Decidability and Complexity Issues for Infinite-State Processes

Jiří Srba

PhD Dissertation

BRICS
Department of Computer Science
University of Aarhus
Denmark

Decidability and Complexity Issues for Infinite-State Processes

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfilment of the Requirements for the
PhD Degree

by
Jiří Srba
January 31, 2003

Abstract

This thesis studies decidability and complexity border-lines for algorithmic verification of infinite-state systems. Verification techniques for finite-state processes are a successful approach for the validation of reactive programs operating over finite domains. When infinite data domains are introduced, most of the verification problems become in general undecidable. By imposing certain restrictions on the considered models (while still including infinite-state spaces), it is possible to provide algorithmic decision procedures for a variety of properties. Hence the models of infinite-state systems can be classified according to which verification problems are decidable, and in the positive case according to complexity considerations.

This thesis aims to contribute to the problems mentioned above by studying decidability and complexity questions of equivalence checking problems, i.e., the problems whether two infinite-state processes are equivalent with regard to some suitable notion of behavioural equivalence. In particular, our interest is focused on strong and weak bisimilarity checking within classical models of infinite-state processes.

Throughout the thesis, processes are modelled by the formalism of process rewrite systems which provides a uniform classification of all the considered classes of infinite-state processes.

We begin our exposition of infinite-state systems by having a closer look at the very basic model of labelled transition systems. We demonstrate a simple geometrical encoding of the labelled systems into unlabelled ones by transforming labels into linear paths of different lengths. The encoding preserves the answers to the strong bisimilarity checking and is shown to be effective e.g. for the classes of pushdown processes and Petri nets. This gives several decidability and complexity corollaries about unlabelled variants of the models.

We continue by developing a general decidability theorem for commutative process algebras like deadlock-sensitive BPP (basic parallel processes), lossy BPP, interrupt BPP and timed-arc BPP nets. The introduced technique relies on the tableau-based proof of decidability of strong bisimilarity for BPP and we extend it in several ways to provide a wider applicability range of the technique. This, in particular, implies that all the mentioned classes of commutative process algebras allow for algorithmic checking of strong bisimilarity.

Another topic studied in the thesis deals with finding tight complexity estimates for strong and weak bisimilarity checking. A novel technique called existential quantification is explicitly described and used to show that the strong bisimilarity and regularity problems for basic process algebra and basic parallel processes are PSPACE-hard. In case of weak bisimilarity checking of basic parallel processes the problems are shown to be again PSPACE-hard — this time, however, also for the normed subclass.

Finally we consider the problems of weak bisimilarity checking for the classes of pushdown processes and PA-processes. Negative answers to the problems are given — both problems are proven to be undecidable. The proof techniques moreover solve some other open problems. For example the undecidability proof for pushdown processes implies also undecidability of strong bisimilarity for prefix-recognizable graphs.

The thesis is concluded with an overview of the state-of-the-art for strong and weak bisimilarity problems within the classes from the hierarchy of process rewrite systems.

Acknowledgements

First of all I would like to thank my advisor Mogens Nielsen for his help and encouragement throughout my studies. I very appreciated to work under his supervision and I owe so much to him.

Many thanks go also to my former advisors Ivana Černá and Mojmír Křetínský for their support and for introducing me into the study of infinite-state systems.

I would like to thank Marco Carbone, Bartek Klin, Pawel Sobocinski and Frank Valencia for reading some parts of the thesis.

I thankfully acknowledge the support from Danish National Research Foundation for funding my PhD education at BRICS.

I am very grateful to my parents Jaroslava Srbová and Jiří Srba for their constant support and encouragement.

Last but not least, I thank to Vanda Jašíková for her love and for accompanying me during my studies in Denmark.

Jiří Srba,
Århus, January 31, 2003.

Contents

Chapter 1

Introduction

The rapid development of computer technologies has resulted in an increasing number of embedded applications used in present systems. These applications run concurrently, communicate with each other and continuously interact with the environment. They are often called *reactive systems*. Typical examples of these complex systems are safety-critical, mission-critical and business-critical applications in a variety of sectors like aerospace, defence, transportation, nuclear technology, communication and medicine.

Design invariants of such systems must hold under all circumstances. The usual verification approaches like simulation and testing do not provide a satisfactory level of reliability since the involved systems are large and all possible situations cannot be simulated nor exhaustively tested. On the other hand, any violation of the critical invariants is costly.

From the mechanical point of view the thread of *hardware failures* is often minimized by introducing *redundancy*: single component functionality is provided by multiple interchangeable devices in order to cope with possible breakdowns. Redundancy can be also used for detecting and recovering from corruption of data. A typical example is the *cyclic redundancy check* which adds additional information to every manipulated block of data in order to detect errors and corruption during storage or transmission.

The reliability of *software systems* can be also increased by introducing redundancy. As an example three independent teams may be asked to develop a software with the same functionality. The software is then run separately on independent computers and if different answers for a situation that occurred are provided by the systems, some kind of majority voting is applied. However, introduction of such a redundancy can be very pricy and does not always guarantee increased reliability. Citing [118]: "Software failures are latent design errors, and hence are very different from hardware failures. Strategies for mitigating hardware failures, such as duplicative redundancy, are unlikely to work for software.".

Another approach to validate reactive systems is by the use of *formal methods*. These are mathematics–based techniques for the specification, development and verification of software and hardware systems. Nowadays, the need of formal verification techniques is widely accepted. Citing again [118]: "Mathematical verification technology has had a profound effect on commercial digital hardware engineering, where finding errors prior to initial fabrication runs in orders of magnitude more cost-effective than discovering these errors afterwards. This technology includes *equivalence checkers* for combinatorial aspects of digital circuits and *model checkers* for sequential and concurrent aspects of digital hardware systems.".

As mentioned in the citation, two main techniques for system verification are *equivalence checking* and *model checking.* In the equivalence checking approach the intended specification of the system is compared to its actual implementation and equivalence between them is verified. In the model checking approach the crucial system properties are formally described (usually by a formula of a suitable modal or temporal logic) and the implementation (or also the specification) is checked against the properties. The general aim is to provide automatic ways for such system verifications.

This thesis studies the equivalence checking problems for several models of concurrent and sequential processes, in particular for systems with unboundedly many reachable states. The field which is in our focus is also known as *verification of infinite-state processes.* Recent overviews of the field are provided in [34] and [110].

One of the first algebraic theories for communicating processes appeared due to Milner in [133] where he introduced his "Calculus of Communicating Systems" (CCS) and provided mathematical foundations for giving semantics to concurrent processes by means of *observational equivalence* which was later refined by Park [146] into *bisimulation equivalence.*

Bisimulation equivalence (or simply bisimilarity) will also be the main focus of this thesis. In particular, we shall study decidability and complexity issues of bisimilarity checking within several classes of infinite-state systems. The following section provides more details.

1.1 Verification of Infinite-State Processes

Verification of *finite-state* systems has been an important area with successful applications to e.g. communication protocols, hardware structures, mobile phones, hi-fi equipment and many others. Powerful verification techniques were introduced for finite-state systems [47, 48, 199, 102, 144], data structures like BDDs (binary decision diagrams) [30, 32, 132] providing efficient representation of large state spaces were developed, and novel approaches like partial order reduction [192, 147, 62] were invented to deal with the "state explosion" problem.

Recently *infinite-state* systems have drawn much attention. There are several classes of infinite-state processes for which equivalence and model checking problems were studied and in some cases they were shown to be decidable.

The introduction of infiniteness is natural and well motivated. There are many examples of systems that generate infinite-state spaces and these systems can be classified according to the source of infiniteness.

- **Infinite data domains.** The considered systems operate over unbounded data structures like counters, integer variables, lists, trees, stacks and queues.

- **Unbounded control structures.** The source of infiniteness comes from a complex control-flow management. This includes control-flow mechanisms like recursive calls of procedures, dynamic process creation and mobility.

- **Real-time features.** Systems which take into account real-time parameters naturally contain computations over infinite (even uncountable) state spaces. Typical examples are telecommunication protocols, embedded systems and real-time controllers.

- **Parametrized reasoning.** Even though the considered systems may be instantiated to finite ones, they can contain a number of parameters which range over infinite domains (typically integers). These parametric bounds give rise to infiniteness during process verification. Networks with an arbitrary number of identical processes and broadcast protocols serve as characteristic examples.

Out of a wide range of models with infinitely many reachable states let us mention e.g. Petri nets and vector addition systems, several classes of process algebras, timed automata, hybrid automata, counter machines and channel systems.

For such systems that operate over infinite sets new methods must be proposed since the model/equivalence checking problems become significantly harder or even undecidable. Nevertheless, a number of algorithmic methods have been developed for process algebras generating infinite-state systems [137, 37, 34, 110], timed process algebras [204], Petri nets [87], lossy vector addition systems [27], counter machines [91, 2] and real-time systems [5, 7, 8, 114].

1.1.1 Labelled Transition Systems

Perhaps the most abstract process behaviour can be described as follows: a process p performs an action and becomes a process p'. Processes are considered as agents that can execute actions in order to communicate with their environment. These actions can be observed by an external observer and determine the visible behaviour of the process.

This simple idea is formally captured by the notion of *labelled transition systems* [151, 137]. Transition systems with labels are perhaps the most common model within concurrency theory. Processes are understood as nodes of certain edge-labelled oriented graphs (labelled transition systems) and a change of a process state caused by performing an action is understood as moving along an edge labelled by the action name.

Formally, a labelled transition system consists of a set of *states* (*processes*), a set of *labels* (*actions*), and a ternary transition relation $\longrightarrow$ describing a change of a process state: if a process p can perform an action a and become a process p', we write $p \xrightarrow{a} p'$.

Example 1.1. Let us start with the classical example of a tea/coffee vending machine. The very simplified behaviour of the process which determines the interaction of the machine with a costumer can be described as follows. From the initial state representing the situation “waiting for a request”, (let us call the state p), two possible actions are enabled. Either the tea button or the coffee button is pressed (the corresponding action ‘*tea*’ or ‘*coffee*’ is executed) and the state of the control process of the machine changes accordingly to p_1 or p_2. Formally, this can be described by the transitions

$$p \xrightarrow{\textit{tea}} p_1 \quad \text{and} \quad p \xrightarrow{\textit{coffee}} p_2.$$

Now the customer is asked to insert the corresponding amount of money, let us say one euro for a cup of tea and two euros for a cup of coffee. This is reflected in the control state of the vending machine with corresponding changes. It can be modelled by the transitions

$$p_1 \xrightarrow{\text{1€}} p_3 \quad \text{and} \quad p_2 \xrightarrow{\text{2€}} p_3.$$

Finally, the drink is collected and the machine returns to its initial state p, ready to accept another customer. This corresponds to the transition

$$p_3 \xrightarrow{\textit{collect}} p.$$

□

We shall often use a graphical representation of labelled transition systems. The following picture represents the tea/coffee machine described above.

$$\begin{tikzcd}
& \overset{p}{\bullet} \arrow[dl, "\textit{tea}"'] \arrow[dr, "\textit{coffee}"] & \\
\bullet \arrow[dr, "\text{1€}"'] & & \bullet \arrow[dl, "\text{2€}"] \\
& \bullet \arrow[uu, "\textit{collect}"] &
\end{tikzcd}$$

From now on, the notion of a *process* will be equivalent to a rooted labelled transition system: the labelled transition system describes the process behaviour and the root (a selected node of the transition system) represents the initial state of the process.

Remark 1.1. Our definition of a process enables also situations like in the following picture.

$$\begin{tikzcd}
p \arrow[d, bend right, "a"'] & \\
p_1 \arrow[u, bend right, "b"'] & p_2 \arrow[l, "d"'] \arrow[loop right, "c"]
\end{tikzcd}$$

This means that the state p_2 where the action c can be performed in a loop is irrelevant for the behaviour of the process p since p_2 can never be reached from p. This may, however, be a desirable situation as explained later. Nevertheless, processes like p motivate the definition of a set of reachable states. We say that a state p' is *reachable* from p iff there exists an oriented path from p to p'. The set of all such states is called the *set of reachable states*. In our example this set contains exactly two states: p and p_1.

So far we were able to describe only finite-state processes. The following sections of this chapter will show how rooted transition systems with infinitely many reachable states can be defined.

1.1.2 Justification of the Study

In this subsection we shall summarize the main reasons why we consider the study of infinite-state systems interesting.

- The study is focused on conceptual problems in concurrency theory. It isolates a few basic entities and abstracts from secondary features. This enables to formulate the considered problems in a simple and clear way while preserving the interesting features like concurrency and communication. It also provides a deeper understanding of many aspects of parallel and concurrent computing.

- Techniques and algorithms developed for verification of infinite-state systems appear to be succesfully applicable to other fields of computer science. An example of the range of applicability can be given from the classical theory of formal languages. The long standing open problem of language equivalence between deterministic pushdown automata was positively answered by Sénizergues [162, 164]. However, the complete proof exposes over 70 pages and is very intricate. Using the techniques developed in concurrency theory, Stirling [186] significantly simplified the proof and recently provided even a primitive recursive decision algorithm [188]. Another example where verification techniques for infinite-state systems find natural application is e.g. control-flow analysis (see [61]).

- Many of the algorithms discovered in the theoretical study provide a good starting point for a construction of automatic verification tools. A significant number of equivalence and model checking tools already exist. They combine different approaches and have different aims. Some of them are commercial while other are built for academic purposes only. An online overview of current verification tools is available e.g. at

  `http://www.fi.muni.cz/yahoda.`

- Last but not least, the studied problems appear to be easily mathematicaly described and theoreticaly interesting. The theory provides neat and short definitions of problems, while the solutions to the problems usually require novel and involved proof techniques. This is another reason why the study of infinite-state systems is an attractive area for theoretical research.

1.2 Semantics of Processes

As indicated before, we shall identify the notion of a process with rooted labelled transition systems. One of the first questions to be answered is which transition systems should be considered equivalent. The usual language, trace or even isomorphism equivalences do not always seem to be the most accurate, especially when dealing with reactive systems. We need more discriminating notions of equivalences in order to capture properly the behaviour of systems that do not exhibit pure input/output or batch executions but require communication with the environment. In this section we recall some ideas of extending the usual semantic approaches to include features like reactive behaviours.

As suggested by van Glabbeek in [194], most semantic notions of behavioural equivalences can be classified along four different lines.

- **Linear vs. branching time.** The question is to what extend we should identify processes that differ in the branching structure of their possible executions?
- **Interleaving vs. partial order.** The question is to what extend we should identify processes that differ in the causal dependences between their executable actions?
- **Degree of abstraction from unobservable actions.** The question is to what extend we should identify processes that differ in their behaviours under unobservable (silent, internal) actions?
- **Approaches to infinite executions.** The question is to what extend we should identify processes that differ in their infinite behaviours?

In what follows we shall discuss these fundamental classification lines and emphasise which of the approaches are in the focus of this thesis.

1.2.1 Linear vs. Branching Time

In this subsection we describe the linear/branching time hierarchy (spectrum) by van Glabbeek [193, 194]. It contains the most studied behavioural equivalences and illustrates the increasing discriminating power of equivalence notions when more and more branching aspects are taken into consideration. The classical linear/branching time hierarchy is presented in Figure 1.1.

Remark 1.2. All the equivalences in this hierarchy (except for bisimulation equivalence) are usually defined as the symmetric closures of the corresponding preorders. Hence a similar hierarchy of behavioural preorders also exists.

Remark 1.3. In the rest of this section we will assume that the set of actions is *finite* and implicitly given. This is a usual assumption and most of the process algebras studied in the area of automatic verification obey the finiteness

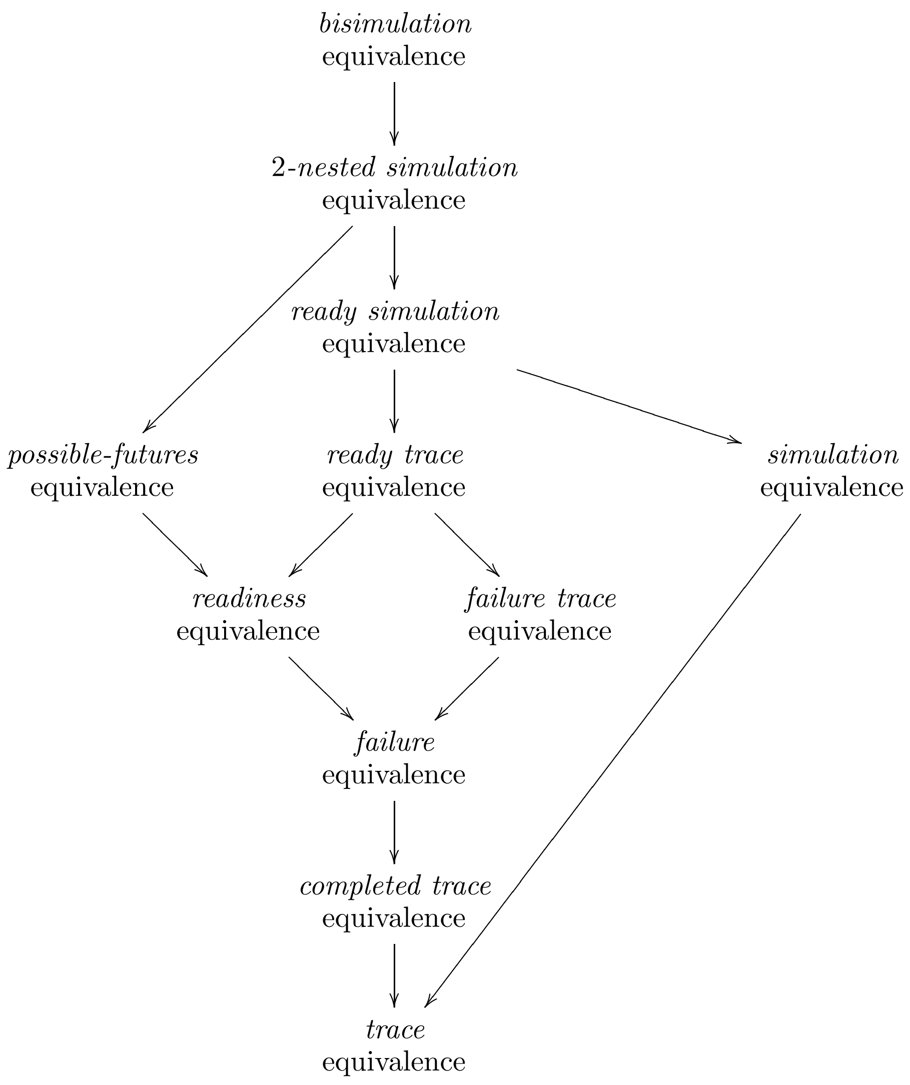

bisimulation equivalence
2-*nested simulation* equivalence
ready simulation equivalence
possible-futures equivalence
ready trace equivalence
simulation equivalence
readiness equivalence
failure trace equivalence
failure equivalence
completed trace equivalence
trace equivalence

Figure 1.1: Linear/branching time hierarchy of behavioural equivalences

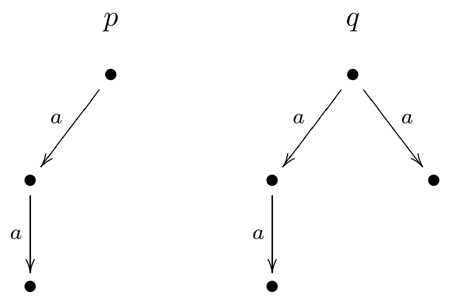
restriction on the number of actions. We also say that an action a is *enabled* in a process p iff p can perform the action a and become a process p'. Otherwise, we say that a is *disabled*.

The coarsest (least discriminating) equivalence in the linear/branching time hierarchy is *trace equivalence*, as defined by Hoare [78]. A (partial) trace of a process (or equivalently of a state in a labelled transition system) is simply any finite sequence of actions that can be performed from the process. Two processes are trace equivalent iff the sets of their traces are equal.

A variant of the trace equivalence is called *completed trace equivalence*, or also *language equivalence*. Here we consider also completed traces, i.e., traces that are maximal in the sense that they cannot be extended to longer ones. Two processes are completed trace equivalent iff they are trace equivalent and moreover they have the same set of completed traces.

Completed trace equivalence is more discriminating than trace equivalence as it is demonstrated by the following example.

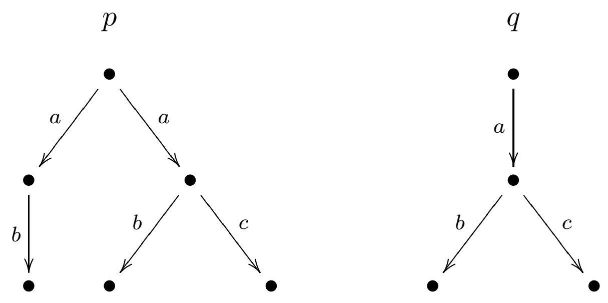
Example 1.2. Let us consider the following two processes p and q.



