R. Echahed, and P. Habermehl. On the verification problem of nonregular properties for nonregular processes. In *Proceedings of IEEE Symposium on Logic in Computer Science (LICS'95)*. IEEE Computer Society Press, 1995.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer-Verlag, 1997.

[BET03] A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. *Interna-*

tional Journal on Foundations of Computer Science, 14(4):551–582, 2003.

[BH96] A. Bouajjani and P. Habermehl. Constrained properties, semilinear systems, and petri nets. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 481–497. Springer-Verlag, 1996.

[BK85] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[BKŘS06] L. Bozzelli, M. Křetínský, V. Řehák, and J. Strejček. On Decidability of LTL Model Checking for Process Rewrite Systems. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, volume 4337 of *Lecture Notes in Computer Science*, pages 248–259. Springer-Verlag, 2006.

[Boz05] L. Bozzelli. Model checking for process rewrite systems and a class of action-based regular properties. In *Proceedings of VMCAI'05*, volume 3385 of *Lecture Notes in Computer Science*, pages 282–297. Springer-Verlag, 2005.

[BST06] A. Bouajjani, J. Strejček, and T. Touili. On Symbolic Verification of Weakly Extended PAD. In *Preliminary Proceedings of the 13th International Workshop on Expressiveness in Concurrency EXPRESS'06*, 2006.

[BT03] A. Bouajjani and T. Touili. Reachability Analysis of Process Rewrite Systems. In *Proceedings of Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, volume 2914 of *Lecture Notes in Computer Science*, pages 74–87. Springer-Verlag, 2003.

[Büc64] J. R. Büchi. Regular canonical systems. *Arch. Math. Logik u. Grundlagenforschung*, 6:91–111, 1964.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[CHM93] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation is decidable for all basic parallel processes. In *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, 1993.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[EK95] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and basic parallel processes. In *CAV*, volume 939 of *Lecture Notes in Computer Science*, pages 353–366. Springer-Verlag, 1995.

[EP00] J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *Proceedings of the 27th Symposium on Principles of Programming Languages (POPL'00)*, pages 1–11. ACM press, 2000.

[Esp94] J. Esparza. On the decidability of model checking for several mu-calculi and petri nets. In *CAAP*, volume 787 of *Lecture Notes in Computer Science*, pages 115–129. Springer-Verlag, 1994.

[Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.

[Esp02] J. Esparza. Grammars as processes. In *Formal and Natural Computing*, volume 2300 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[EVW02] K. Etessami, M. Y. Vardi, and Th. Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279–295, 2002.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Conference Record of the 7th Annual ACM Symposium on Principles of Programming Languages (POPL'80)*, pages 163–173. ACM press, 1980.

[Hab97] P. Habermehl. On the complexity of the linear-time μ-calculus for Petri nets. In *Proceedings of ICATPN'97*, volume 1248 of *Lecture Notes in Computer Science*, pages 102–116. Springer-Verlag, 1997.

[Hir] Y. Hirshfeld. Methods and tools for the verification of infinite state systems. Private communication.

[Hoa80] C. A. R. Hoare. Communicating Sequential Processes. In *On the connstruction of programs – an advanced course*, pages 229–254. Cambridge University Press, 1980.

[HS05] H. Hüttel and J. Srba. Recursion vs. replication in simple cryptographic protocols. In *Proceedings of SOFSEM 2005: Theory and Practice of Computer Science*, volume 3381 of *Lecture Notes in Computer Science*, pages 178–187. Springer-Verlag, 2005.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1979.

[Jan95a] P. Jančar. High Undecidability of Weak Bisimilarity for Petri Nets. In *Proc. of TAPSOFT*, volume 915 of *Lecture Notes in Computer Science*, pages 349–363. Springer-Verlag, 1995.

[Jan95b] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995.

[Jan03] P. Jančar. Strong bisimilarity on basic parallel processes is PSPACE-complete. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 218–227. IEEE Computer Society Press, 2003.

[JEM99] P. Jančar, J. Esparza, and F. Moller. Petri nets and regular processes. *Journal of Computer and System Sciences*, 59(3):476–503, 1999.

[JKM01] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theoretical Computer Science*, 258:409–433, 2001.

[JM95] P. Jancar and F. Moller. Checking regular properties of petri nets. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *Lecture Notes in Computer Science*, pages 348–362. Springer-Verlag, 1995.

[JS04] P. Jančar and J. Srba. Highly undecidable questions for process algebras. In *Proceedings of the 3rd IFIP International Conference on Theoretical Computer Science (TCS'04)*, Exploring New Frontiers of Theoretical Informatics, pages 507–520. Kluwer Academic Publishers, 2004.

[Kam68] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[KM02] A. Kučera and R. Mayr. On the complexity of semantic equivalences for pushdown automata and BPA. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS'02)*, volume 2420 of *Lecture Notes in Computer Science*, pages 433–445. Springer-Verlag, 2002.

[KŘS04a] M. Křetínský, V. Řehák, and J. Strejček. Extended process rewrite systems: Expressiveness and reachability. In *Proceedings of 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 355–370. Springer-Verlag, 2004.

[KŘS04b] M. Křetínský, V. Řehák, and J. Strejček. On extensions of process rewrite systems: Rewrite systems with weak finite-state unit. In *Proceedings of INFINITY 2003, the 5th International Workshop on Verification of Infinite-State Systems, a satellite workshop of CONCUR 2003*, volume 98 of *Electronic Notes in Theoretical Computer Science*, pages 75–88. Elsevier Science Publishers, 2004.