Obviously, the set of traces of p is $\{\epsilon, a, aa\}$ where ϵ in the empty sequence. Since q has the same set of traces, p and q are trace equivalent. On the other hand p and q are not completed trace equivalent because the set of completed traces of p is $\{aa\}$ while the set of completed traces of q is $\{a, aa\}$. □

Failure equivalence as introduced by Brookes, Hoare and Roscoe [29] further extends the notion of completed trace equivalence with the extra possibility of checking that after performing a trace sequence, a certain set of actions is disabled. Other equivalences following this idea were also introduced. E.g. testing equivalence has been investigated by Nicola and Hennessy [53] and it coincides with failure equivalence on finitely branching labelled transition systems, similarly as do the equivalences of Kennaway [103] and Darondeau [51] — this was showed in [52]. Failure equivalence is finer than completed trace equivalence as it is demonstrated by the following example.

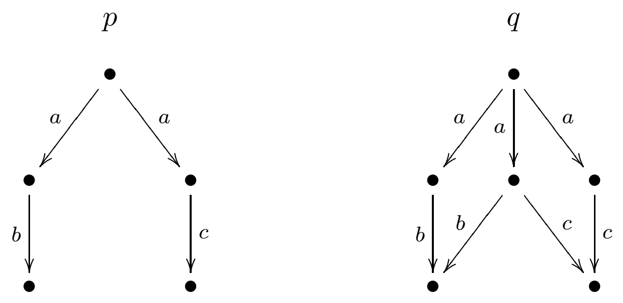
Example 1.3. Let us consider two processes p and q.



Processes p and q are completed trace equivalent since they are trace equivalent and have $\{ab, ac\}$ as the set of completed traces. They are, however, not failure equivalent. In the process p there is a trace a (taking the left branch) such that after this trace the action c is disabled. On the other hand, whenever the action a is performed in the process q, the action c is always enabled. □

Olderog and Hoare [143] suggested *readiness equivalence*: two processes are ready equivalent iff they have the same set of pairs consisting of a trace and a set of actions enabled after performing the trace. It is an easy observation that any two ready equivalent processes are also failure equivalent. Moreover (as demonstrated in the following example), readiness equivalence strengthens failure equivalence.

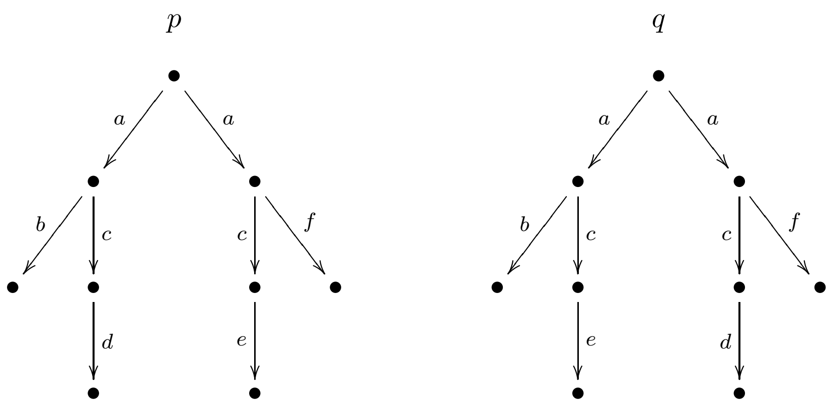
Example 1.4. Let us consider two processes p and q.



It is a routine exercise to list the sets of all possible traces from p and q together with the sets of actions that are disabled after performing the traces, and check that these two sets are equal. (The mentioned equal sets consist of (ϵ, S) for any $S \subseteq \{b,c\}$, $(a,\{a,b\})$, $(a,\{a,c\})$, $(a,\{a\})$, $(a,\{b\})$, $(a,\{c\})$, $(a,\emptyset)$, and (ab,S), (ac,S) for any $S \subseteq \{a,b,c\}$.) Hence p and q are failure equivalent. On the other hand, the process q contains a trace a (the middle branch), after which the set of actions $\{b,c\}$ is enabled. There is no such a trace in p, after which both b and c are enabled. This means that p and q are not ready equivalent. $\square$

The notion of *failure trace equivalence* (for finitely branching processes it is the same as *refusal equivalence* of Phillips [150]) is slightly more difficult to explain. The considered traces now include (except for the ordinary actions) also sets of actions such that a process p can perform a set of action X and become again the process p iff the set of actions enabled in p is disjoint with X. Two processes are failure trace equivalent iff they have the same set of extended traces.

Example 1.5. Let us consider the following two processes p and q.



Processes p and q are failure equivalent. On the other hand, they are not failure trace equivalent. The process p contains an extended trace $a\{f\}cd$ whereas q

does not: if the left branch is taken in q then no action d is possible in the future, and if the right branch is taken in q then after performing a the action f is not disabled. $\square$

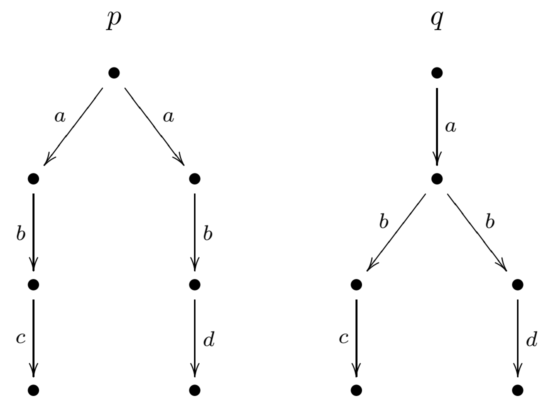
Remark 1.4. It is also easy to see that readiness equivalence and failure trace equivalence are incomparable. Processes p and q from Example 1.5 are ready equivalent but not failure trace equivalent. Similarly, p and q from Example 1.4 are failure trace equivalent but not ready equivalent.

Ready trace equivalence naturally completes the space above readiness and failure trace equivalences. It was independently introduced by Pnueli [152] (called *barbed equivalence*), Baeten, Bergstra and Klop [11] and Pomello [153] (called *exhibited behaviour equivalence*). Again, traces can contain a mixture of ordinary actions and sets of actions. This time a process p performs a set of actions X and becomes again the process p iff X contains exactly the actions enabled in p.

Remark 1.5. Justification of the position of ready trace equivalence above readiness equivalence and failure trace equivalence is provided in [194]. Moreover processes from Example 1.5 serve for demonstrating that ready trace equivalence is more discriminating than readiness equivalence, and similarly Example 1.4 shows the same for ready trace equivalence and failure trace equivalence.

Simulation equivalence is based on the classical notion of simulation (see e.g. Park [146]) and it is independent of all the equivalences described so far except for trace equivalence. We say that a process p simulates a process q iff whenever the process q can perform an action a and become a process q', the process p can perform the same action a and become a process p' such that p' simulates q'. Two processes p and q are simulation equivalent iff p simulates q, and q simulates p.

Example 1.6. Let us consider the following two processes p and q.



It is easy to see that p and q are ready trace equivalent. They are, however, not simulation equivalent since p cannot simulate q. The process q can perform the action a and become q'. Process p has to simulate this step by taking either the left or right branch in p and becoming p'. Assume that the left branch was taken (the other situation is symmetric). Now q' can perform the sequence bd

by taking the right branch and p' is obviously not able to simulate this sequence. Hence p and q are not simulation equivalent.

In order to complete the picture, note that processes p and q from Example 1.2 are simulation equivalent while they are not completed trace equivalent. □

Ready simulation equivalence (proposed by Bloom, Istrail and Meyer [23]) is a finer equivalence than both ready trace equivalence and simulation equivalence. We say that a process p ready simulates q iff the sets of actions enabled in p and q are the same and whenever the process q can perform an action a and become a process q', the process p can perform the same action a and become a process p' such that p' ready simulates q'. Two processes p and q are ready simulation equivalent iff p ready simulates q, and q ready simulates p.

Remark 1.6. To see that ready simulation is a finer equivalence than ready trace equivalence, consider Example 1.6: while p and q are ready trace equivalent, they are not ready simulation equivalent. Similarly, p and q from Example 1.2 are simulation equivalent but not ready simulation equivalent.

For the completeness of the picture we define also the following two equivalences but we do not discuss their positions in Figure 1.1. The interested reader can find more details in [194].

Possible-futures equivalence was introduced by Rounds and Brookes [158]. Two processes are possible-futures equivalent iff they have the same sets of pairs consisting of a trace sequence and the entire trace set after performing the sequence.

Finally, the last equivalence below bisimulation is 2-*nested simulation equivalence*, defined by Groote and Vaandrager [65]. 2-nested simulation is a simulation contained in a simulation equivalence, i.e., all the pairs in the definition of simulation must also be simulation equivalent. Symmetric closure of this preorder gives 2-nested simulation equivalence.

Last but not least, we will discuss *bisimulation equivalence.* It is perhaps *the* equivalence that has attracted most attention in concurrency theory, and it is also the main interest of this thesis. Two processes p and q are bisimulation equivalent (or simply bisimilar) iff whenever p can perform an action a and become a process p', the process q can perform the same action a and become a process q' such that p' and q' are bisimilar; and symmetrically whenever q can perform an action a and become a process q', the process p can perform the same action a and become a process p' such that p' and q' are bisimilar.

As originally introduced by Park [146] and Milner [133], bisimilarity appeared to play a prominent role due to many pleasant properties it possesses. We shall mention some of them in detail since decidability and complexity issues of bisimilarity checking is the main focus of this thesis.

Game Characterization

Bisimulation equivalence has an elegant characterisation in terms of *bisimulation games* [190, 183]. A bisimulation game on a pair of processes p and q is a two-player game between an ‘attacker’ and a ‘defender’. The game is played in *rounds*. In each round the players change the current processes p and q according to the following rule.

- The attacker chooses either p or q and performs an action a from the selected process.
- The defender has to perform the same action a from the other process.

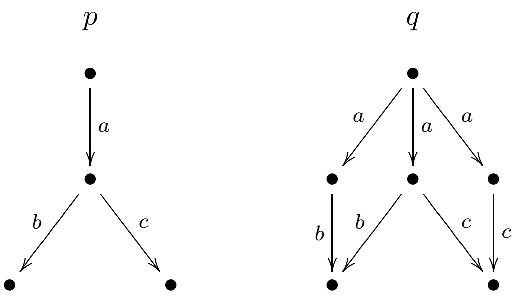
The players reached a new pair of processes p' and q' and the game continues with another round from the current processes p' and q'.

A *play* is a maximal sequence of pairs of states formed by the players according to the rule described above, and starting from the initial processes p and q. The defender is the winner in every infinite play. A finite play is lost by the player who is stuck. Note that the attacker gets stuck in current processes p' and q' if and only if no actions are enabled from these two processes.

The following standard fact highlights the connection between the winning strategies in bisimulation games and bisimulation equivalence: processes p and q are bisimilar iff the defender has a winning strategy (and nonbisimilar iff the attacker has a winning strategy).

The game-theoretic characterization of bisimilarity introduced above is simple, yet powerful. It provides an intuitive understanding of this notion and will be frequently applied throughout the rest of the thesis.

Example 1.7. In this example we shall demonstrate that simulation equivalence is not the same as bisimulation equivalence. Let us consider the following processes p and q.



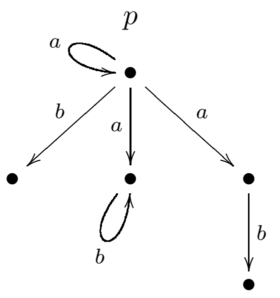
It is easy to see that p and q are simulation equivalent. We will define a winning strategy for the attacker in order to show that p and q are not bisimilar. In the first round the attacker chooses the action a in the process q by taking the left-most transition. The defender has only one possible answer from p thus reaching a process p'. In the second round the attacker switches the processes and plays from p' under the action c. The defender does not have the action c enabled in the other process and hence he loses. □

Logical Characterization

There exists a nice correspondence between bisimilarity and a well known modal logic of Hennessy and Milner [68]. This logic is essentially a propositional logic without propositional variables and with two constants 'true' and 'false'. For each action a it moreover contains the modal operator '$[a]$'. Given a process p

and formula ϕ, we can ask the question whether ϕ is valid in p ($p \models \phi$), which can be read also as "p satisfies ϕ". The interpretation of 'true', 'false', and of the logical connectives is as expected. The formula $[a]\phi$ is valid in p if and only if for all p' reachable from p under the action a, it is the case that ϕ is valid in p'.

Example 1.8. Let p be the following process.



Let ϕ be a formula saying that after performing the action a, the action b becomes enabled. This can be expressed as follows.

$$\phi \equiv [a]\neg[b]\text{false}$$

Obviously p satisfies ϕ. In the usual presentation of this logic, the modal operator '$\langle a\rangle$' (the meaning of $\langle a\rangle\psi$ is that there must be a transition labelled by a into a process satisfying ψ) is often used. The formula $\langle a\rangle\psi$ is an abbreviation for $\neg[a]\neg\psi$. Hence in our case, we can write the formula ϕ also as $[a]\langle b\rangle$true. □

An important observation of Hennessy and Milner [68] is that two (finitely branching) processes are bisimilar if and only if they satisfy the same formulae of the logic introduced above. This result was later extended to a stronger logic called modal μ-calculus (see [105] and [155]) which is based on Hennessy-Milner logic and adds the extra power of recursive definitions.

These results show that it is possible to link two different approaches to formal verification: equivalence checking and model checking.

Example 1.9. Let us consider the processes p and q from Example 1.7. Since p and q are not bisimilar, there should be a formula distinguishing them. Indeed, e.g. the formula

$$\phi \equiv \langle a\rangle[c]\text{false}$$

is valid in q but not in p. □

Remark 1.7. For completeness let us mention that it is possible to give logical characterizations also to other equivalences from the linear/branching time hierarchy [109]. These characterizations are achieved by defining a corresponding sublogic of Hennessy-Milner logic for a given behavioural equivalence. This provides a modal characterization of the equivalence in the previously introduced sense: two processes are equivalent if and only if they satisfy the same formulae of the sublogic.

Computational Feasibility
Another remarkable property of bisimilarity is its computational feasibility. Let us give a few examples of this. The interested reader is referred e.g. to the overview note of Moller and Smolka [138].

- The first example is from the well studied class of *finite-state processes*, i.e., rooted transition systems with finitely many states. The problem is to decide whether a pair of finite-state processes is equivalent w.r.t. some notion of behavioural equivalence. This problem for all equivalences (save bisimilarity) from the linear/branching time spectrum is PSPACE-complete (see [102]). On the other hand, bisimilarity has the rare status of being decidable in polynomial time [102, 144].

- Another example uses (infinite) labelled transition systems generated by *ϵ-free context-free grammars* applying left-most derivations only — this class is often called Basic Process Algebra (BPA). It is a well known fact from the formal language theory that completed trace (language) equivalence is undecidable for this class (see e.g. [80]). Moreover, Huynh and Tian proved [85] that readiness and failure equivalences are also undecidable. These undecidability results were later extended by Groote and Hüttel [64] to all equivalences in the linear/branching time hierarchy save bisimilarity. Surprisingly, (strong) bisimulation equivalence is decidable for BPA [46]. If the grammar has no redundant nonterminals, bisimilarity is decidable even in polynomial time [74], while the other equivalences remain undecidable [80, 85, 64].

- A similar story as in the previous point is true also for the *commutative* version of *context-free processes.* This class is usually referred to as Basic Parallel Processes (BPP). Again, (strong) bisimilarity is decidable for BPP [43]. In the restricted case without redundant nonterminals even in polynomial time [75]. However, none of the equivalences below bisimilarity are decidable [82].

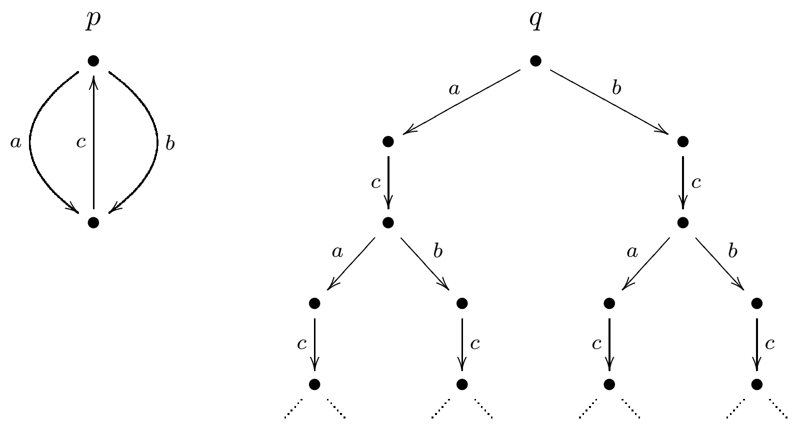
- The last example we provide deals with the class of transition systems generated by *pushdown automata.* From the language point of view there is no difference between pushdown automata and context-free grammars, since both formalisms describe the class of context-free languages. On the other hand the situation is different when considering bisimilarity as the equivalence relation: pushdown automata describe a larger class than context-free grammars (see [34]). Nevertheless, Stirling proved decidability of (strong) bisimilarity for normed (from every reachable state it is possible to empty the stack) pushdown automata [184] and the same question for the whole class was positively answered by Sénizergues [163].

Categorical Definition of Bisimilarity
The essence of bisimilarity, quoting Hennessy and Milner [68], "is that the behaviour of a program is determined by how it communicates with an observer."

Therefore, the notion of bisimilarity for different models is defined in terms of their behaviours and observable behaviours.

For example for rooted labelled transition systems it seems natural to identify their behaviours with (possibly infinite) synchronization trees [133] into which they unfold, and to take sequences of actions as observations.

Example 1.10. Let us consider the following two processes p and q (we show only a finite fraction of the process q, however, the infinite behaviour of q is regular according to the obvious pattern).



The process q is the unfolding of p, which means that q naturally represents the essence of behaviour of the process p. Moreover, q is a tree and the intuition is that whenever the process p makes a choice between the actions a and b, the futures of such computations are never the same, i.e., the computations in q never enter the same state. Also note that the unfolding of a finite process can contain infinitely many reachable states, as demonstrated by our example. □

The abstract definition of bisimilarity for arbitrary categories of models due to Joyal, Nielsen and Winskel [99] formalizes this idea. Given a category of models where objects are behaviours and morphisms indicate how one behaviour extends the other one, and given a subcategory of observable behaviours, the abstract definition yields a notion of bisimilarity for all behaviours with respect to observable behaviours.

For example, for rooted labelled transition systems, taking synchronization trees as their behaviours, and sequences of actions as the observable behaviours, we recover the standard notion of bisimilarity introduced above.

Another abstract definition of bisimilarity is that based on coalgebras. Transition systems of various kinds can be viewed as coalgebras for appropriate endofunctors. This approach gives rise to a definition of bisimulation as a span of coalgebras [191, 160].

One more example demonstrating how causality of actions can affect understanding of process equivalences uses labelled *asynchronous* transition systems [18, 166] which unfold into event structures as their behaviours. Assuming that instead of sequences (linear orders) of actions we can observe partial orders of action occurrences, the categorical approach from [99] yields the abstract definition of hereditary history preserving bisimilarity (hhp-bisimilarity), independently introduced and studied by Bednarczyk [19]. The next subsection will give more details about this partial order semantics.

1.2.2 Interleaving vs. Partial Order

In this subsection we will compare different approaches dealing with concurrent executions of actions (events). Let us assume that we want to model a process P which can perform concurrently (independently) actions a and b, and terminate afterwards.

In the *interleaving approach*, concurrency is modelled using non-determinism. This means that the process P can be modelled by the following process p (we also show the corresponding unfolding q of the process p).

p

$$\begin{tikzcd}
& \bullet \arrow[dl, "a"'] \arrow[dr, "b"] & \\
\bullet \arrow[dr, "b"'] & & \bullet \arrow[dl, "a"] \\
& \bullet &
\end{tikzcd}$$

q

$$\begin{tikzcd}
& & \bullet \arrow[dl, "a"'] \arrow[dr, "b"] & & \\
& \bullet \arrow[dl, "b"'] & & \bullet \arrow[dr, "a"] & \\
\bullet & & & & \bullet
\end{tikzcd}$$

Hence the process p can perform either the action a followed by the action b or vice versa. In the interleaving approach there is no apparent way to distinguish between a concurrent execution of the two actions and a nondeterministic choice between them (as shown in the process p). This is supported also by the fact that in the unfolding q of the process p, we do not reach the same state after performing the sequences ab and ba, i.e., q is a tree.

In the *partial order approach*, we specify the distinction between concurrency and nondeterminism explicitly. Mazurkiewicz's traces were suggested as a suitable semantics for this purpose [130, 131]. Modelling of concurrency is achieved by introducing an independence alphabet with an explicit independence relation.

Example 1.11. In this example we shall explain the main idea of independence alphabet. Assume that the set of actions is equal to $\{a, b, c\}$ such that the actions a and b are mutually independent, but dependent with the action c.

This can be formally defined by introducing an irreflexive and symmetric *independence relation* I over the set of actions. In our example we have

$$I \overset{\text{def}}{=} \{(a, b), (b, a)\}.$$

The intuition is that whenever during a process execution the actions a and b appear after one another, the observer of such a system should not be able to determine in which order they were performed. Hence e.g. the execution of a sequence *caabc* should appear the same as e.g. *cabac*.

This can be formalized by saying that two action sequences are one-step related whenever one sequence can be obtained from the other by swapping two neighbouring independent actions. By taking a reflexive and transitive closure of this one-step relation we get an equivalence relation which identifies exactly those action sequences that cannot be distinguished.

In our example *caabc* and *cabac* are one-step related because in *caabc* we have two neighbouring independent actions at positions 3 and 4, and by swapping them we get the sequence *cabac*. Since also *cabac* is one-step related to *cbaac*, we can conclude that *caabc* is (in two steps) equivalent to *cbaac*. In fact the equivalence class represented by *caabc* contains exactly the following sequences of actions: *caabc*, *cabac* and *cbaac*.

Another possible notation is to describe such an equivalence class as a labelled partially ordered set (trace). The labelled partial order corresponding to the sequence *caabc* is the following one.

$$\begin{tikzcd}
 & c \arrow[dl, dash] \arrow[ddr, dash] & \\
a \arrow[d, dash] & & \\
a \arrow[dr, dash] & & b \arrow[dl, dash] \\
 & c &
\end{tikzcd}$$

By taking all linearisations of this partial order we obtain exactly the equivalence class of the sequences represented by *caabc*, i.e., $\{caabc, cabac, cbaac\}$. □

Many models were suggested to fulfill these ideas, e.g. elementary net systems, event structures, trace languages and asynchronous transition systems. For surveys see [203, 142]. We will focus on the model of labelled asynchronous transition systems since it is a natural generalisation of labelled transition systems mentioned in the previous sections of this thesis.

Asynchronous transition systems were introduced by Bednarczyk [18] and by Shields [166]. The idea is that labelled transition systems are extended with an irreflexive and symmetric independence relation over the actions. Whenever two actions a and b are independent and can be performed in a sequence (i.e. from a state s after performing the sequence ab a state t is reached), the asynchronous transition system must satisfy that from s it is possible to perform also the sequence ba and reach the same state t.

When modelling the process P that can perform independently the actions a and b, we get the following asynchronous transition system (to indicate explicitly the independence between a and b we connect them by an arc).

$$\begin{tikzcd}
 & s \arrow[dl, "a"'] \arrow[dr, "b"] & \\
\bullet \arrow[dr, "b"'] & & \bullet \arrow[dl, "a"] \\
 & t &
\end{tikzcd}$$

In order to define a suitable notion of unfolding of the process s above, we still require that the unfolding is an acyclic labelled transition system but we do not insist that it has to be a tree. In particular, whenever two independent actions can be performed in a sequence, the requirement on the asynchronous transition system mentioned above is reflected also in the unfolding. Hence e.g. in our picture the unfolding of the asynchronous process s is isomorphic to the process s itself. On the other hand, nondeterministic choice between dependent actions does not obey the "diamond" property and unfolds in the usual way into a tree. This also means that any labelled transition system can be considered as an asynchronous transition system with the empty independence relation.

Remark 1.8. Let us mention an interesting observation which illustrates how interleaving and partial order approaches intuitively explain the different understanding of concurrency. First, we consider the process p from the beginning of this subsection and its behaviour (unfolding) q. As mentioned before, after the actions a and b were performed, different states in q are reachable depending on the order in which these actions were performed. It can be checked by going back into the history and verifying the last action that was executed (either b or a). This action is always unique. On the other hand, when considering the process s (defined above) which unfolds into itself, after the actions a and b were performed the current situation of the system is represented by the state t. However, looking back into the history we cannot say whether a or b was performed as the last action since the situation is symmetric for these two actions. This reflects the intuition that whenever two actions are truly concurrent, one cannot determine the order in which they were executed.

When defining a notion of behavioural equivalence for asynchronous transition systems, the independence relation should be taken into account. Such equivalences are usually defined on the corresponding unfoldings of given asynchronous processes. Considering e.g. bisimilarity, we can extend the rules of bisimulation games on unfoldings to allow backward moves of the attacker and the defender. Hence during a bisimulation game the complete history of the game is remembered and it is possible to backtrack into the history. In case where the independence relation is empty, the history is uniquely determined since the unfolding is a tree. If it is not the case, there might be different choices for going one step back into the history.

Assume again the process s from the previous picture. After performing the sequence ab the state t is reached. If the attacker now decides to make the backward move, he has two choices: either to the state reachable from s after performing the action a (the left branch), or to the state reachable from s after performing the action b (the right branch).

This gives rise to the notion of *hereditary history preserving bisimilarity* (hhp-bisimilarity) that was introduced by Bednarczyk [19] and discovered independently by Joyal, Nielsen and Winskel [99] using a categorical approach to bisimilarity.

Remark 1.9. Another equivalence called *history preserving bisimilarity* was introduced by many authors, among others by Rabinovich and Trakhtenbrot [156], and van Glabbeek and Goltz [196]. This equivalence is similar to hhp-bisimilarity but allows only a limited access into the history.

The negative news is that these versions of bisimilarity based on partial orders are usually intractable for automatic verification. For finite-state asynchronous processes, history preserving bisimilarity remains decidable (proved by Vogler [200] for a related model of 1-safe Petri nets) but it becomes an EXPTIME-complete problem (a result by Jategaonkar and Meyer [98]). Even more, hhp-bisimilarity was shown to be undecidable for labelled finite-state asynchronous processes and 1-safe Petri nets by Jurdzinski and Nielsen [100]. Recently we extended the result to demonstrate that the problem is undecidable also for unlabelled asynchronous transition systems [101].

In this thesis we shall adopt the interleaving paradigm: we will assume that an external observer of a system cannot detect causality between visible actions of the system (see e.g. [135]). This is a standard approach when studying complexity and decidability issues for infinite-state systems. The main reason for this is perhaps the computational hardness (or even undecidability) of partial order semantics already for finite-state processes.

1.2.3 Degree of Abstraction from Unobservable Actions

During the process of modelling a concurrent system one usually considers two kinds of actions that the system can perform: *visible (observable) actions*, and *internal (unobservable) actions*. Observable actions often symbolize communication with the environment, that is, they are supposed to be visible for an independent observer of such a system. On the other hand, unobservable actions serve for internal purposes of the process only (such as e.g. synchronization), and are commonly denoted by τ. The internal action τ is not expected to be detectable by an external observer.

Several semantic approaches can differ in the way they treat the unobservable action τ. When discussing the role of τ for behavioural equivalences, we will focus on the notion of bisimilarity because it is the equivalence studied throughout this thesis.

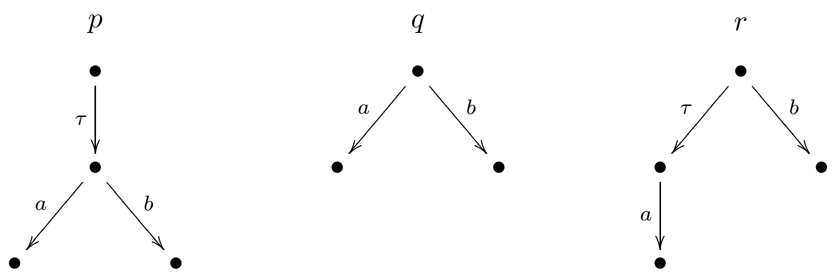
One extreme when dealing with internal actions is to regard the action τ as a visible one. In case of bisimilarity this gives rise to so called *strong bisimilarity* [146, 133].

Another possibility is to disregard the τ actions and agree that only the visible actions are observable. Citing Milner [135]: "... we merely require that each τ action is matched by *zero* or more τ actions ...". The notion of bisimilarity achieved this way is called *weak bisimilarity.*

In order to define weak bisimilarity one usually introduces so called *weak transition relation.* The idea is that a process p performs under the weak transition relation an action a and becomes a process q whenever it is possible to perform from p zero or more τ actions and then the action a, followed again by zero or more τ actions. We also allow that p under the weak transition relation performs the τ action and becomes p again.

Weak bisimilarity is then a bisimilarity (as defined in Subsection 1.2.1) where instead of the ordinary transitions we use weak transition relations.

Example 1.12. Let us consider three processes p, q and r.

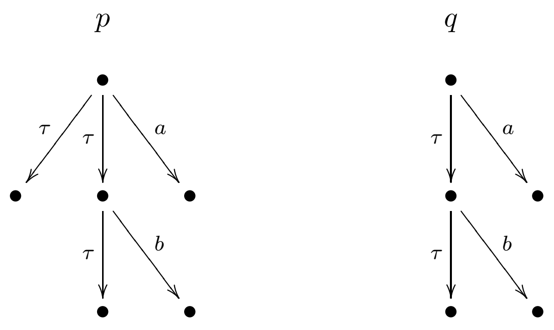


When τ is regarded as an ordinary (visible) action then p and q are not bisimilar (strongly bisimilar). On the other hand it can be easily seen that p and q are weakly bisimilar.

An interesting observation is that q and r are not even weakly bisimilar. The process r can use the action τ and the process q can only stay in the same state. Now it is possible to play along the action b in the process q but there is no such move in the other process. □

In [197] van Glabbeek and Weijland introduced a finer notion of behavioural equivalence than weak bisimilarity called *branching bisimilarity*. Their approach builds on the ideas of weak bisimilarity but it moreover distinguishes between processes that change their branching properties after the performance of individual τ actions. This in particular means that if a τ action is performed in one of the processes then the other process not only has to match this move by a sequence of τ actions but also all the intermediate states reached during this sequence have to be bisimilar to the first process.

Example 1.13. Let us consider the following processes p and q.



It can be easily seen that p and q are weakly bisimilar. The reason for this is that a move under the action τ which uses the left-most branch in the process p can be simulated by a sequence of two τ actions in the process q. However, since the state which is reachable after the first τ action performed in the process q is not bisimilar to p, the processes p and q are not branching bisimilar. □

For completeness let us mention that at least two other behavioural equivalences called *eta bisimilarity* [14] and *delay bisimilarity* [134] were introduced. They treat the abstraction from unobservable actions in a slightly different way than branching bisimilarity and are positioned between weak and branching bisimilarity, mutually incomparable.

Since in this thesis we study only strong and weak bisimilarity, we shall not provide further details about the other notions of bisimilarity. The interested reader is referred to [198].

1.2.4 Approaches to Infinite Executions

In this subsection we shall focus on different approaches to infinite computations. The phenomenon we have in mind is called *divergence.* A process is divergent iff it can perform an infinite sequence of unobservable τ actions.

Divergence has been intensively studied, among others e.g. by Hennessy and Plotkin [69], Milner [134], Abramsky [3], Stirling [182], Walker [202], Aceto and Hennessy [4], van Glabbeek [195], and Lohrey, Argenio and Hermanns [117].

Let us consider the following processes p and q.

$$\begin{tikzcd} p & & q \\ \bullet \arrow[loop left, "a"] & & \bullet \arrow[loop left, "a"] \arrow[loop right, "\tau"] \end{tikzcd}$$

While p and q are weakly bisimilar, it is the case that q is divergent but p is not. Often the fact that weak bisimulation equivalence is not sensitive to the existence of infinite internal computations appears to be useful, however, sometimes it may be appropriate to take this aspect into account.

Let us mention only some of the studied behavioural equivalences dealing (to different extend) with divergence. Again, we concentrate in particular on equivalences based on the notion of bisimilarity.

Two processes are *stable weakly bisimilar* iff they are weakly bisimilar and moreover any pair of processes reached during the bisimulation game satisfies: if in one process the action τ is disabled then the other process can perform a finite sequence of τ actions such that τ becomes also disabled.

Example 1.14. Let us consider the processes p and q above. It is easy to see that they are not stable weakly bisimilar because in p the τ action is disabled but τ is enabled in any state reachable from q (the set of all states reachable from q is equal to $\{q\}$). □

Similarly two processes are *completed weakly bisimilar* iff they are weakly bisimilar and the extra condition that any pair in the bisimulation game has to satisfy is the following one: if in one process all actions are disabled then the other process can perform a finite sequence of τ actions such that all actions (including τ) become also disabled.

Another behavioural equivalence requiring that the two considered processes are weakly bisimilar and moreover either both of them or none of them diverge is called *divergent stable weak bisimilarity.*

Example 1.15. Again, the processes p and q above are not divergent stable weakly bisimilar while being weakly and even completed weakly bisimilar. □

$$\begin{tikzcd}
\begin{array}{c}\textit{divergent stable}\\ \text{weak bisimilarity}\end{array} \arrow[dd] \arrow[dr] & \\
& \begin{array}{c}\textit{stable}\\ \text{weak bisimilarity}\end{array} \arrow[d] \\
\begin{array}{c}\textit{divergent}\\ \text{weak bisimilarity}\end{array} \arrow[dr] & \begin{array}{c}\textit{completed}\\ \text{weak bisimilarity}\end{array} \arrow[d] \\
& \text{weak bisimilarity}
\end{tikzcd}$$

Figure 1.2: Hierarchy of divergence equivalences

The last notion we shall mention here is known as *divergent weak bisimilarity*. It is similar to divergent stable weak bisimilarity except that instead of the condition that the processes simultaneously diverge, we require that they simultaneously satisfy the following condition: a process that diverges or is stuck (no actions enabled) is reachable.

Formal definitions of these equivalences can be found e.g. in [117]. We restrict ourselves to concluding this subsection with presenting a hierarchy of the introduced divergence equivalences. The hierarchy is shown in Figure 1.2 and examples demonstrating strictness of this hierarchy together with further comments are available in [195].

1.3 Processes with Infinitely Many States

The main goal of this thesis is to study *infinite-state processes*, i.e., rooted labelled transition systems with infinitely many reachable states. Since we aim at answering algorithmic and complexity questions, we have to agree first on a model which provides finite descriptions of such infinite-state systems.

There have been many different approaches to this issue and we will explain some of them in this section. We start by briefly mentioning graph grammars and then focus in more detail on formalisms based on process algebras, in particular on process rewrite systems.

Let us introduce the main question

"how to define a process with infinitely many reachable states in a finite way"

by using the following example. Assume a process p which has an infinite structure as depicted in Figure 1.3. Our task in the following subsections will be to show different formalisms which enable to define processes like p.

$$\begin{tikzcd}
\overset{p}{\bullet} \arrow[r, "a"] \arrow[d, "b"] & \bullet \arrow[r, "a"] \arrow[d, "b"] & \bullet \arrow[r, "a"] \arrow[d, "b"] & \cdots \\
\bullet \arrow[r, "a"] & \bullet \arrow[r, "a"] & \bullet \arrow[r, "a"] & \cdots
\end{tikzcd}$$

Figure 1.3: Process p with infinitely many reachable states

1.3.1 Graph Grammars

One of the approaches uses the fact that the process p from Figure 1.3 has a regular structure. We can observe that the following building block repeats with a regular pattern in the process p.

$$\begin{tikzcd}
\bullet \arrow[r, "a"] \arrow[d, "b"] & \bullet \arrow[d, "b"] \\
\bullet \arrow[r, "a"] & \bullet
\end{tikzcd}$$

The idea of *graph grammars* (see [66, 40, 56]) is based on such regular repetitions in the graph structure. The definition of a graph grammar uses *hypergraphs*, i.e., labelled oriented graphs where every edge is either an ordinary edge or hyperedge. For each hyperedge there is a hypergraph into which the edge rewrites. Pairs of hyperedges and the corresponding hypergraphs are called *rules* and we assume that all hypergraphs in the rules have finitely many nodes. A graph grammar is a finite collection of rules plus an initial (finite) hypergraph.

An infinite-state labelled transition system can be generated by a graph grammar as follows. We start from the initial hypergraph and replace all its hyperedges with corresponding hypergraphs according to the rules. This is called a *rewriting step*. A new hypergraph is obtained and the rewriting process is repeated from this hypergraph until a fixed-point is reached. The fixed-point can be reached either after finitely many rewriting steps (the resulting hypergraph contains no hyperedges and is finite) or after infinitely many rewriting steps (in this case the graph grammar generates an infinite-state system).

Example 1.16. We give a graph grammar which generates the labelled transition system from Figure 1.3. Consider the following rule

$$\begin{tikzcd}
\bullet \arrow[d, dotted, "A"] & & \bullet \arrow[r, "a"] \arrow[d, "b"] & \bullet \arrow[d, dotted, "A"] \\
\bullet & \longrightarrow & \bullet \arrow[r, "a"] & \bullet
\end{tikzcd}$$

where the dotted lines represent hyperedges. The other edges in the picture are ordinary ones. This rule says that the hyperedge A is replaced by the hypergraph on the right-hand side after the arrow. This is the only rule of the graph grammar and the initial hypergraphs consists of exactly the hyperedge A.

Starting from the hyperedge A, after two rewriting steps the hypergraph looks as follows.

$$\begin{tikzcd}
\bullet \arrow[r, "a"] \arrow[d, "b"] & \bullet \arrow[r, "a"] \arrow[d, "b"] & \bullet \arrow[d, dotted, "A"] \\
\bullet \arrow[r, "a"] & \bullet \arrow[r, "a"] & \bullet
\end{tikzcd}$$

Finally, after reaching the fixed-point we get exactly the graph from Figure 1.3.
□

1.3.2 Process Algebras

The idea of process algebras is based on defining a set of *basic (atomic) processes* modelling very simple behaviours together with *composition operators* which enable to define complex behaviours from the atomic processes by composing them in different ways. This main idea appears in many variations, let us mention e.g. the classical process algebras CCS [135], ACP [15] and CSP [79].

In this thesis we shall develop an elegant approach by Mayr [126] called *process rewrite systems (PRS)*. The basis of this model can be found already in [137], however, Mayr simplified and slightly extended the existing approaches. The model of process rewrite systems uses only two composition operators — sequential and parallel — and it provides a uniform framework, general enough to describe all the classes of infinite-state processes studied in this thesis.

There are several advantages of this model, among others:

- it has an interesting connection with formal language theory
- it is simple and easy to explain even on an informal level
- it establishes a uniform treatment of the composition operators
- it gives an interesting hierarchy of processes, containing e.g. basic process algebra, basic parallel processes, pushdown processes and Petri nets.

We start by pointing out the connection between formal language theory and process rewrite systems. Citing [59]: "The 'grammars as processes' research program proposes to look at grammars from a different point of view; not as language generators, but as generators of *behaviours* (formally captured as labelled transition systems)."

Let us now demonstrate these ideas. We begin with basic process behaviours. Take a look at production rules of left-linear grammars like

$$X \longrightarrow aY.$$

When interpreting X and Y as processes, the production rule above can be read as:

"process X performs the action a and becomes process Y".

It corresponds to the following labelled transition system

$$\begin{tikzcd}
\overset{X}{\bullet} \arrow[r, "a"] & \overset{Y}{\bullet}
\end{tikzcd}$$

and in the syntax of process rewrite systems it is written as

$$X \xrightarrow{a} Y.$$

Similarly the other possible production rules in left-linear grammars have the form

$$X \longrightarrow a$$

which has the following meaning in the process theory:

"process X performs the action a and terminates".

It corresponds to the labelled transition system

$$\begin{tikzcd}\overset{X}{\bullet} \arrow[rr, "a"] & & \epsilon\end{tikzcd}$$

where 'ϵ' is the symbol for the *empty process* that cannot perform any transition. In the syntax of process rewrite systems this rule is written as

$$X \xrightarrow{a} \epsilon.$$

We can now proceed by taking a look at the first operator for composing atomic process behaviours: *sequential composition.* The introduction of this operator is inspired by context-free production rules like

$$X \longrightarrow aYZ.$$

The interpretation of this rule is

"process X performs the action a and becomes a sequential composition of processes Y and Z".

In process rewrite systems this rule is written as

$$X \xrightarrow{a} Y.Z$$

with the intuition that the computation of $Y.Z$ starts by executing the process Y first, and only when Y terminates, the computation continues by executing the process Z. This corresponds to a *procedure call.* The process X calls a procedure Y and then continues as a process Z.

In order to define such composed behaviours, we do not use the usual inference rule for context-free grammars

$$\frac{X \longrightarrow w}{uXv \longrightarrow uwv}$$

but rather the *prefix-rewriting rule* (already formulated in the PRS-syntax)

$$\frac{X \xrightarrow{a} w}{X.v \xrightarrow{a} w.v}$$

which reflects the intuition that the execution of the second component in sequential composition cannot start before the process in the first component terminates (by convention we consider the process $\epsilon.X$ equivalent to X).

Remark 1.10. Even though context-free grammars with the usual inference rule and with the prefix-rewriting rule are equivalent (they generate the same languages), this is not the case for process rewrite systems because their behaviours are usually compared on different principles.