[KŘS05] M. Křetínský, V. Řehák, and J. Strejček. Reachability of Hennessy-Milner properties for weakly extended PRS. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science*, volume 3821 of *Lecture Notes in Computer Science*, pages 213–224. Springer-Verlag, 2005.

[KŘS06] M. Křetínský, V. Řehák, and J. Strejček. Refining the Undecidability Border of Weak Bisimilarity. In *Proceedings of the 7th International Workshop on Verification of Infinite-State Systems (INFINITY'05)*, volume 149 of *Electronic Notes in Theoretical Computer Science*, pages 17–36. Elsevier Science Publishers, 2006.

[KS04] A. Kučera and Ph. Schnoebelen. A general approach to comparing infinite-state systems with their finite-state specifications. In *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 371–386. Springer-Verlag, 2004.

[Lip76] R. Lipton. The reachability problem is exponential-space hard. Technical Report 62, Department of Computer Science, Yale University, 1976.

[LS98] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 50–66. Springer-Verlag, 1998.

[Mai00] M. Maidl. The common fragment of CTL and LTL. In *Proceedings of the 41th Annual Symposium on Foundations of Computer Science (FOCS'00)*, pages 643–652, 2000.

[May81] E. W. Mayr. An algorithm for the general Petri net reachability problem. In *Proceedings of 13th Symposium on Theory of Computing*, pages 238–246. ACM press, 1981.

[May84] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.

[May97a] R. Mayr. Combining Petri nets and PA-processes. In *Proceedings of TACS'97: International Symposium on Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 547–561. Springer-Verlag, 1997.

[May97b] R. Mayr. Process rewrite systems, in proceedings of EXPRESS'97. *Electronic Notes in Theoretical Computer Science*, 7 (1997):185–205, 1997.

[May98] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, Technische Universität München, 1998.

[May00] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.

[May05] R. Mayr. Weak bisimilarity and regularity of context-free processes is EXPTIME-hard. *Theoretical Computer Science*, 330(3):553–575, 2005.

[McM93] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Min67] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[Mol96] F. Moller. Infinite results. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer-Verlag, 1996.

[Mol98] F. Moller. A taxonomy of infinite state processes. In *Proceedings of International Symposium on Mathematical Foundations of Computer Science (MFCS'98)*, volume 18 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1998.

[MR98] R. Mayr and M. Rusinowitch. Reachability is decidable for ground AC rewrite systems. In *Proceedings of INFINITY'98 workshop*, 1998.

[MS85] D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[MSS92] D. Muller, A. Saoudi, and P. Schupp. Alternating automata, the weak monadic theory of trees and its complexity. *Theoretical Computer Science*, 97(2):233–244, 1992.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *Proceedings of 5th GI-Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modelling of Systems*. Prentice-Hall, 1981.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS'77)*, pages 46–57, 1977.

[Rei85] W. Reisig. *Petri Nets—An Introduction*. Springer-Verlag, 1985.

[Sén98] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS'98)*, pages 120–129. IEEE Computer Society Press, 1998.

[SR90] V. A. Saraswat and M. Rinard. Concurrent constraint programming. In *Proceedings of 17th Symposium on Principles of Programming Languages (POPL'90)*, pages 232–245. ACM press, 1990.

[Srb02a] J. Srba. Roadmap of infinite results. *EATCS Bulletin*, (78):163–175, 2002. `http://www.brics.dk/~srba/roadmap/`.

[Srb02b] J. Srba. Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard. In *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 535–546. Springer-Verlag, 2002.

[Srb02c] J. Srba. Strong bisimilarity and regularity of basic process algebra is PSPACE-hard. In *Proceedings of 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *Lecture Notes in Computer Science*, pages 716–727. Springer-Verlag, 2002.

[Srb02d] J. Srba. Undecidability of weak bisimilarity for pushdown processes. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *Lecture Notes in Computer Science*, pages 579–593. Springer-Verlag, 2002.

[Srb03a] J. Srba. Complexity of weak bisimilarity and regularity for BPA and BPP. *Mathematical Structures in Computer Science*, 13:567–587, 2003.

[Srb03b] J. Srba. Undecidability of weak bisimilarity for PA-processes. In *Proceedings of the 6th International Conference on Developments in Laguage Theory (DLT'02)*, volume 2450 of *Lecture Notes in Computer Science*, pages 197–208. Springer-Verlag, 2003.

[Srb04] J. Srba. *Roadmap of Infinite results*, volume Vol 2: Formal Models and Semantics. World Scientific Publishing Co., 2004.

[Sti96] C. Stirling. Modal and temporal logics for processes. In F. Moller and G. M. Birtwistle, editors, *Banff Higher Order Workshop 1995*, volume 1043 of *Lecture Notes in Computer Science*, pages 149–237. Springer-Verlag, 1996.

[Str02] J. Strejček. Rewrite systems with constraints, in proceedings of EXPRESS'01. *Electronic Notes in Theoretical Computer Science*, 52, 2002.

[Str04] J. Strejček. *Linear Temporal Logic: Expressiveness and Model Checking*. PhD thesis, Faculty of Informatics, Masaryk University in Brno, 2004.

[vG93] R. J. van Glabbeek. The linear time – branching time spectrum II. In *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1993.