Example 1.17. Let us consider a process rewrite system with two simple rewrite rules:

$$X \xrightarrow{a} X.X \quad \text{and} \quad X \xrightarrow{b} \epsilon.$$

The first rule says that the process X can perform the action a and become a sequential composition of two copies of X. The process X can also perform the action b and terminate. Hence using the prefix-rewriting rule our system describes the following process X (i.e. labelled transition system rooted with X).

$$\begin{tikzcd}
X \arrow[r, bend left, "a"] \arrow[d, "b"'] & X.X \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] & X.X.X \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] & X.X.X.X \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] & \cdots \arrow[l, bend left, "b"] \\
\epsilon
\end{tikzcd}$$

Let us demonstrate e.g. the existence of a transition from the process $X.X.X$ to $X.X$ labelled by b. Because of the second rewrite rule we know that $X \xrightarrow{b} \epsilon$. Hence using the prefix-rewriting rule we get that also $X.X.X \xrightarrow{b} \epsilon.X.X$. As noted above, the process $\epsilon.X.X$ is equivalent to $X.X$. This proves that $X.X.X \xrightarrow{b} X.X$.

This example also demonstrates that a finite description of the process X (provided by two rewrite rules only) gives rise to a labelled transition system with infinitely many reachable states. □

In our analogy between grammars and process rewrite systems we can go even further by considering non-context-free production rules like

$$XY \longrightarrow aZ.$$

In the process terminology this rule represents *value passing* between processes and it is written as

$$X.Y \xrightarrow{a} Z.$$

The intuition is that X represents a value returned by some previous computation and the behaviour of the process Y is affected by this value. The following example shall demonstrates this.

Example 1.18. We describe a very simple implementation of a while-loop (without evaluating the body of the while command). Assume that a process X represents this while-loop in the sense that while performing the visible action a, the process nondeterministically decides whether the boolean condition is evaluated to true or false. This can be formally written as

$$X \xrightarrow{a} T.X \quad \text{and} \quad X \xrightarrow{a} F.X.$$

Now the process X can read the value of the computed boolean condition and react accordingly: if the value is true then it performs a visible action $\mathtt{tt}$ and becomes X again, otherwise it performs the action $\mathtt{ff}$ and terminates. This is described by two rewrite rules:

$$T.X \xrightarrow{\mathtt{tt}} X \quad \text{and} \quad F.X \xrightarrow{\mathtt{ff}} \epsilon.$$

A transition system representing the process X follows.

$$\begin{tikzcd} X \arrow[r, bend left, "a"] \arrow[d, "a"'] & T.X \arrow[l, bend left, "\mathtt{tt}"] \\ F.X \arrow[d, "\mathtt{ff}"'] & \\ \epsilon & \end{tikzcd}$$

$\square$

We now proceed by introducing the operator of *parallel composition* written as '$\|$'. Take a look again at context-free production rules like

$$X \longrightarrow aYZ.$$

This time we will, however, interpret these rules as introduction of parallel composition. In our terminology these rules are written as

$$X \xrightarrow{a} Y\|Z$$

with the intuition that

"process X performs the action a and becomes a parallel composition of processes Y and Z".

In other words the process X *forks* into Y and Z. Since the processes Y and Z should be run in parallel without any specific order of execution, we assume that the parallel operator '$\|$' is *commutative*, i.e., we consider $Y\|Z$ and $Z\|Y$ as the same process.

Similarly as for prefix-rewriting, we will have the following rule for parallel rewriting (keep in mind that '$\|$' is commutative).

$$\frac{X \xrightarrow{a} w}{X\|v \xrightarrow{a} w\|v}$$

If only context-free rules are considered, new process can be created but they cannot interact. *Interaction* is achieved by considering non-context-free production rules like

$$XY \longrightarrow aZ$$

which are in our terminology written as

$$X\|Y \xrightarrow{a} Z.$$

These kinds of rules are understood as follows:

"processes X and Y synchronize by jointly executing the action a and becoming the process Z."

Example 1.19. Assume that we have two processes X and Y such that under the action a they evaluate (in parallel) either to true or false, i.e.,

$$X \xrightarrow{a} T, \quad X \xrightarrow{a} F, \quad Y \xrightarrow{a} T, \quad \text{and} \quad Y \xrightarrow{a} F.$$

Consider now the process $X\|Y$ which should evaluate to a conjunction of the truth values computed by X and Y. This can be written as

$$T\|T \xrightarrow{\mathtt{tt}} T, \quad T\|F \xrightarrow{\mathtt{ff}} F, \quad \text{and} \quad F\|F \xrightarrow{\mathtt{ff}} F.$$

Recall that because of the commutativity of '$\|$', we do not have to add the rule $F\|T \xrightarrow{\mathtt{ff}} F$. The labelled transition system of the process $X\|Y$ looks as follows.

$X\|Y$

$$\begin{tikzcd}
 & & & \bullet \arrow[dlll, "a"'] \arrow[dl, "a"'] \arrow[dr, "a"] \arrow[drrr, "a"] & & & \\
T\|Y \arrow[dr, "a"] \arrow[drrr, "a"] & & F\|Y \arrow[dr, "a"] \arrow[drrr, "a"] & & X\|T \arrow[dlll, "a"'] \arrow[dl, "a"'] & & X\|F \arrow[dlll, "a"'] \arrow[dl, "a"] \\
 & T\|T \arrow[dr, "\mathtt{tt}"'] & & T\|F \arrow[dr, "\mathtt{ff}"'] & & F\|F \arrow[dl, "\mathtt{ff}"] & \\
 & & T & & F & &
\end{tikzcd}$$

$\square$

Finally, in the most general cases, we allow a mixture of process expressions containing the sequential and parallel operators on both sides of the rewrite rules. It allows rules like

$$X.Y \xrightarrow{a} (U\|V).Z.$$

This particular rule can be interpeted as follows: the process Y receives a return value X and performs the action a; after this a parallel execution of U and V is initiated and when both of the parallel components terminate, the computation continues with executing the process Z.

To finish the introduction of process rewrite systems we give rewrite rules that generate the infinite-state process p from Figure 1.3 (in fact they generate a labelled transition system isomorphic p). There are e.g. the following two possibilities.

- The rules use the sequential operator only and the root of the system is the process X.

$$X \xrightarrow{a} X.X \qquad X \xrightarrow{b} Y \qquad Y \xrightarrow{a} Y.Y$$

- The rules use the parallel operator only and the root of the system is the process $X\|Y$.

$$X \xrightarrow{a} X\|X \qquad Y \xrightarrow{b} \epsilon$$

Hierarchy of Process Rewrite Systems

As grammars are classified in the Chomsky hierarchy, a similar hierarchy can be introduced by restricting the left-hand and right-hand sides of rewrite rules. In its complete form (as it will be formally defined and discussed in Chapter 2), the hierarchy of process rewrite systems was introduced by Mayr [126]. Figure 1.4 shows this hierarchy where every class from the hierarchy is represented by the most general available form of rules.

One of the interesting points about the hierarchy is that most of the classes in Figure 1.4 have a natural machine characterization (context-free grammars, pushdown automata, Petri nets). These connections will be in detail discussed in Chapter 2. Let us now only remark that e.g. rules of the type $X \xrightarrow{a} Y.Z$ define the class of *context-free processes* (also called BPA for *Basic Process Algebra*), rules of the type $X \xrightarrow{a} Y\|Z$ correspond to *BPP processes* (BPP for *Basic Parallel Processes*), rules like $X.Y \xrightarrow{a} Z.U$ characterize the class of *pushdown processes* (PDA), rules of the type $X\|Y \xrightarrow{a} Z\|U$ are *Petri net*'s rules, and $X \xrightarrow{a} (Y\|Z).U$ are characteristic rules of *PA-processes* (PA for *Process Algebra*).

Based on the previous work summarized in [137], Mayr showed that the hierarchy of process rewrite systems is strict with regard to (strong) bisimulation equivalence. This means that e.g. when adding the possibility of value passing (rules like $X.Y \xrightarrow{a} Z.U$) to the rules of the form $X \xrightarrow{a} Y.Z$, more transition systems can be described in the sense that the richer class contains processes that are not bisimilar to any process from the class lower in the hierarchy. Nevertheless, e.g. the classes mentioned above (with and without value passing) are still language equivalent.

The study of algorithmic problems within this hierarchy is the main issue of this thesis. In order to further motivate the reader, we shall describe a neat relationship between process rewrite systems and control-flow analysis in the following section. Before doing that, let us briefly mention similar hierarchies and models studied by the researchers.

$$\begin{tikzcd}
 & & W.(X\|Y) \xrightarrow{a} (Z\|U).V \arrow[dl, dash] \arrow[dr, dash] & & \\
 & X.Y \xrightarrow{a} (Z\|U).V \arrow[dl, dash] \arrow[dr, dash] & & X\|Y \xrightarrow{a} (Z\|U).V \arrow[dl, dash] \arrow[dr, dash] & \\
X.Y \xrightarrow{a} Z.U \arrow[dr, dash] & & X \xrightarrow{a} (Y\|Z).U \arrow[dl, dash] \arrow[dr, dash] & & X\|Y \xrightarrow{a} Z\|U \arrow[dl, dash] \\
 & X \xrightarrow{a} Y.Z \arrow[dr, dash] & & X \xrightarrow{a} Y\|Z \arrow[dl, dash] & \\
 & & X \xrightarrow{a} Y & &
\end{tikzcd}$$

Figure 1.4: PRS-hierarchy — characteristic rewrite rules

Other Hierarchies

There have been several approaches to clasify infinite-state processes in a uniform way. Caucal provided a hierarchy of sequential processes [40] which defines classes of infinite-state systems by giving restrictions on rewrite rules in *prefix rewriting.*

Caucal's hierarchy starts with *type* 3 systems which correspond to the class of finite-state processes. Every rule in these systems is restricted in the following way: left-hand sides of the rewrite rules are atomic processes (no composition operators are involved) and right-hand sides are atomic processes or the empty process ϵ.

Type 2 systems (also called *basic process algebra*) capture context-free grammars in Greibach normal form: every rule is of the form $X \xrightarrow{a} \beta$ where X is an atomic process and β is an arbitrary sequential composition of processes.

Systems of *type* $1\frac{1}{2}$ and *type* 0 represent the same class of processes up to isomorphism [40] and are known as *pushdown graphs.* Every rule in type $1\frac{1}{2}$ takes the classical pushdown form $pX \xrightarrow{a} q\beta$ where p and q are control states, X is a stack symbol and β is a sequential composition of stack symbols. Type 0 rules are of the form $\alpha \xrightarrow{a} \beta$ where α and β are arbitrary sequential compositions.

So far, the defined classes used only a finite number of rewrite rules obeying the described forms. Caucal's hierarchy defines two additional classes which allow for infinite number of rules. *Type* -1 systems consist of a finite number of rewrite meta-rules of the form $R \xrightarrow{a} \beta$ where R is a regular language over sequential compositions of processes and β is an arbitrary sequential composition. Each rule $R \xrightarrow{a} \beta$ is interpreted as a potentially infinite family of rules $\alpha \xrightarrow{a} \beta$ such that $\alpha \in R$.

Similarly, *type* -2 systems (also known as *prefix-recognizable graphs*) have a finite number of meta-rules of the form $R_1 \xrightarrow{a} R_2$ where R_1 and R_2 are regular languages over sequential compositions of processes and each such rule defines a family of rules $\alpha \xrightarrow{a} \beta$ such that $\alpha \in R_1$ and $\beta \in R_2$.

Another hierarchy of infinite-state processes was presented by Moller in [137]. Except for sequential rewriting he introduces also parallel rewriting and provides a classification of systems with the parallel operator. This includes classes like *basic parallel processes*, *Petri nets* and *PA-processes.*

Selection of Other Models

There is a number of other infinite-state models which were not mentioned so far. For completeness, let us mention some of them here.

We start with the classical process algebras CCS (*Calculus of Communicating Systems*) [135], ACP (*Algebra of Communicating Processes*) [15] and CSP (*Communicating Sequential Processes*) [79]. These algebras are based on similar principles as process rewrite systems: a few primitive process behaviours are in a structural way composed into more complex ones via a number of composition operators. The difference is that a wider variety of these operators is introduced (including e.g. restriction, relabelling and forced communication). On the other hand, these models are in general Turing powerful and hence most of the interesting properties become undecidable.

Even though the classes from the PRS-hierarchy cover many of the infinite-state processes, a few other subclasses are of particular interest. Processes generated by *one counter automata and nets* (a subclass of pushdown processes) were studied, providing a substantial simplification of verification algorithms (also with regard to complexity issues) [91, 95, 108] compared to the more general class of pushdown processes. Another often studied class is the parallel analogue to pushdown processes called *multiset automata* (and denoted as PPDA). This is a proper subclass of Petri nets where strong bisimilarity is still undecidable [34, 97]. More exotic formalisms include e.g. *queue processes* [33].

From other models let us mention mainly *communication protocols* [28, 24], *vector addition systems* (essentially equivalent to Petri nets) and in particular *lossy vector addition systems* [27, 127], *lossy channel systems* [1], *timed process algebras* [204], *real-time and hybrid automata* [5, 7, 8, 114, 6], and systems for *concurrent constraint programming* [161].

1.4 Control-Flow Analysis

This section demonstrates how the simple principles of process rewrite systems find a natural application in interprocedural [104] control-flow analysis of programs.

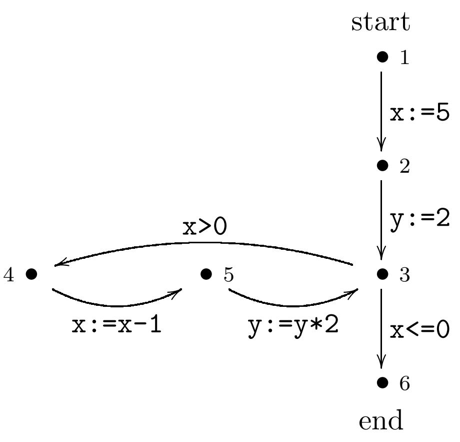
In general, *control-flow analysis* aims at deciding run-time properties of programs at the compilation phase, i.e., without actually executing the programs. Since all the interesting properties are in general undecidable (the considered model is Turing-powerful), we can never provide complete answers to the questions of interest. The approach of control-flow analysis overcomes this disadvantage by abstracting from the specific data and by studying the flow of control only. Hence instead of strict *"yes/no"* answer to a given question we may also get the *"yes/maybe/no"* style of answer. The soundness of the answer is still preserved, however, it is not always guaranteed that a full answer is provided. The theory of abstract interpretation [50, 119] is a formal foundation for considerations like these.

Typical questions in control-flow analysis are focused on program optimisations like elimination of partially redundant expressions and partially redundant assignments, partially dead code elimination or strength reduction [139].

In this section we will show that process rewrite systems provide a neat semantics to sophisticated control mechanisms like recursive procedures, multithreading and synchronization.

Let us start with an example demonstrating how a control-flow graph is obtained from a simple while-loop.

```
x:=5;
y:=2;
while x>0 do
  x:=x-1;
  y:=y*2;
endwhile
```



In the picture above the given program is assigned a control-flow graph such that the assignments are considered as atomic operations and the nodes in the graph (marked 1 to 6) indicate the interesting control points of the program. An execution of the control-flow graph is any directed path from the start point to the end point. The abstraction from the concrete data we made in particular means that executions of the control-flow graph contain except for *real* executions of the program also some *superfluous* executions (e.g. the path containing the actions `x:=5`, `y:=2`, and `x<=0`). This is caused by the fact that any branching of the program according to a boolean condition (like in the "while" and "if" commands) is modelled by a nondeterministic choice.

Nevertheless, the important observation is that any property holding on all executions of the control-flow graph will in particular hold on the real executions.

Let us now show how to model such a control-flow graph by means of process rewrite systems. The idea is that the atomic commands are considered as actions and each control point of the graph is assigned a process name (in our example let us say $X_1, \ldots, X_6$). Then any edge in the control-flow graph of the form $n \xrightarrow{v:=e} m$ where n and m and natural numbers gives rise to the rewrite rule $X_n \xrightarrow{v:=e} X_m$. Moreover, whenever n is the end point, we also have the rule $X_n \xrightarrow{end} \epsilon$. In our particular case we get the following rewrite rules.

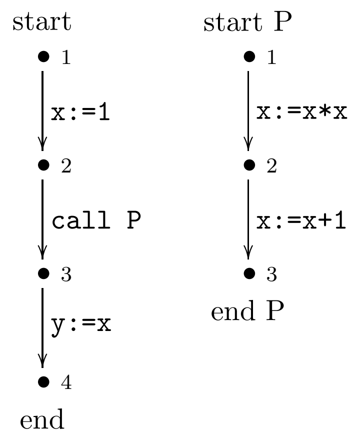
$$X_1 \xrightarrow{\texttt{x:=5}} X_2 \qquad X_2 \xrightarrow{\texttt{y:=2}} X_3 \qquad X_3 \xrightarrow{\texttt{x>0}} X_4 \qquad X_4 \xrightarrow{\texttt{x:=x-1}} X_5$$

$$X_5 \xrightarrow{\texttt{y:=y*2}} X_3 \qquad X_3 \xrightarrow{\texttt{x<=0}} X_6 \qquad X_6 \xrightarrow{end} \epsilon$$

Let us now consider an example of a simple *procedure call*, where the main program calls a procedure `P` and when the execution of the procedure terminates, the control-flow returns to the main program.

```
x:=1;
call P;
y:=x;


procedure P
  x:=x*x;
  x:=x+1;
end
```



Such procedure calls can modelled by sequential compositions. Hence every edge in the control-flow graphs of the form $n \xrightarrow{\texttt{call P}} m$ has the corresponding rewrite rule $X_n \xrightarrow{\texttt{call P}} P_1.X_m$ assuming that the process P_1 represents the starting point in the control-flow graph of the procedure P. This means that the process P_1 has to terminate before the execution of the main program continues from the control point m.

The complete set of rewrite rules looks as follows.

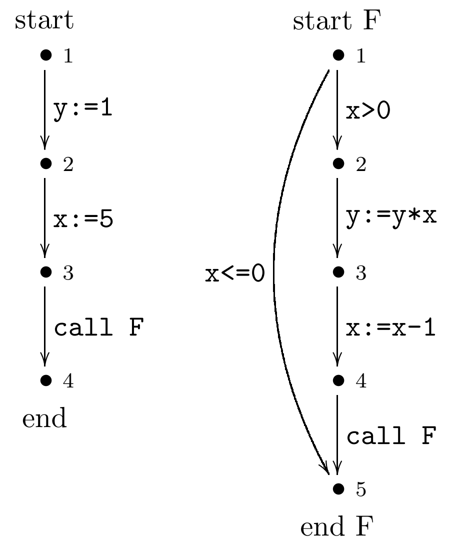
$$X_1 \xrightarrow{\texttt{x:=1}} X_2 \qquad X_2 \xrightarrow{\texttt{call P}} P_1.X_3 \qquad X_3 \xrightarrow{\texttt{y:=z}} X_4 \qquad X_4 \xrightarrow{end} \epsilon$$

$$P_1 \xrightarrow{\texttt{x:=x*x}} P_2 \qquad P_2 \xrightarrow{\texttt{x:=x+1}} P_3 \qquad P_3 \xrightarrow{end} \epsilon$$

So far we employed only processes with finitely many reachable states (even though the sequential operator was used). The situation changes when considering *recursive procedures.* This is demonstrated by the following example of a program computing the factorial function.

```
y:=1;
x:=5;
call F;


procedure F
  if x>0 then
    y:=y*x;
    x:=x-1;
    call F;
  endif
end
```



The rewrite rules for the factorial program follow.

$$X_1 \xrightarrow{\texttt{y:=1}} X_2 \qquad X_2 \xrightarrow{\texttt{x:=5}} X_3 \qquad X_3 \xrightarrow{\texttt{call F}} F_1.X_4 \qquad X_4 \xrightarrow{end} \epsilon$$

$$F_1 \xrightarrow{\texttt{x<=0}} F_5 \qquad F_1 \xrightarrow{\texttt{x>0}} F_2 \qquad F_2 \xrightarrow{\texttt{y:=y*x}} F_3 \qquad F_3 \xrightarrow{\texttt{x:=x-1}} F_4$$

$$F_4 \xrightarrow{\texttt{call F}} F_1.F_5 \qquad F_5 \xrightarrow{end} \epsilon$$

Note that computations like the following one (actions are omitted)

$$X_1 \longrightarrow X_2 \longrightarrow X_3 \longrightarrow F_1.X_4 \longrightarrow \cdots \longrightarrow F_4.X_4 \longrightarrow F_1.F_5.X_4 \longrightarrow \cdots$$

$$\longrightarrow F_4.F_5.X_4 \longrightarrow F_1.F_5.F_5.X_4 \longrightarrow \cdots \longrightarrow F_1.F_5.F_5.F_5.X_4 \longrightarrow \cdots$$

prove that there are infinitely many reachable states in the process X_1.

Let us continue with a brief sketch of how parallel procedure calls, spawning of processes and synchronization can be modelled.

An edge in the control-flow graph of the form $n \xrightarrow{\texttt{call P}\|\texttt{Q}} m$ represents the fact that procedures P and Q are called in *parallel.* This is reflected by the rewrite rule $X_n \xrightarrow{\texttt{call P}\|\texttt{Q}} (P_1\|Q_1).X_m$ where P_1 and Q_1 are the starting points of procedures P and Q, respectively.

Similarly, the action that a process *spawns* another one is described in the control-flow graph by the edge $n \xrightarrow{\texttt{spawn P}} m$. The corresponding rewrite rule is $X_n \xrightarrow{\texttt{spawn P}} P_1\|X_m$ where P_1 is the starting point of the procedure P.

Finally, *synchronization* can be modelled by rules of the form $X\|Y \xrightarrow{a} Z\|U$. The interested reader is referred to [59] for further details (including discussions about local and global communication channels).

To sum up, flat while-programs can be modelled by finite-state processes. When adding procedures, sequential composition pops up on the right-hand side of some rules. Parallel threads move us to the type of rules that contain both sequential and parallel operators on the right-hand side. Synchronization between threads requires the use of parallel composition on the left-hand side of the rules, and when procedures are allowed to return values we need the sequential composition on the left-hand sides as well.

1.5 Studied Problems

In the previous sections we introduced the notions of infinite-state processes and behavioural equivalences. This section outlines the algorithmic questions we may ask about the systems. We shall in particular focus on the *equivalence checking* problems.

1.5.1 Deciding Equivalences

The intuition is that the *implementation* of a concurrent process is modelled as a potentially infinite-state labelled transition system as well as the *specification* (intended behaviour) of the process. The correctness of the implementation is determined by comparing it to the given specification, hopefully in an automatic way. The comparison is done by testing whether the transition systems of implementation and specification are equivalent with regard to a suitable notion of behavioural equivalence. Most often we shall use strong and weak bisimulation equivalence for this purpose.

The decision problem is then formulated as follows.

"Given finite descriptions of two rooted labelled transition system, the task is to decide whether they are equivalent with regard to some behavioural equivalence."

Remark 1.11. It is a nice feature that both the implementation and specification can often be described within the same class of labelled transition systems (e.g. within a class from the PRS-hierarchy). In this case it is sometimes more convenient to redefine the decision problem of equivalence checking as follows.

"Given a labelled transition system together with a pair of states of the system, the task is to decide whether the two states are equivalent."

Since two labelled transition systems can be put side-by-side to form a single transition system (by taking a disjoint union of the systems), the reformulated decision problem is equivalent to the one defined above.

In general, the equivalence checking problem may become undecidable either because the model of infinite-state processes is too powerful or by the selection of the behavioural equivalence. It can happen that for a given class of infinite-state processes one equivalence is decidable while another one is not. Hence the decidability question is usually parametrized by two orthogonal dimensions:

- a class of considered transition systems
- a choice of behavioural equivalence.

In this thesis the selection of the class of transition systems ranges over the classes from the PRS-hierarchy and such transition systems will be tested with regard to strong and weak bisimulation equivalence.

1.5.2 Deciding Equivalences with a Finite-State Process

There are several examples of systems where a complex implementation of the systems is supposed to exhibit only a finite-state behaviour. In this case we get a simplified version of equivalence checking where one of the tested labelled transition systems comes from a class of infinite-state processes and the other one represents a given finite-state process. The decision question is again whether the two transition systems are equivalent with regard to some behavioural equivalence.

Questions of this nature are interesting because they relate behaviours of infinite-state systems with their finite-state specifications. Moreover, recent development showed that many of these problems become computationally feasible and in some instances they are solvable even in polynomial time. A complete overview is presented in Chapter 8.

1.5.3 Deciding Regularity

The last question we will ask is whether a given infinite-state process can be described by an equivalent finite-state process, i.e., whether there exists a process with finitely many reachable states which is equivalent to the given infinite-state process. Whenever a process satisfies this property we call it a *regular process* with regard to the considered notion of behavioural equivalence.

The interest in regularity checking is based on the fact that e.g. for bisimilarity checking of finite-state processes we already have efficient polynomial time algorithms [102, 144]. A positive answer to the regularity question for a given pair of infinite-state processes, together with the possibility of algorithmic construction of bisimilar finite-state systems, provides an immediate answer to bisimilarity checking between the original processes.

1.6 Key Results and Techniques

In this section we will briefly mention some of the equivalence checking results that we consider the most important and influential. Large surveys of the results and techniques are provided in [137, 34, 110].

1.6.1 Bisimilarity Between Infinite-State Processes

The first positive decidability result for a class of infinite-state processes is by Baeten, Bergstra and Klop [10, 12]. They exploited periodicity of context-free grammars without redundant nonterminals (normed BPA processes) to show that strong bisimilarity is a decidable equivalence in this class. Simpler proofs relying on structural properties of strong bisimilarity in the class of normed BPA were later provided by Caucal [38, 39] (he introduced the technique of bisimulation bases), Hüttel and Stirling [83] (using the tableau technique), and Groote [63]. Huynh and Tian [84] gave a nondeterministic algorithm which relies on NP oracle, and Hirshfeld, Jerrum and Moller [74] presented a polynomial time algorithm. Another fast algorithm solving the problem is by Hirshfeld and Moller [76]. Finally, the strong bisimilarity question for the class of arbitrary BPA processes was positively answered by Christensen, Hüttel and Stirling [45, 46]; they showed that bisimilarity can be represented by a finite bisimulation base and the decidability result is obtained as a combination of two semidecision procedures. Hence the result does not imply any complexity upper bound. An elementary decision procedure was later given by Burkart, Caucal and Steffen [35]. The authors claim that straightforward optimizations yield a doubly exponential algorithm. An interesting observation is that these results contrast to the fact that all other equivalences from the linear/branching time hierarchy by van Glabbeek are undecidable even for normed BPA [85, 64].

Considering the parallel case of BPA called BPP, Christensen, Hirshfeld and Moller [43, 44, 42] designed a tableau technique which gives a positive answer to decidability of strong bisimilarity (a nice overview of the tableau technique can be found in [97]). The decidability theorem also follows (as observed by Hirshfeld in [71]) from the fact that any congruence on a finitely generated commutative semigroup is finitely generated. For the restricted class of normed BPP (from every reachable state there is a computation ending in the empty process) even a polynomial time algorithm exists. This is a result by Hirshfeld, Jerrum and Moller [75].

Strong bisimilarity problems in the classes of pushdown processes and Petri nets become significantly harder. Nevertheless, the long standing problem of deciding strong bisimilarity for PDA was answered positively: for the normed case by Stirling [184] and for the general case by Sénizergues [163] (other useful references are [164, 165, 186]). On the other hand, strong bisimilarity of Petri nets appeared to be undecidable. This result is due to Jančar [88, 90] and it is by reduction from the halting problem of Minsky counter machines. The technique developed in his proof has many consequences (see [70, 96]); e.g. the problem of strong bisimilarity for multiset automata (a proper subclass of Petri nets) can be proved undecidable [137, 97] by using Jančar's ideas.

The problem of strong bisimilarity for the whole class of PA-processes is still open, however, for the normed subclass a doubly exponential nondeterministic time algorithm exists due to Hirshfeld and Jerrum [73].

As for weak bisimilarity, many problems are still open. Weak bisimilarity is semilinear for BPP processes [57] and hence semidecidable. Decidability of the problem is open, however, Jančar [92] recently developed a new technique for strong bisimilarity of BPP which indicates that the answer to the weak bisimilarity problem may be also positive. In addition, decidability of weak bisimilarity was introduced for several restricted subclasses of BPP [72, 187]. The weak bisimilarity problem for BPA is also open, however, a positive answer for the restricted subclass of totally normed BPA is provided in [72].

On the other hand, weak bisimilarity of Petri nets becomes even highly undecidable, i.e., it lies beyond the arithmetical hierarchy [89].

1.6.2 Bisimilarity Between Infinite and Finite-State Processes

The problem was first studied for strong bisimilarity between Petri nets and finite-state processes and shown to be decidable by Jančar and Moller [96]. On the other hand, the weak bisimilarity problem already becomes undecidable (Jančar and Esparza [93]). Nevertheless, for BPP and finite-state processes Mayr showed that weak bisimilarity is decidable [120]. This result was later extended to other classes of systems in [86] and [94]. Kučera and Mayr [111, 113] moreover showed that weak bisimilarity between (i) BPA and finite-state processes and (ii) normed BPP and finite-state processes is solvable in polynomial time.

1.6.3 Regularity Problems

Jančar and Esparza [93] showed that given a Petri net, it is decidable whether the net is regular with regard to strong bisimilarity. Burkart, Caucal and Steffen [36] proved the same positive result for the class of BPA processes. On the other hand, weak regularity of Petri nets is already undecidable — a result of Jančar and Esparza [93]. The strong regularity problem for normed subclasses of processes usually corresponds to the boundedness problem (i.e. to the problem whether the set of reachable states is syntactically finite) and is easier to handle. This was demonstrated e.g. by Kučera [107] for normed PA-processes where strong regularity is shown to be decidable in polynomial time. However, for many other classes from the PRS-hierarchy the regularity problem remains still open.

1.6.4 Main Results of the Thesis

Let us conclude this section by mentioning what we consider as two main contributions of the thesis.

The first one is an explicit formulation of a novel technique called existential quantification which enables to give (hopefully) simple and insightful proofs of complexity lower bounds for several process algebras (see [173]). This is demonstrated by showing PSPACE-hardness of strong bisimilarity and regularity for BPA and BPP. Other examples where the technique was already used are EXPTIME-hardness of weak bisimilarity for normed BPA [128] and EXPTIME-hardness of strong bisimilarity for normed PDA [112].

The second main contribution is a solution to two open problems: weak bisimilarity of pushdown processes and weak bisimilarity of PA-processes. Both problems are shown to be undecidable. The result for PDA also directly implies undecidability of strong bisimilarity for prefix-recognizable graphs.

1.7 Thesis Organization

In what follows we describe the structure of this thesis. Every chapter usually starts with a motivation part including the introduction of the considered problems and relevant references for further study, and concludes with two sections:

"Concluding Remarks" where a summary of the chapter and research problems are provided, and "Bibliographical Remarks" where the author's contribution and credits to other sources are stated.

Chapter 2 This chapter offers a formal introduction to the problems studied in this thesis and gives mathematical definitions of terms (like labelled transition system, process rewrite system and strong/weak bisimulation equivalence) which were explained on a semi-formal level in the present chapter. A detailed discussion about several classes from the process rewrite system hierarchy can be also found in this chapter.

Chapter 3 In this chapter the basic notion of labels in transition systems is discussed. A general reduction from labelled transition systems to unlabelled ones is given. The reduction preserves the answers to the strong bisimilarity checking and its usefulness is demonstrated on two classes of infinite-state processes, namely on the sequential model of pushdown processes and on the fully parallel model of Petri nets.

Chapter 4 The tableau technique proved to be a useful method for showing decidability of equivalence checking. In this chapter we extend the technique in several ways. This generalization enables to demonstrate that strong bisimilarity is a decidable equivalence for a large variety of commutative-based labelled transition systems. We demonstrate the fact on classes of deadlock sensitive BPP, lossy BPP processes, BPP with interrupt and timed-arc BPP nets.

Chapter 5 Hardness estimates for weak bisimilarity and regularity checking are studied in this chapter. Basic techniques for proving complexity lower bounds of bisimilarity problems are introduced and these are demonstrated on the classes of BPA and BPP. In particular, we prove that weak bisimilarity of normed BPP is PSPACE-hard, and show how to reduce weak bisimilarity questions to weak regularity checking.

Chapter 6 This chapter further develops some of the techniques from Chapter 5 and strengthens the validity of the results to the strong bisimilarity checking. Formal proofs of PSPACE-hardness of strong bisimilarity and regularity are given for the classes BPA and BPP. NL-completeness of strong regularity problems for the normed subclasses of BPA and BPP is also established.

Chapter 7 This chapter investigates the decidability borders of weak bisimilarity checking within the PRS-hierarchy. Some of the problems which were left open are answered and their undecidability is proved. First, undecidability of weak bisimilarity for pushdown processes is demonstrated by reduction from the halting problem of Minsky counter machines. Second, weak bisimilarity of PA-processes is also shown to be undecidable, this time by reduction from Post's correspondence problem. The proof techniques developed in this chapter have interesting consequences. They e.g. negatively answer the strong bisimilarity problem for prefix-recognizable graphs, and enable to show that weak bisimilarity of pushdown processes lies beyond the arithmetical hierarchy of undecidable problems.

Chapter 8 The last chapter of the thesis gives a comprehensive summary of bisimilarity checking problems for the classes from the PRS-hierarchy. Both normed and unnormed processes are considered and accurate references to the mentioned results are provided.

1.8 Author's Contribution

The overall contribution of the present author to the theoretical research in computer science is summarized in this section (not all of the material is, however, a part of the thesis). As already mentioned before, every chapter of the thesis contains a section called "Bibliographical Remarks" where the author's contribution to the chapter is explicitly stated. This includes relevant references to published papers, as well as remarks about yet unpublished work.

Journal Articles

1. *Strong Bisimilarity of Simple Process Algebras: Complexity Lower Bounds* (by J. Srba). Acta Informatica, 2003. To appear.

2. *Roadmap of Infinite Results* (by J. Srba). Bulletin of the European Association for Theoretical Computer Science, pages 163–175, volume 78, columns: Concurrency, 2002.

3. *Undecidability of Domino Games and Hhp-Bisimilarity* (by M. Jurdzinski, M. Nielsen and J. Srba). Information and Computation, 2002. To appear.

4. *Complexity of Weak Bisimilarity and Regularity for BPA and BPP* (by J. Srba). Mathematical Structures in Computer Science. To appear.

5. *Basic Process Algebra with Deadlocking States* (by J. Srba). Theoretical Computer Science, pages 605–630, volume 266 (1–2), Elsevier Science, 2001.

International Conference and Workshop Proceedings

1. *Undecidability of Weak Bisimilarity for PA-Processes* (by J. Srba). In Proceedings of the 6th International Conference on Developments in Language Theory (DLT'02). LNCS, Springer-Verlag, 2002. To appear.

2. *Undecidability of Weak Bisimilarity for Pushdown Processes* (by J. Srba). In Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02), pages 579–593, volume 2421 of LNCS, Springer-Verlag, 2002.

3. *Strong Bisimilarity and Regularity of Basic Process Algebra is PSPACE-Hard* (by J. Srba). In Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02), pages 716–727, volume 2380 of LNCS, Springer-Verlag, 2002.

4. *Note on the Tableau Technique for Commutative Transition Systems* (by J. Srba). In Proceedings of the 5th Foundations of Software Science and Computation Structures (FOSSACS'02), pages 387–401, volume 2303 of LNCS, Springer-Verlag, 2002.

5. *Strong Bisimilarity and Regularity of Basic Parallel Processes is PSPACE-Hard* (by J. Srba). In Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02), pages 535–546, volume 2285 of LNCS, Springer-Verlag, 2002.

6. *Properties of Distributed Timed-Arc Petri Nets* (by M. Nielsen, V. Sassone and J. Srba). In Proceedings of the 21st International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01), pages 280–291, volume 2245 of LNCS, Springer-Verlag, 2001.

7. *On the Power of Labels in Transition Systems* (by J. Srba). In Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'01), pages 277–291, volume 2154 of LNCS, Springer-Verlag, 2001.

8. *Towards a Notion of Distributed Time for Petri Nets* (by M. Nielsen, V. Sassone and J. Srba). In Proceedings of the 22nd International Conference on Application and Theory of Petri Nets (ICATPN'01), pages 23–31, volume 2075 of LNCS, Springer-Verlag, 2001.

9. *Complexity of Weak Bisimilarity and Regularity for BPA and BPP* (by J. Srba). In Proceedings of the 7th International Workshop on Expressiveness in Concurrency (EXPRESS'00), pages 29–44, 2000.

10. *Matching Modulo Associativity and Idempotency is NP-Complete* (by O. Klíma and J. Srba). In Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS'00), pages 456–466, volume 1893 of LNCS, Springer-Verlag, 2000.

11. *Pattern Equations and Equations with Stuttering* (by I. Černá, O. Klíma and J. Srba). In Proceedings of the 26th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'99), pages 369–378, volume 1725 of LNCS, Springer-Verlag, 1999.

12. *Deadlocking States in Context-Free Process Algebra* (by J. Srba). In Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98), pages 388–398, volume 1450 of LNCS, Springer-Verlag, 1998.

Technical Reports

Eight technical reports, mostly extended versions of conference proceedings.

Other Work

- *Roadmap of Infinite Results.* A project which aims to provide an *updated*, online overview of the state-of-the-art in bisimilarity checking of infinite-state processes. See `http://www.brics.dk/~srba/roadmap/`.
- *Applications of the Existential Quantification Technique* (by J. Srba). In Proceedings of the 4th International Workshop on Verification of Infinite-State Systems (INFINITY'02), pages 151–152, 2002. Short presentation.

This thesis is based on journal papers 1,2 and 4 and on conference (or workshop) proceedings 1,2,3,4,5,7 and 9.

1.9 Bibliographical Remarks

A part of Subsection 1.2.1 introducing the linear/branching time hierarchy is based on the work of van Glabbeek [193, 194], including examples which distinguish between the considered behavioural equivalences. The hierarchy of divergence equivalences described in Figure 1.2 was presented in a paper by Lohrey, D'Argenio and Hermanns [117]. The correspondence between formal language theory and process rewrite systems shown in Subsection 1.3.2 is built on the observations by Esparza [59]. Section 1.4 contains a slightly modified presentation of interprocedural control-flow analysis by means of process rewrite systems, originally described by Esparza and Knoop [61]. Good source of references for Section 1.6 were overview articles by Burkart, Caucal, Moller and Steffen [34] and by Kučera and Jančar [110].

Chapter 2

Basic Definitions

This chapter introduces the basic notions studied in the thesis. We define labelled transition systems together with the notion of a process, we introduce process algebras in a uniform way using the formalism of process rewrite systems, and we formally describe the studied problems.

2.1 Transition Systems and Processes

Semantics to process algebras is usually given in terms of (infinite-state) labelled transition systems [151]. Processes are understood as nodes of certain labelled transition systems and the transition relation is defined in a compositional way.

Definition 2.1 (Labelled transition system).
A *labelled transition system* T is a triple $T = (S, \mathcal{A}ct, \longrightarrow)$ where

- S is a set of *states* (or *processes*),
- $\mathcal{A}ct$ is a set of *labels* (or *actions*), and
- $\longrightarrow \subseteq S \times \mathcal{A}ct \times S$ is a *transition relation*, written $\alpha \xrightarrow{a} \beta$, for $(\alpha, a, \beta) \in\longrightarrow$.

As usual we extend the transition relation to the elements of $\mathcal{A}ct^*$, i.e., $\alpha \xrightarrow{\epsilon} \alpha$ for every $\alpha \in S$, and $\alpha \xrightarrow{aw} \beta$ iff $\alpha \xrightarrow{a} \alpha'$ and $\alpha' \xrightarrow{w} \beta$ for every $\alpha, \beta \in S$, $a \in \mathcal{A}ct$ and $w \in \mathcal{A}ct^*$. We use the notation $\alpha \xrightarrow{a}$ meaning that there is some $\beta \in S$ such that $\alpha \xrightarrow{a} \beta$.

We write $\alpha \longrightarrow^* \beta$ iff $\alpha \xrightarrow{w} \beta$ for some $w \in \mathcal{A}ct^*$. We also write $\alpha \overset{a}{\not\longrightarrow}$ whenever there is no β such that $\alpha \xrightarrow{a} \beta$, and $\alpha \not\longrightarrow$ whenever $\alpha \overset{a}{\not\longrightarrow}$ for all $a \in \mathcal{A}ct$.

We will call a rooted transition system a *process*. This follows the usual definition from process algebras where a system Δ of defining equations or rewrite rules is given and a process is understood as a state of the transition system generated by Δ.

Definition 2.2 (Process and its reachable states).
A *process* is a pair (α, T) where $T = (S, \mathcal{A}ct, \longrightarrow)$ is a labelled transition system and $\alpha \in S$. We say that $\beta \in S$ is *reachable in* (α, T) iff $\alpha \longrightarrow^* \beta$.

Definition 2.3. Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system and $\alpha \in S$. By T_α we denote a labelled transition system restricted to the states of T reachable from α. More precisely, $T_\alpha \stackrel{\text{def}}{=} (S_\alpha, \mathcal{A}ct, \longrightarrow_\alpha)$ where $S_\alpha \stackrel{\text{def}}{=} \{\alpha' \in S \mid \alpha \longrightarrow^* \alpha'\}$ and $\alpha_1 \xrightarrow{a}_\alpha \alpha_2$ iff $\alpha_1 \xrightarrow{a} \alpha_2$ and $\alpha_1, \alpha_2 \in S_\alpha$.

Definition 2.4 (Finite-state process).
Whenever a process (α, T) has only finitely many reachable states, we call it a *finite-state process.*

Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system such that $\mathcal{A}ct$ contains a distinguished *silent* action τ. Actions from $\mathcal{A}ct \smallsetminus \{\tau\}$ will be called *visible* actions. We define a *weak transition relation* $\Longrightarrow \subseteq S \times \mathcal{A}ct \times S$ such that $\Longrightarrow$ respects the relation $\longrightarrow$ on visible actions but allows to collapse all the τ-transitions.

$$\stackrel{a}{\Longrightarrow} \stackrel{\text{def}}{=} \begin{cases} (\xrightarrow{\tau})^* \circ \xrightarrow{a} \circ (\xrightarrow{\tau})^* & \text{if } a \neq \tau \\ (\xrightarrow{\tau})^* & \text{if } a = \tau. \end{cases}$$

As before we extend the weak transition relation to the elements of $\mathcal{A}ct^*$ and use the notation $\Longrightarrow^*$ and $\stackrel{a}{\not\Longrightarrow}$.

Example 2.1. Let $T \stackrel{\text{def}}{=} (\{s_1, s_2, s_3, s_4\}, \{a, b, \tau\}, \longrightarrow)$ such that

$$s_1 \xrightarrow{\tau} s_2 \qquad s_2 \xrightarrow{a} s_3 \qquad s_3 \xrightarrow{\tau} s_4 \qquad s_3 \xrightarrow{b} s_4.$$

Now e.g. $s_1 \stackrel{a}{\Longrightarrow} s_4$, $s_1 \stackrel{ab}{\Longrightarrow} s_4$, $s_2 \stackrel{\tau}{\Longrightarrow} s_2$, $s_1 \Longrightarrow^* s_4$, and $s_1 \stackrel{b}{\not\Longrightarrow}$. $\square$

2.2 Strong and Weak Bisimilarity

In this section we define strong and weak bisimilarity and introduce a game-theoretic characterization of bisimilarity.

Definition 2.5 (Strong bisimulation).
Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system. A binary relation $R \subseteq S \times S$ is a *strong bisimulation* iff whenever $(\alpha, \beta) \in R$ then for each $a \in \mathcal{A}ct$:

- if $\alpha \xrightarrow{a} \alpha'$ then $\beta \xrightarrow{a} \beta'$ for some β' such that $(\alpha', \beta') \in R$
- if $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ for some α' such that $(\alpha', \beta') \in R$.

Definition 2.6 (Weak bisimulation).
Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system (such that $\mathcal{A}ct$ possibly contains the distinguished silent action τ). A binary relation $R \subseteq S \times S$ is a *weak bisimulation* iff whenever $(\alpha, \beta) \in R$ then for each $a \in \mathcal{A}ct$:

- if $\alpha \xrightarrow{a} \alpha'$ then $\beta \stackrel{a}{\Longrightarrow} \beta'$ for some β' such that $(\alpha', \beta') \in R$
- if $\beta \xrightarrow{a} \beta'$ then $\alpha \stackrel{a}{\Longrightarrow} \alpha'$ for some α' such that $(\alpha', \beta') \in R$.

Remark 2.1. It is possible to give an alternative definition of weak bisimilarity as follows. A binary relation $R \subseteq S \times S$ is a *weak bisimulation* iff whenever $(\alpha, \beta) \in R$ then for each $a \in \mathcal{A}ct$:

- if $\alpha \overset{a}{\Longrightarrow} \alpha'$ then $\beta \overset{a}{\Longrightarrow} \beta'$ for some β' such that $(\alpha', \beta') \in R$
- if $\beta \overset{a}{\Longrightarrow} \beta'$ then $\alpha \overset{a}{\Longrightarrow} \alpha'$ for some α' such that $(\alpha', \beta') \in R$.

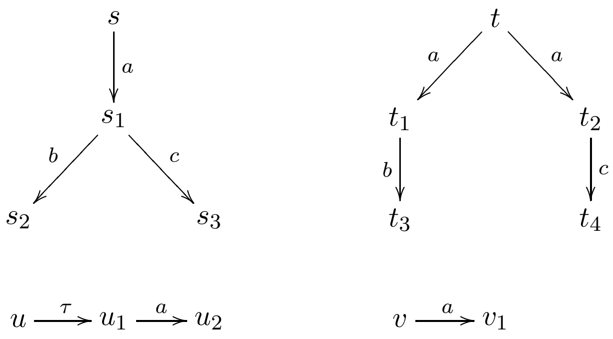
It is straightforward (see e.g. [135]) to realise the this definition of weak bisimilarity defines the same notion as Definition 2.6.

Processes (α_1, T) and (α_2, T) are *strongly* (resp. *weakly*) *bisimilar*, written $(\alpha_1, T) \sim (\alpha_2, T)$ (resp. $(\alpha_1, T) \approx (\alpha_2, T)$) or simply $\alpha_1 \sim \alpha_2$ (resp. $\alpha_1 \approx \alpha_2$) if T is clear from the context, iff there is a strong (resp. weak) bisimulation R such that $(\alpha_1, \alpha_2) \in R$.

Example 2.2. Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition systems such that $S \overset{\text{def}}{=} \{s, s_1, s_2, s_3, t, t_1, t_2, t_3, t_4, u, u_1, u_2, v, v_1\}$, $\mathcal{A}ct \overset{\text{def}}{=} \{a, b, c, \tau\}$ and where

$$\begin{array}{llll}
s \xrightarrow{a} s_1 & s_1 \xrightarrow{b} s_2 & s_1 \xrightarrow{c} s_3 & \\
t \xrightarrow{a} t_1 & t \xrightarrow{a} t_2 & t_1 \xrightarrow{b} t_3 & t_2 \xrightarrow{c} t_4 \\
 & & & \\
u \xrightarrow{\tau} u_1 & u_1 \xrightarrow{a} u_2 & & \\
v \xrightarrow{a} v_1. & & &
\end{array}$$

The transition system T has the following graphical representation.



$$\begin{tikzcd}
& s \arrow[d, "a"] & & & t \arrow[dl, "a"'] \arrow[dr, "a"] & \\
& s_1 \arrow[dl, "b"'] \arrow[dr, "c"] & & t_1 \arrow[d, "b"'] & & t_2 \arrow[d, "c"] \\
s_2 & & s_3 & t_3 & & t_4
\end{tikzcd}$$

$$\begin{tikzcd}
u \arrow[r, "\tau"] & u_1 \arrow[r, "a"] & u_2 & & v \arrow[r, "a"] & v_1
\end{tikzcd}$$

This is a standard example demonstrating that $(s, T) \not\sim (t, T)$ even if the processes (s, T) and (t, T) exhibit the same sequences of actions, namely $\{\epsilon, a, ab, ac\}$. In order to see that $(s, T) \not\sim (t, T)$ it is enough to enumerate all possible binary relations of S and check that none of these is a strong bisimulation containing the pair (s, t). In fact, we do not have to consider the states u, u_1, u_2, v and v_1 since these states are not reachable from s or t. In Definition 2.7 we will introduce a game characterization of bisimilarity which gives a simpler argument for $(s, T) \not\sim (t, T)$.

Let us now consider the processes (u, T) and (v, T). Obviously $(u, T) \not\sim (v, T)$. On the other hand $(u, T) \approx (v, T)$ because $R \overset{\text{def}}{=} \{(u, v), (u_1, v), (u_2, v_1)\}$ is a weak bisimulation such that $(u, v) \in R$. $\square$

Given a pair of processes (α_1, T_1) and (α_2, T_2) where $T_1 = (S_1, \mathcal{A}ct_1, \longrightarrow_1)$ and $T_2 = (S_2, \mathcal{A}ct_2, \longrightarrow_2)$ such that $S_1 \cap S_2 = \emptyset$, we write $(\alpha_1, T_1) \sim (\alpha_2, T_2)$ (resp. $(\alpha_1, T_1) \approx (\alpha_2, T_2)$) iff $(\alpha_1, T) \sim (\alpha_2, T)$ (resp. $(\alpha_1, T) \approx (\alpha_2, T)$) such that $T \overset{\text{def}}{=} (S_1 \cup S_2, \mathcal{A}ct_1 \cup \mathcal{A}ct_2, \longrightarrow)$ where $\alpha \xrightarrow{a} \beta$ iff $\alpha, \beta \in S_1$ and $\alpha \xrightarrow{a}_1 \beta$, or $\alpha, \beta \in S_2$ and $\alpha \xrightarrow{a}_2 \beta$.

Example 2.3. Let $T_1 = (\{s,t\},\{a,b\},\longrightarrow_1)$ and $T_2 = (\{t\},\{a,c\},\longrightarrow_2)$ be labelled transition systems such that $s \xrightarrow{a}_1 t$, $t \xrightarrow{b}_1 t$, and $t \xrightarrow{a}_2 t$, $t \xrightarrow{c}_2 t$. In order to ask questions like $(s,T_1) \sim (t,T_2)$ we first have to rename the states of e.g. T_2 such that $\overline{T}_2 = (\{\overline{t}\},\{a,c,\},\longrightarrow_{\overline{2}})$ where $\overline{t} \xrightarrow{a}_{\overline{2}} \overline{t}$ and $\overline{t} \xrightarrow{c}_{\overline{2}} \overline{t}$. Now the states of T_1 and $\overline{T}_2$ are disjoint and we can verify whether $(s,T_1) \sim (\overline{t},\overline{T}_2)$ by the definition given above. □

The following proposition points out the relationship between strong and weak bisimilarity.

Proposition 2.1 ([135]). Let (α_1,T) and (α_2,T) be a pair of processes. If $(\alpha_1,T) \sim (\alpha_2,T)$ then also $(\alpha_1,T) \approx (\alpha_2,T)$.

Remark 2.2. Example 2.2 shows that the opposite direction of Proposition 2.1 does not hold.

Bisimulation equivalence has an elegant characterisation in terms of *bisimulation games.*

Definition 2.7 (Strong and weak bisimulation game).
A *strong* (resp. *weak*) *bisimulation game* on a pair of processes (α_1,T) and (α_2,T) where $T = (S,\mathcal{A}ct,\longrightarrow)$ is a two-player game of an 'attacker' and a 'defender'. The game is played in *rounds* on pairs of states from $S \times S$. In each round the players change the *current states* β_1 and β_2 (initially α_1 and α_2) according to the following rule.

1. The attacker chooses $i \in \{1,2\}$, $a \in \mathcal{A}ct$ and $\beta_i' \in S$ such that $\beta_i \xrightarrow{a} \beta_i'$.
2. The defender responds by choosing $\beta_{3-i}' \in S$ such that $\beta_{3-i} \xrightarrow{a} \beta_{3-i}'$ in case of the strong game (resp. $\beta_{3-i} \overset{a}{\Longrightarrow} \beta_{3-i}'$ in case of the weak game).
3. The states β_1' and β_2' become the current states.

A *play* is a maximal sequence of pairs of states formed by the players according to the rule described above, and starting from the initial states α_1 and α_2. The defender is the winner in every infinite play. A finite play is lost by the player who is stuck. Note that the attacker gets stuck in current states β_1 and β_2 if and only if both $\beta_1 \not\longrightarrow$ and $\beta_2 \not\longrightarrow$.

We remind the reader of the fact that if the attacker chooses a move under the action τ in one of the processes (in the weak bisimulation game), the defender can (as one possibility) simply answer by doing "nothing", i.e., by staying in the same state of the other process. The following proposition is a standard one (see e.g. [183, 190]).

Proposition 2.2. Processes (α_1,T) and (α_2,T) are strongly (resp. weakly) bisimilar if and only if the defender has a winning strategy in the strong (resp. weak) bisimulation game starting from α_1 and α_2 (and nonbisimilar iff the attacker has a winning strategy in the corresponding game).

Example 2.4. Let us consider the processes (s,T) and (t,T) from Example 2.2. In order to show that $(s,T) \not\sim (t,T)$ it is enough to demonstrate that the attacker has a winning strategy in the strong bisimulation game played from the pair s and t. In the first round the attacker plays e.g. $t \xrightarrow{a} t_1$ and the defender can only answer by $s \xrightarrow{a} s_1$. Now the game continues from the pair s_1 and t_1. In the second round the attacker plays $s_1 \xrightarrow{c} s_3$ and the defender loses because $t_1 \overset{c}{\not\longrightarrow}$. □

Another important notion is the definition of regularity (finiteness) of a given process.

Definition 2.8 (Strong and weak regularity).
We say that a process (α,T) is *strongly* (resp. *weakly*) *regular* iff there exists some finite-state process strongly (resp. weakly) bisimilar to it.

Example 2.5. Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system such that $S \overset{\text{def}}{=} \{s_i \mid i \in \mathbb{N}_0\} \cup \{t_i \mid i \in \mathbb{N}_0\} \cup \{u\}$, $\mathcal{A}ct \overset{\text{def}}{=} \{a,\tau\}$, and

$$\begin{array}{lll} s_i \xrightarrow{a} s_{i+1} & & \text{for all } i \in \mathbb{N}_0 \\ t_i \xrightarrow{a} t_{i+1} & t_{i+1} \xrightarrow{\tau} t_i & \text{for all } i \in \mathbb{N}_0 \\ u \xrightarrow{a} u. & & \end{array}$$

A picture of a fragment of the transition system T follows.

$$\begin{tikzcd}
s_0 \arrow[r, "a"] & s_1 \arrow[r, "a"] & s_2 \arrow[r, "a"] & s_3 \arrow[r, "a"] & \cdots \\
t_0 \arrow[r, bend left, "a"] & t_1 \arrow[l, bend left, "\tau"] \arrow[r, bend left, "a"] & t_2 \arrow[l, bend left, "\tau"] \arrow[r, bend left, "a"] & t_3 \arrow[l, bend left, "\tau"] \arrow[r, bend left, "a"] & \cdots \arrow[l, bend left, "\tau"] \\
u \arrow[loop right, "a"] & & & &
\end{tikzcd}$$

The process (s_0,T) is strongly regular because it is strongly bisimilar to the finite-state process (u,T). The process (t_0,T) is not strongly regular because it has infinitely many reachable and pairwise nonbisimilar states: $(t_i,T) \not\sim (t_j,T)$ for any $i,j \in \mathbb{N}_0$ such that $i \neq j$. On the other hand (t_0,T) is weakly regular because $(t_0,T) \approx (u,T)$. □

2.3 Process Rewrite Systems

Let $\mathcal{A}ct$ and $\mathcal{C}onst$ be sets of *actions* and *process constants*, respectively.

Definition 2.9 (Classes of process expressions).
The classes of *process expressions* called 1 (process constants plus the empty process), $\mathcal{P}$ (parallel process expressions), $\mathcal{S}$ (sequential process expressions), and $\mathcal{G}$ (general process expressions) are defined by the following abstract syntax

$$1: \quad E ::= \epsilon \mid X$$

$$\mathcal{P}: \quad E ::= \epsilon \mid X \mid E \| E$$

$$\begin{tikzcd}
 & \mathcal{G} \arrow[dl, dash] \arrow[dr, dash] & \\
\mathcal{S} \arrow[dr, dash] & & \mathcal{P} \arrow[dl, dash] \\
 & 1 &
\end{tikzcd}$$

Figure 2.1: Classes of process expressions

$$\mathcal{S}: \qquad E \ ::= \ \epsilon \ \mid \ X \ \mid \ E.E$$

$$\mathcal{G}: \qquad E \ ::= \ \epsilon \ \mid \ X \ \mid \ E\|E \ \mid \ E.E$$

where 'ϵ' is the *empty process*, X ranges over $\mathcal{C}onst$, the operator '.' stands for a *sequential composition* and '$\|$' stands for a *parallel composition.* We shall adopt the convention that the sequential operator binds tighter than the parallel one. Thus for example $X\|Y.Z$ means $X\|(Y.Z)$.

Obviously, $1 \subset \mathcal{S}$, $1 \subset \mathcal{P}$, $\mathcal{S} \subset \mathcal{G}$ and $\mathcal{P} \subset \mathcal{G}$. The classes $\mathcal{S}$ and $\mathcal{P}$ are incomparable and $\mathcal{S} \cap \mathcal{P} = 1$. See Figure 2.1.

Notation 2.1. We use the notation $\mathcal{G}(\mathcal{C}onst)$, $\mathcal{S}(\mathcal{C}onst)$, $\mathcal{P}(\mathcal{C}onst)$ and $1(\mathcal{C}onst)$ whenever we need to explicitly specify from which process constants the expressions are formed.

Definition 2.10 (Structural congruence).
We do not distinguish between process expressions related by a *structural congruence* $\equiv$, which is the smallest congruence over process expressions such that the following laws hold:

- '.' is associative,
- '$\|$' is associative and commutative, and
- 'ϵ' is a unit for '.' and '$\|$'.

Example 2.6. A process expression $X.\epsilon.Y\|Z\|Y\|\epsilon$ is structurally equivalent to the expression $Y\|Z\|X.Y$. $\square$

We can now define the main notion of process rewrite systems following Mayr's approach from [126].

Definition 2.11 (Process rewrite system (PRS)).
Let $\alpha, \beta \in \{1, \mathcal{S}, \mathcal{P}, \mathcal{G}\}$ such that $\alpha \subseteq \beta$. An (α, β)-PRS is a finite set

$$\Delta \subseteq (\alpha \smallsetminus \{\epsilon\}) \times \mathcal{A}ct \times \beta$$

of *rewrite rules*, written $E \xrightarrow{a} F$ for $(E, a, F) \in \Delta$.

Notation 2.2. Let us denote the set of actions and process constants that appear in Δ by $\mathcal{A}ct(\Delta)$ and $\mathcal{C}onst(\Delta)$, respectively. Note that $\mathcal{A}ct(\Delta)$ and $\mathcal{C}onst(\Delta)$ are finite sets.

$$\frac{(E \xrightarrow{a} E') \in \Delta}{E \xrightarrow{a} E'} \qquad \frac{E \xrightarrow{a} E'}{E.F \xrightarrow{a} E'.F} \qquad \frac{E \xrightarrow{a} E'}{E\|F \xrightarrow{a} E'\|F}$$

Figure 2.2: SOS rules

Example 2.7. Let Δ be an $(\mathcal{S},\mathcal{G})$-PRS such that

$$\Delta \overset{\text{def}}{=} \{X.Y \xrightarrow{a} X\|Y.Z,\ Z.Z \xrightarrow{b} \epsilon\}.$$

Then $\mathcal{C}onst(\Delta) = \{X, Y, Z\}$ and $\mathcal{A}ct(\Delta) = \{a, b\}$. $\square$

We give a formal semantics to PRS by means of labelled transition systems.

Definition 2.12 (Transition system $T(\Delta)$).
Let Δ be an (α,β)-PRS. The system Δ determines a labelled transition system $T(\Delta) \overset{\text{def}}{=} (\beta, \mathcal{A}ct(\Delta), \longrightarrow)$, where *states* are process expressions from the class β (modulo the structural congruence introduced in Definition 2.10), $\mathcal{A}ct(\Delta)$ is the set of *labels*, and the *transition relation* is the least relation satisfying the SOS (structural operational semantics) rules from Figure 2.2 — recall that '$\|$' is commutative.

Definition 2.13 ((α,β)-process).
An (α,β)-*process* is a process $\big(P, T(\Delta)\big)$ — see Definition 2.2 — where Δ is an (α,β)-PRS and $P \in \beta$ is a process expression.

Remark 2.3. We remind the reader of the fact that process notions like finiteness, bisimilarity and regularity introduced in Sections 2.1 and 2.2 define the corresponding process properties also for (α,β)-processes. Moreover, in the rest of this thesis we denote an (α,β)-process $\big(P, T(\Delta)\big)$ by only (P,Δ), or even P if Δ is clear from the context.

Many classes of infinite-state systems studied so far — e.g. basic process algebra (BPA), basic parallel processes (BPP), pushdown processes (PDA), Petri nets (PN) and process algebra (PA) — are contained in the hierarchy of process rewrite systems presented in Figure 2.3. This hierarchy is strict w.r.t. strong bisimilarity and we refer the reader to [126] for further discussions. It is worth mentioning that even the class of $(\mathcal{G},\mathcal{G})$-PRS is not Turing powerful since e.g. the reachability problem remains decidable [126].

Definition 2.14 (Normed (α,β)-process).
An (α,β)-process (P,Δ) is *normed* iff from every reachable state E in (P,Δ) there is a computation terminating in the empty process, i.e., for all E such that $P \longrightarrow^* E$ it is the case that $E \longrightarrow^* \epsilon$.

Definition 2.15 (Totally normed (α,β)-process).
An (α,β)-process (P,Δ) is *totally normed* iff it is normed and for every reachable state E in (P,Δ) such that $E \not\equiv \epsilon$ holds that $E \overset{\tau}{\not\Longrightarrow} \epsilon$.

$$\begin{tikzcd}
 & & \begin{array}{c}(\mathcal{G},\mathcal{G})\text{-PRS}\\ \text{PRS}\end{array} \arrow[dl, -] \arrow[dr, -] & & \\
 & \begin{array}{c}(\mathcal{S},\mathcal{G})\text{-PRS}\\ \text{PAD}\end{array} \arrow[dl, -] \arrow[dr, -] & & \begin{array}{c}(\mathcal{P},\mathcal{G})\text{-PRS}\\ \text{PAN}\end{array} \arrow[dl, -] \arrow[dr, -] & \\
\begin{array}{c}(\mathcal{S},\mathcal{S})\text{-PRS}\\ \text{PDA}\end{array} \arrow[dr, -] & & \begin{array}{c}(1,\mathcal{G})\text{-PRS}\\ \text{PA}\end{array} \arrow[dl, -] \arrow[dr, -] & & \begin{array}{c}(\mathcal{P},\mathcal{P})\text{-PRS}\\ \text{PN}\end{array} \arrow[dl, -] \\
 & \begin{array}{c}(1,\mathcal{S})\text{-PRS}\\ \text{BPA}\end{array} \arrow[dr, -] & & \begin{array}{c}(1,\mathcal{P})\text{-PRS}\\ \text{BPP}\end{array} \arrow[dl, -] & \\
 & & \begin{array}{c}(1,1)\text{-PRS}\\ \text{FS}\end{array} & &
\end{tikzcd}$$

Figure 2.3: Hierarchy of process rewrite systems

Remark 2.4. In some papers, the definition of normedness requires only the fact that from every reachable state there is a terminating computation, not necessarily ending in the empty process. However, e.g. for BPA, in order to achieve a reasonable notion of normedness, it is also assumed that every process constant used in the system can perform a transition, i.e., it has at least one rewrite rule associated to it. This alternative definition implies the notion of normedness introduced in Definition 2.14. Moreover, the definition we gave becomes more interesting even for the models like PDA, PN, PAD, PAN and PRS where our notion of normedness guarantees deadlock freedom (here the empty process is not understood as a deadlock). This means that e.g. in the case of PDA the stack can always be emptied, and in the case of PN all tokens in places can be removed. Considering simply the possibility of termination without reaching the empty process usually does not restrict the power of the models sufficiently.

2.4 Studied Problems

In this section we define the basic decidability problems studied in this thesis.

Problem:	Strong Bisimilarity
Instance:	An (α,β)-PRS Δ and a pair of processes (P_1,Δ) and (P_2,Δ).
Question:	$(P_1,\Delta) \sim (P_2,\Delta)$?

Problem:	Weak Bisimilarity
Instance:	An (α,β)-PRS Δ and a pair of processes (P_1,Δ) and (P_2,Δ).
Question:	$(P_1,\Delta) \approx (P_2,\Delta)$?

Problem:	Strong Regularity
Instance:	An (α,β)-PRS Δ and a process (P,Δ).
Question:	Is there a finite-state process (F,Δ') such that $(P,\Delta) \sim (F,\Delta')$?

Problem:	Weak Regularity
Instance:	An (α,β)-PRS Δ and a process (P,Δ).
Question:	Is there a finite-state process (F,Δ') such that $(P,\Delta) \approx (F,\Delta')$?

Problem:	Strong Bisimilarity with a Finite-State Process
Instance:	An (α,β)-PRS Δ with a process (P,Δ), and a finite-state process (F,Δ').
Question:	$(P,\Delta) \sim (F,\Delta')$?

Problem:	Weak Bisimilarity with a Finite-State Process
Instance:	An (α,β)-PRS Δ with a process (P,Δ), and a finite-state process (F,Δ').
Question:	$(P,\Delta) \approx (F,\Delta')$?

2.5 Classes from PRS-Hierarchy

In this section we discuss in more detail the classes from the PRS-hierarchy and emphasize some features relevant for this thesis.

2.5.1 Finite-State Processes

Finite-state processes (FS) — or equivalently $(1,1)$-PRS — generate a class of transition systems with finitely many reachable states. Let Δ be a FS system. Every rewrite rule from Δ is of the form

$$X \xrightarrow{a} Y \quad \text{or} \quad X \xrightarrow{a} \epsilon$$

where $X, Y \in \mathcal{C}onst(\Delta)$ and $a \in \mathcal{A}ct(\Delta)$. From the finiteness of Δ it follows that every FS process (P, Δ) has only finitely many reachable states. In fact the number of reachable states is bounded by $|\mathcal{C}onst(\Delta)| + 1$.

The FS class is interesting from many points of view. In the context of decidability of behavioural equivalences, however, many of the studied problems become trivial. The answer to the regularity question is always positive and both strong and weak bisimilarity are easily seen to be decidable. There are even efficient polynomial time algorithms [102, 144]. Moreover, the problem of strong (and hence also of weak) bisimilarity for FS is known to be P-hard [17].

Example 2.8. Consider a system Δ where $\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{X, Y, X', Y', Z'\}$ and $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{a, b\}$.

$$X \xrightarrow{a} Y \qquad Y \xrightarrow{a} Y \qquad Y \xrightarrow{b} \epsilon$$

$$X' \xrightarrow{a} Y' \quad Y' \xrightarrow{a} Y' \quad X' \xrightarrow{a} Z' \quad Z' \xrightarrow{b} \epsilon$$

The system Δ generates the following labelled transition system.

$$\begin{tikzcd}
X \arrow[r, "a"] & Y \arrow[loop right, "a"] \arrow[d, "b"] & X' \arrow[r, "a"] \arrow[d, "a"] & Y' \arrow[loop right, "a"] \\
 & \epsilon & Z' \arrow[l, "b"'] &
\end{tikzcd}$$

We can now show that $(X, \Delta) \not\sim (X', \Delta)$. The attacker in the strong bisimulation game starting from X and X' has e.g. the following winning strategy: in the first round the attacker plays $X' \xrightarrow{a} Z'$ and the defender has only one possible answer $X \xrightarrow{a} Y$. In the second round played from Y and Z' the attacker performs the action a by using the rule $Y \xrightarrow{a} Y$ and the defender loses since $Z' \overset{a}{\not\longrightarrow}$.

In order to see that X and X' are not strongly bisimilar it is in fact enough to realize that (X, Δ) is a normed process whereas (X', Δ) is not (a process constant Y' is reachable from X' and $Y' \not\longrightarrow^{*} \epsilon$). $\square$

2.5.2 Basic Process Algebra

Basic process algebra (BPA) — or equivalently $(1, \mathcal{S})$-PRS — represents the class of processes introduced by Bergstra and Klop (see [21]). BPA is a model of purely sequential process behaviour. This class also corresponds to the transition systems associated with context-free grammars in Greibach normal form (GNF), in which only left-most derivations are allowed.

Let Δ be a BPA system. Every rewrite rule from Δ is of the form

$$X \xrightarrow{a} E$$

where $X \in \mathcal{C}onst(\Delta)$, $a \in \mathcal{A}ct(\Delta)$ and $E \in \mathcal{S}(\mathcal{C}onst(\Delta))$. It is usually assumed that for each $X \in \mathcal{C}onst(\Delta)$ there is at least one rewrite rule in Δ, i.e., that there is some $a \in \mathcal{A}ct(\Delta)$ and $E \in \mathcal{S}\big(\mathcal{C}onst(\Delta)\big)$ such that $(X, a, E) \in \Delta$. If it is not the case, we say that the system contains *deadlocks*. A study of decidability problems for BPA with deadlocks is provided in [169].

Notation 2.3. Let m be a natural number and $A \in \mathcal{C}onst$ be a process constant. Whenever it is clear from the context that the '.' operator is considered, we use the notation A^m for a sequential composition of m occurrences of A, i.e., $A^0 \overset{\text{def}}{=} \epsilon$ and $A^{m+1} \overset{\text{def}}{=} A^m.A$.

Using the sequential composition we can e.g. model a simple behaviour of a counter.

Example 2.9. Let $\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{Z, C\}$ and $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{\mathit{zero},\ \mathit{inc},\ \mathit{dec}\}$. Assume that Δ consists of the rewrite rules:

$$Z \xrightarrow{\mathit{zero}} Z \qquad Z \xrightarrow{\mathit{inc}} C.Z \qquad C \xrightarrow{\mathit{inc}} C.C \qquad C \xrightarrow{\mathit{dec}} \epsilon.$$

The following picture shows a fragment of the transition system generated by the process (Z, Δ).

$$\begin{tikzcd}
Z \arrow[loop left, "\mathit{zero}"] \arrow[r, bend left, "\mathit{inc}"] & C.Z \arrow[l, bend left, "\mathit{dec}"] \arrow[r, bend left, "\mathit{inc}"] & C^2.Z \arrow[l, bend left, "\mathit{dec}"] \arrow[r, bend left, "\mathit{inc}"] & C^3.Z \arrow[l, bend left, "\mathit{dec}"] \arrow[r, bend left, "\mathit{inc}"] & \cdots \arrow[l, bend left, "\mathit{dec}"]
\end{tikzcd}$$

From the process Z we can increment the value of our counter (represented by the number of occurrences of the process constant C) by performing the action '*inc*'. The counter can be decremented by the action '*dec*' and the action '*zero*' can be used only if the counter is empty. □

Another simple BPA system is presented in the following example.

Example 2.10. Let $\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{Q_1, Q_2, \dots, Q_k\}$ for a natural number $k > 0$ and let $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{a\}$. Consider the following BPA system Δ containing the rewrite rules:

$$\begin{array}{ll} Q_1 \xrightarrow{a} \epsilon & \\ Q_{j+1} \xrightarrow{a} Q_j & \qquad \text{for all } j,\ 1 \le j < k. \end{array}$$

Observe that $Q_j \xrightarrow{a^j} \epsilon$ for every j, $1 \le j \le k$, and no other transitions are possible. Also notice that e.g. $(Q_1^5, \Delta) \sim (Q_1.Q_2^2, \Delta)$.

Assume now that $m_1, m_2 > 0$ are natural numbers. For every ℓ_1, $1 \le \ell_1 \le m_1$, and every ℓ_2, $1 \le \ell_2 \le m_2$, let $i_{\ell_1} \in \{1, 2, \dots, k\}$ and $j_{\ell_2} \in \{1, 2, \dots, k\}$. It is an easy observation that

$$(Q_{i_1}.Q_{i_2}.\cdots.Q_{i_{m_1}}, \Delta) \sim (Q_{j_1}.Q_{j_2}.\cdots.Q_{j_{m_2}}, \Delta)$$

if and only if

$$\sum_{\ell_1=1}^{m_1} i_{\ell_1} = \sum_{\ell_2=1}^{m_2} j_{\ell_2}.$$

This example demonstrates that even though the BPA class is non-commutative, we can achieve a restricted commutative behaviour by assuming that $\mathcal{A}ct(\Delta)$ is a singleton set and by encoding process constants in this unary alphabet. □

2.5.3 Basic Parallel Processes

Basic parallel processes (BPP) — or equivalently $(1, \mathcal{P})$-PRS — are a fragment of CCS [135] without restriction, relabelling and communication. It is a parallel analogue to BPA. BPP class was first studied by Christensen [42], and it is equivalent to the communication-free subclass of Petri nets (each transition has exactly one input place). The classes BPA and BPP are also called *simple process algebras*.

Let Δ be a BPP system. Every rewrite rule from Δ is of the form

$$X \xrightarrow{a} E$$

where $X \in \mathcal{C}onst(\Delta)$, $a \in \mathcal{A}ct(\Delta)$ and $E \in \mathcal{P}(\mathcal{C}onst(\Delta))$. Unlike for BPA, the presence of deadlocks in BPP systems is not essential. Assume that $D \in \mathcal{C}onst(\Delta)$ is a deadlock, i.e., $D \not\longrightarrow$. Then $(E, \Delta) \sim (E\|D, \Delta)$ for any expression $E \in \mathcal{P}\big(\mathcal{C}onst(\Delta)\big)$ and we can safely replace all occurrences of such deadlocks in Δ by the empty process ϵ.

Notation 2.4. Let m be a natural number and $A \in \mathcal{C}onst$ be a process constant. Whenever it is clear from the context that we consider only the '$\|$' operator, we use the notation A^m for a parallel composition of m occurrences of A, i.e., $A^0 \overset{\text{def}}{=} \epsilon$ and $A^{m+1} \overset{\text{def}}{=} A^m\|A$.

We can try to design a system which will model a counter behaviour, similarly as for BPA.

Example 2.11. Let $\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{Z, C\}$ and $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{\mathit{zero},\ \mathit{inc},\ \mathit{dec}\}$. Assume that Δ consists of the rewrite rules:

$$Z \xrightarrow{zero} Z \qquad Z \xrightarrow{inc} C\|Z \qquad C \xrightarrow{inc} C\|C \qquad C \xrightarrow{dec} \epsilon.$$

A fragment of the transition system generated by the process (Z, Δ) follows.

$$\begin{tikzcd}
Z \arrow[loop left, "zero"] \arrow[r, bend left, "inc"] & C\|Z \arrow[l, bend left, "dec"] \arrow[loop above, "zero"] \arrow[r, bend left, "inc"] & C^2\|Z \arrow[l, bend left, "dec"] \arrow[loop above, "zero"] \arrow[r, bend left, "inc"] & C^3\|Z \arrow[l, bend left, "dec"] \arrow[loop above, "zero"] \arrow[r, bend left, "inc"] & \cdots \arrow[l, bend left, "dec"]
\end{tikzcd}$$

The counting property of the counter remains the same as for BPA since the actions '*inc*' and '*dec*' correctly add resp. remove the process constant C. However, the test for zero can be performed anytime, irrelevant whether the counter is empty or not. We can see that we are missing a test for zero in our example, and it is a well known fact that it is impossible to model this test even using Petri nets (a superclass of BPP). Hence we can use only counters without the explicit test for emptiness. $\square$

The following example aims to demonstrate that the operator '$\|$' in BPP systems allows a parallel access to all process constants contained in the current state.

Example 2.12. Let $\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{Q_1, Q_2, \ldots, Q_k\}$ for a natural number $k > 0$ and let $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{q_1, q_2, \ldots, q_k\}$. The set of rewrite rules Δ is defined by:

$$Q_j \xrightarrow{q_j} Q_j \qquad \text{for all } j,\ 1 \leq j \leq k.$$

Assume now that $m_1, m_2 > 0$ are natural numbers. For every ℓ_1, $1 \leq \ell_1 \leq m_1$, and every ℓ_2, $1 \leq \ell_2 \leq m_2$, let $i_{\ell_1} \in \{1, 2, \ldots, k\}$ and $j_{\ell_2} \in \{1, 2, \ldots, k\}$. We conclude that

$$(Q_{i_1} \| Q_{i_2} \| \cdots \| Q_{i_{m_1}}, \Delta) \sim (Q_{j_1} \| Q_{j_2} \| \cdots \| Q_{j_{m_2}}, \Delta)$$

if and only if

$$\{i_1, i_2, \ldots, i_{m_1}\} = \{j_1, j_2, \ldots, j_{m_2}\}.$$

In other words, the processes are strongly bisimilar if and only if for all j, $1 \leq j \leq k$, the process constant Q_j appears either in both sides of the processes or in neither of them. In the first case the number of occurrences of Q_j is irrelevant since $(Q_j^m, \Delta) \sim (Q_j, \Delta)$ for any natural number $m > 0$ — see Notation 2.4. □

2.5.4 Pushdown Processes

Pushdown processes (PDA) — or equivalently $(\mathcal{S}, \mathcal{S})$-PRS — represent the class of processes introduced via sequential prefix rewriting with unrestricted rules.

Let Δ be an $(\mathcal{S}, \mathcal{S})$-PRS. Every rewrite rule from Δ is of the form

$$E \xrightarrow{a} F$$

where $E, F \in \mathcal{S}(\mathcal{C}onst(\Delta))$ and $a \in \mathcal{A}ct(\Delta)$. This seems to be more expressive than the standard definition of PDA where we assume that $\mathcal{C}onst(\Delta)$ is a disjoint union of *control states* Q and a *stack alphabet* Γ, and $\Delta \subseteq Q \times \Gamma \times \mathcal{A}ct \times Q \times \Gamma^*$. Usually, an element $(p, X, a, q, \alpha) \in \Delta$ is written as $pX \xrightarrow{a} q\alpha$, meaning that from a configuration in a control state p where the top of the stack contains X we can execute the action a such that the control state is changed to q and the top of the stack is replaced with (possibly empty) sequence α. States of such a PDA system Δ are then of the form $p.\alpha$ where $p \in Q$ and $\alpha \in \Gamma^*$.

Notation 2.5. Instead of denoting a state as e.g. $p.X.Y.X$ we usually omit the operator for sequential composition and write simply $pXYX$. A state $p\epsilon \in Q \times \{\epsilon\}$, where '$\epsilon$' is the symbol for the *empty stack*, is usually written only as p.

Obviously, any PDA system Δ as introduced above is by definition also an $(\mathcal{S}, \mathcal{S})$-PRS. Moreover, Caucal [40] showed that an arbitrary unrestricted $(\mathcal{S}, \mathcal{S})$-PRS can be transformed into a PDA system such that the generated transition systems are isomorphic up to the names of states. Hence $(\mathcal{S}, \mathcal{S})$-PRS and PDA are equivalent formalisms.

Remark 2.5. To be more precise about the transformation from $(\mathcal{S}, \mathcal{S})$-PRS to PDA we must remark that these classes might differ when complexity issues are involved. The transformation presented in [40] in general requires an exponential blow up in the size of Δ. Hence whenever we provide some complexity bounds for PDA we always assume that the size of the system is given with regard to the traditional definition of PDA.

We have to provide also some comments about the notion of normedness for PDA. According to Definition 2.14 a process is normed iff from every reachable state the empty process ϵ can be reached. Since we, however, often assume that a PDA system Δ is given by rewrite rules of the form $pX \xrightarrow{a} q\alpha$, we have to modify the notion of normedness slightly (otherwise all processes become unnormed with regard to Definition 2.14).

Definition 2.16 (Normed PDA process).
We say that a PDA process $(p\alpha, \Delta)$ is *normed* iff for every reachable state $q\beta$ in $(p\alpha, \Delta)$ there is a computation which empties the stack, i.e., there is a state $r\epsilon \in Q \times \{\epsilon\}$ such that $q\beta \longrightarrow^* r\epsilon$.

Remark 2.6. Another, and for most of the purposes equivalent way to introduce normedness for PDA is to assume that the stack contains a distinguished bottom symbol Z. Let z be a fresh action. After adding the rules $pZ \xrightarrow{z} \epsilon$ for all $p \in Q$, we can use the notion of normedness from Definition 2.14. In this thesis we will rely on Definition 2.16.

Notation 2.6. Let i be a natural number and $A \in \Gamma$. We use the notation A^i for a sequence of i occurrences of A, i.e., $A^0 \stackrel{\text{def}}{=} \epsilon$ and $A^{i+1} \stackrel{\text{def}}{=} A^i A$. For example pX^2Y^3 is an abbreviation for $pXXYYY$.

Example 2.13. Let us consider a PDA system Δ where the sets of control states Q and stack alphabet Γ are given by $Q \stackrel{\text{def}}{=} \{p, q, r\}$ and $\Gamma \stackrel{\text{def}}{=} \{X\}$, respectively, the set of actions is given by $\mathcal{A}ct(\Delta) \stackrel{\text{def}}{=} \{a, b, c, d\}$, and Δ is defined as follows.

$$pX \xrightarrow{a} pXX \qquad pX \xrightarrow{b} q \qquad pX \xrightarrow{b} r \qquad qX \xrightarrow{c} q \qquad rX \xrightarrow{d} r$$

The following fragment of a transition system is generated by the process (pX, Δ).

$$\begin{tikzcd}
q & qX \arrow[l, "c"'] & qX^2 \arrow[l, "c"'] & \cdots \arrow[l, "c"'] \\
pX \arrow[u, "b"'] \arrow[d, "b"] \arrow[r, "a"] & pX^2 \arrow[u, "b"'] \arrow[d, "b"] \arrow[r, "a"] & pX^3 \arrow[u, "b"'] \arrow[d, "b"] \arrow[r, "a"] & \cdots \\
r & rX \arrow[l, "d"'] & rX^2 \arrow[l, "d"'] & \cdots \arrow[l, "d"']
\end{tikzcd}$$

As argued in [137] there is no BPA or BPP system which is strongly bisimilar to (pX, Δ). It is also an easy observation that the process (pX, Δ) is normed, i.e., from every reachable state it is possible to reach a state with the stack being empty ($q\epsilon$ or $r\epsilon$). $\Box$

Example 2.14. Let $Q \stackrel{\text{def}}{=} \{p, p_1, p', p'_1, p'_2, p'_3\}$, $\Gamma \stackrel{\text{def}}{=} \{X, Y\}$, $\mathcal{A}ct \stackrel{\text{def}}{=} \{a, b, c, \tau\}$, and let Δ be the following pushdown process.

$$\begin{array}{lll}
pX \xrightarrow{a} p_1X & pX \xrightarrow{\tau} p'X & p_1X \xrightarrow{c} p_1X \\
 & & \\
p'X \xrightarrow{a} p'_1X & p'_1X \xrightarrow{c} p'_1X & \\
p'X \xrightarrow{\tau} p'_2X & p'_2X \xrightarrow{c} p'_3 & \\
p'_2X \xrightarrow{\tau} p'_2YX & p'_2Y \xrightarrow{\tau} p'_2YY & p'_2Y \xrightarrow{b} p'_2
\end{array}$$

A fraction of the transition system $T(\Delta)$ is depicted in the following picture.

$$\begin{tikzcd}
 & pX \arrow[dl, "a"'] \arrow[d, "\tau"] & & & \\
p_1X \arrow["c"', loop, distance=2em, in=235, out=305] & p'X \arrow[dl, "a"'] \arrow[d, "\tau"] & & & \\
p'_1X \arrow["c"', loop, distance=2em, in=235, out=305] & p'_2X \arrow[d, "c"] \arrow[r, bend left, "\tau"] & p'_2YX \arrow[l, bend left, "b"] \arrow[r, bend left, "\tau"] & p'_2YYX \arrow[l, bend left, "b"] \arrow[r, bend left, "\tau"] & \cdots \arrow[l, bend left, "b"] \\
 & p'_3 & & &
\end{tikzcd}$$

Let us consider processes (pX, Δ) and $(p'X, \Delta)$. We show that $(pX, \Delta) \approx (p'X, \Delta)$ by describing a winning strategy for the defender in the bisimulation game starting from pX and $p'X$. The attacker has the following four possibilities in the first round:

1. $pX \xrightarrow{a} p_1X$, or
2. $pX \xrightarrow{\tau} p'X$, or
3. $p'X \xrightarrow{a} p'_1X$, or
4. $p'X \xrightarrow{\tau} p'_2X$.

We shall explain defender's answers to these moves now.

1. The defender answers by playing $p'X \overset{a}{\Longrightarrow} p'_1X$. Now the game continues from p_1X and p'_1X, however, these two states are obviously weakly bisimilar and hence the defender has a winning strategy in this case.

2. The defender answers by doing nothing ($p'X \overset{\tau}{\Longrightarrow} p'X$). We remind the reader of the fact that this is a legitimate defender's move. Now the players reached the pair $p'X$ and $p'X$. Since these two states are syntactically the same, there is an obvious winning strategy for the defender.

3. The defender answers by playing $pX \overset{a}{\Longrightarrow} p_1X$ or $pX \overset{a}{\Longrightarrow} p'_1X$. Either of these answers is good for the defender.

4. The defender answers by $pX \overset{\tau}{\Longrightarrow} p'_2X$ and this is a winning move for the defender because the game continues from syntactically equal states.

Hence $(pX, \Delta) \approx (p'X, \Delta)$ using Proposition 2.2. In this example we also recall some of our previous definitions. The process (pX, Δ) can terminate (empty its stack) since $pX \longrightarrow^* p'_3$ but it is not normed — it can reach e.g. the state p_1X from which there is no terminating computation. On the other hand the process (p'_2Y^iX, Δ) is normed for any $i \geq 0$.

Also note that (pX, Δ) is not weakly (nor strongly) regular because it has infinitely many reachable and weakly (strongly) nonbisimilar states. Consider the states p'_2Y^iX and p'_2Y^jX for $i \neq j$. Obviously, $pX \longrightarrow^* p'_2Y^iX$ and $pX \longrightarrow^* p'_2Y^jX$. We leave it to the reader to find a winning strategy for the attacker in the weak (strong) bisimulation game from the pair p'_2Y^iX and p'_2Y^jX and thus show that $(p'_2Y^iX, \Delta) \not\approx (p'_2Y^jX, \Delta)$ and $(p'_2Y^iX, \Delta) \not\sim (p'_2Y^jX, \Delta)$. □

2.5.5 PA–Processes

PA-processes (PA for process algebra) — or equivalently $(1, \mathcal{G})$-PRS — represent the class of processes originally introduced by Baeten and Weijland [15]. This formalism combines the parallel and sequential operator but allows neither communication nor global-state control.

Nowadays the word "process algebra" has much broader meaning than just the specific algebra by Baeten and Weijland and we tend to call their process algebra *PA-processes* in order to avoid confusion.

Let Δ be a $(1, \mathcal{G})$-PRS. Every rewrite rule from Δ is of the form

$$X \xrightarrow{a} E$$

where $X \in \mathcal{C}onst(\Delta)$, $a \in \mathcal{A}ct(\Delta)$ and $E \in \mathcal{G}(\mathcal{C}onst(\Delta))$.

Example 2.15. Let us consider a PA system Δ

$$\begin{array}{lll} C_i \xrightarrow{inc_i} C_i.C_i & C_i \xrightarrow{dec_i} \epsilon & \text{for } 1 \leq i \leq 2 \\ Z_i \xrightarrow{inc_i} C_i.Z_i & Z_i \xrightarrow{zero_i} Z_i & \text{for } 1 \leq i \leq 2 \\ & & \\ X \xrightarrow{a} Z_1 \| Z_2 & & \end{array}$$

where the set of process constants is $\mathcal{C}onst(\Delta) \stackrel{\text{def}}{=} \{C_1, C_2, Z_1, Z_2, X\}$ and the set of actions is $\mathcal{A}ct(\Delta) \stackrel{\text{def}}{=} \{inc_1, inc_2, dec_1, dec_2, zero_1, zero_2, a\}$. The process (X, Δ) introduces two counters (similarly as in Example 2.9) which are composed together via the parallel operator. A fragment of the transition system generated by $(Z_1 \| Z_2, \Delta)$ follows.

$$\begin{tikzcd}
Z_1\|Z_2 \arrow[loop above, "zero_2"] \arrow[loop left, "zero_1"] \arrow[r, shift left, "inc_1"] \arrow[d, shift right, "inc_2"'] & C_1.Z_1\|Z_2 \arrow[loop above, "zero_2"] \arrow[l, shift left, "dec_1"] \arrow[r, shift left, "inc_1"] \arrow[d, shift right, "inc_2"'] & C_1^2.Z_1\|Z_2 \arrow[loop above, "zero_2"] \arrow[l, shift left, "dec_1"] \arrow[r, shift left, "inc_1"] \arrow[d, shift right, "inc_2"'] & \cdots \arrow[l, shift left, "dec_1"] \\
Z_1\|C_2.Z_2 \arrow[loop left, "zero_1"] \arrow[u, shift right, "dec_2"'] \arrow[r, shift left, "inc_1"] \arrow[d, shift right, "inc_2"'] & C_1.Z_1\|C_2.Z_2 \arrow[u, shift right, "dec_2"'] \arrow[l, shift left, "dec_1"] \arrow[r, shift left, "inc_1"] \arrow[d, shift right, "inc_2"'] & C_1^2.Z_1\|C_2.Z_2 \arrow[u, shift right, "dec_2"'] \arrow[l, shift left, "dec_1"] \arrow[r, shift left, "inc_1"] \arrow[d, shift right, "inc_2"'] & \cdots \arrow[l, shift left, "dec_1"] \\
Z_1\|C_2^2.Z_2 \arrow[loop left, "zero_1"] \arrow[u, shift right, "dec_2"'] \arrow[r, shift left, "inc_1"] \arrow[d, shift right, "inc_2"'] & C_1.Z_1\|C_2^2.Z_2 \arrow[u, shift right, "dec_2"'] \arrow[l, shift left, "dec_1"] \arrow[r, shift left, "inc_1"] \arrow[d, shift right, "inc_2"'] & C_1^2.Z_1\|C_2^2.Z_2 \arrow[u, shift right, "dec_2"'] \arrow[l, shift left, "dec_1"] \arrow[r, shift left, "inc_1"] \arrow[d, shift right, "inc_2"'] & \cdots \arrow[l, shift left, "dec_1"] \\
\vdots \arrow[u, shift right, "dec_2"'] & \vdots \arrow[u, shift right, "dec_2"'] & \vdots \arrow[u, shift right, "dec_2"'] &
\end{tikzcd}$$

This means that the two counters can behave independently and e.g. the following transition sequence is possible.

$$X \xrightarrow{a} Z_1\|Z_2 \xrightarrow{inc_1} C_1.Z_1 \parallel Z_2 \xrightarrow{zero_2} C_1.Z_1 \parallel Z_2 \xrightarrow{inc_1} C_1^2.Z_1 \parallel Z_2 \longrightarrow \ldots$$

□

2.5.6 Petri Nets

Petri nets (PN) — or equivalently $(\mathcal{P},\mathcal{P})$-PRS — represent the class of processes which correspond to the standard notion of labelled place/transition (P/T) nets as originally proposed by Petri [149] .

Let Δ be a $(\mathcal{P},\mathcal{P})$-PRS. Every rewrite rule from Δ is of the form

$$E \xrightarrow{a} G$$

where $E, G \in \mathcal{P}(\mathcal{C}onst(\Delta))$ and $a \in \mathcal{A}ct(\Delta)$.

In order to see the connection between $(\mathcal{P},\mathcal{P})$-PRS and labelled Petri net let us first introduce P/T nets. A *P/T net* is a triple $N = (P, T, F)$ where

- P is a finite set of *places*,
- T is a finite set of *transitions* such that $T \cap P = \emptyset$, and
- $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation.*

We define ${}^\bullet t \overset{\text{def}}{=} \{p \mid (p,t) \in F\}$ and $t^\bullet \overset{\text{def}}{=} \{p \mid (t,p) \in F\}$ for a transition $t \in T$, and ${}^\bullet p \overset{\text{def}}{=} \{t \mid (t,p) \in F\}$ and $p^\bullet \overset{\text{def}}{=} \{t \mid (p,t) \in F\}$ for a place $p \in P$.

A *marking* M of a P/T net $N = (P, T, F)$ is a mapping $M : P \to \mathbb{N}_0$, i.e., each place is assigned a nonnegative number of *tokens*. We say that $t \in T$ is *enabled* in a marking M iff $\forall p \in {}^\bullet t.\ M(p) > 0$. If t is enabled in M then it can be *fired*, producing a marking M' (written $M[t\rangle M'$) such that:

- $M'(p) \overset{\text{def}}{=} M(p)$ for all $p \in \big(P \smallsetminus ({}^\bullet t \cup t^\bullet)\big) \cup ({}^\bullet t \cap t^\bullet)$
- $M'(p) \overset{\text{def}}{=} M(p) - 1$ for all $p \in {}^\bullet t \smallsetminus t^\bullet$
- $M'(p) \overset{\text{def}}{=} M(p) + 1$ for all $p \in t^\bullet \smallsetminus {}^\bullet t$.

A *labelled Petri net* is a tuple $N = (P, T, F, \mathcal{A}ct, \lambda)$ where (P, T, F) is a P/T net, $\mathcal{A}ct$ is a set of *labels* (actions) and $\lambda : T \to \mathcal{A}ct$ is a *labelling function.*

Remark 2.7. Without loss of generality we assume that if $M[t_1\rangle M'$ and $M[t_2\rangle M'$ then $\lambda(t_1) \neq \lambda(t_2)$ for any pair of markings M, M' and transitions t_1 and t_2.

Definition 2.17 (Labelled transition system $T(N)$).
Let $N = (P, T, F, \mathcal{A}ct, \lambda)$ be a labelled Petri net. We define a corresponding labelled transition system $T(N)$ as $T(N) \overset{\text{def}}{=} ([P \to \mathbb{N}_0], \mathcal{A}ct, \longrightarrow)$ where $M \xrightarrow{a} M'$ whenever $M[t\rangle M'$ and $a = \lambda(t)$ for $M, M' \in [P \to \mathbb{N}_0]$ and $t \in T$.

Every rewrite rule $E \xrightarrow{a} G$ in a $(\mathcal{P}, \mathcal{P})$-PRS Δ corresponds to a transition $t_{E \xrightarrow{a} G}$ in a labelled Petri net $\big(\mathcal{C}onst(\Delta), \{t_{E \xrightarrow{a} G} \mid (E, a, G) \in \Delta\}, F, \mathcal{A}ct(\Delta), \lambda\big)$ such that the number of occurrences of a process constant $X \in \mathcal{C}onst(\Delta)$ in E (in G) corresponds to number of tokens consumed (produced) by $t_{E \xrightarrow{a} G}$ from (to) the place X, and $\lambda(t_{E \xrightarrow{a} G}) \overset{\text{def}}{=} a$. Similarly any labelled Petri net corresponds to a certain $(\mathcal{P}, \mathcal{P})$-PRS.

Remark 2.8. This transformation may in general introduce a slightly more general class of labelled Petri nets with multiple arcs (see e.g. [148]) since the process constant X in E and G can have multiple occurrences. Nevertheless, all the constructions presented in this thesis rely only on the model of P/T nets without multiple arcs.

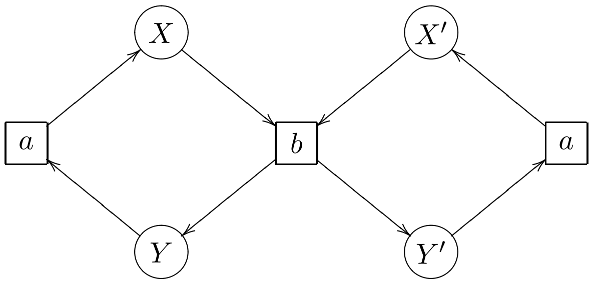
Example 2.16. Let us consider the following system

$$\Delta \overset{\text{def}}{=} \{Y \xrightarrow{a} X,\ X \| X' \xrightarrow{b} Y \| Y',\ Y' \xrightarrow{a} X'\}$$

where $\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{X, X', Y, Y'\}$ and $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{a, b\}$. The system Δ corresponds to a labelled Petri net

$$N \overset{\text{def}}{=} \big(\{X, X', Y, Y'\},\ \{t_{Y \xrightarrow{a} Y'},\ t_{X \| X' \xrightarrow{b} Y \| Y'},\ t_{Y' \xrightarrow{a} X'}\}, F, \mathcal{A}ct(\Delta), \lambda\big)$$

where $\lambda(t_{Y \xrightarrow{a} Y'}) \overset{\text{def}}{=} a$, $\lambda(t_{X \| X' \xrightarrow{b} Y \| Y'}) \overset{\text{def}}{=} b$ and $\lambda(t_{Y' \xrightarrow{a} X'}) \overset{\text{def}}{=} a$. The flow relation F is given by the standard graphical notation.



□

2.5.7 PAD, PAN and PRS Processes

The systems PAD, PAN and PRS correspond to $(\mathcal{S},\mathcal{G})$-PRS, $(\mathcal{P},\mathcal{G})$-PRS and $(\mathcal{G},\mathcal{G})$-PRS, respectively. These classes complete the hierarchy of process rewrite systems and were introduced by Mayr [126].

The PAD class is the smallest common generalization of PA and PDA [123] and PAN is a combination of the models PA and PN [121]. The most general class PRS was introduced in [122] and it subsumes all the previously mentioned classes. Nevertheless, even the PRS model preserves some interesting properties with regard to automatic verification techniques: it is not Turing powerful and some problems such as reachability remain decidable [126].

2.6 Bibliographical Remarks

In this chapter we introduced the basic notation and definitions used in the thesis. We aimed to define separately the notion of labelled transition systems (together with the notion of a process, strong/weak bisimilarity and regularity) and the notion of process rewrite systems (PRS) as a finite description for certain classes of infinite-state transition systems. The choice of PRS is justified by several arguments. First of all we think that PRS as introduced in [126] provide elegant and succinct process definitions. The key features are uniformity of the description and sufficient generality. This in particular means that all the infinite-state systems studied in this thesis can be defined as a subclass of PRS and a uniform treatment of these formalisms is guaranteed.

Moreover, since the strongest notion of behavioural equivalence considered for the systems is strong bisimilarity, the PRS classes correspond naturally to the well studied classes from other hierarchies introduced before — see [40, 137, 34]. Taking into account the earlier definitions of BPA and BPP using CCS like syntax, our definitions of these classes using process rewrite systems correspond more or less to the original definitions of BPA and BPP given in Greibach normal form (GNF). It is a well known fact that BPA and BPP systems can be described (up to strong bisimilarity) in GNF (see [81, 13, 10, 77, 42]). This gives a certain regime for using action prefixing and nondeterministic choice, while it still preserves the descriptive power of the systems. Hence the classes of original process algebras and the classes introduced in the PRS-hierarchy are equivalent up to strong bisimilarity.

Another issue this thesis heavily relies on is the game-theoretic characterization of strong and weak bisimilarity. Probably the best sources for these kinds of considerations are [190] and [183]. Bisimulation checking is reduced to finding winning strategies for the players in the bisimulation game. This in general provides more transparent description of proofs and of the ideas behind them, and moreover it supports a clear intuitive understanding of the nature of bisimilarity.

Chapter 3

Unlabelled Transition Systems

In this chapter we discuss the role of labels in transition systems. We suggest a general reduction from labelled transition systems to unlabelled ones, preserving strong (and hence also weak) bisimilarity. We apply the reduction to the class of transition systems generated by Petri nets and pushdown processes, and obtain several decidability/complexity corollaries for unlabelled systems. Probably the most interesting result is undecidability of strong bisimilarity for unlabelled Petri nets.

3.1 Motivation

As shown in Section 2.1, it is possible to define bisimilarity checking problems irrespective of the concrete syntax of the considered process algebra. The mathematical structure of labelled transition systems is sufficient to provide definitions of strong and weak bisimilarity. Hence it appears essential to study in more detail this notion of transition systems as it provides semantics to e.g. all the formalisms considered in this thesis.

In this chapter we focus in particular on the role of labels. There are two aspects of the branching structure described by a labelled transition system. First, given a state of a labelled transition system, there can be several outgoing edges with different labels — Figure 3.1 a). Second, given a state and a label, there can be several outgoing edges under the label — Figure 3.1 b). We claim that for our purposes only the second property is the essential one.

In other words, given a labelled transition system, we can construct another transition system where all edges are labelled by the same label, i.e., where the labels become irrelevant. We call such systems *unlabelled transition systems*. What is important is the fact that our construction preserves the answer to the question of strong (and weak) bisimilarity.

In order to demonstrate usefulness of the transformation we will present two applications. First for the class of Petri nets and then also for pushdown processes. Petri nets are a typical example of a fully parallel model of computation, whereas pushdown processes can model sequential stack-like behaviours. Both Petri nets and pushdown systems generate (in general infinite) labelled transition systems. The question is whether the transformed unlabelled transi-

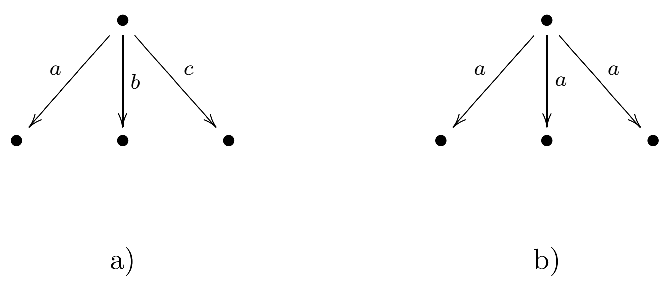


Figure 3.1: Two aspects of branching in labelled transition systems

tion systems (given by the construction mentioned in the previous paragraph) are still definable by the formalism of Petri nets resp. pushdown systems. The answer is shown to be positive for both our models — there are even polynomial time transformations. This implies several decidability/complexity results for bisimilarity checking of unlabelled Petri nets and pushdown processes.

Probably the most interesting corollary is the application of the transformation to Petri nets. We can conclude that strong bisimilarity for unlabelled Petri nets (where the set of labels is a singleton set) is undecidable. This is a stronger result than undecidability of strong bisimilarity for labelled Petri nets given by Jančar [90] where labelling of arcs is essential. The undecidability theorem for unlabelled Petri nets contrasts to a positive decidability result of strong bisimilarity for a subclass of Petri nets which are deterministic — for any marking M and a label a there is at most one outgoing transition from M labelled by a. In fact there is even a stronger result showing that strong bisimilarity is decidable for Petri nets which are deterministic up to bisimilarity [90], i.e., $\mathcal{F}$-deterministic nets of Vogler [201]. This again demonstrates that the role of labels is not important for decidability questions and what is crucial is the branching structure induced by transitions with the same label.

In the end of this chapter we briefly discuss possible applications of the transformation to BPA and BPP. Unfortunately, it is sketched that these models — unlike Petri nets and pushdown systems — are not strong enough to express deadlock behaviour which is essential for the transformation. In other words, the corresponding unlabelled transition system of a given BPA (BPP) transition graph is not necessarily definable in the BPA (BPP) syntax. Nevertheless, we still think that at least for the case of BPA our approach might bring some merits e.g. in connection with improving 2-EXPTIME [35] complexity upper bound of strong bisimilarity checking.

3.2 From Labelled to Unlabelled Transition Systems

In this section we present a transformation from labelled transition systems to unlabelled ones while preserving strong bisimilarity.

Let us first define the notion of unlabelled transition systems.

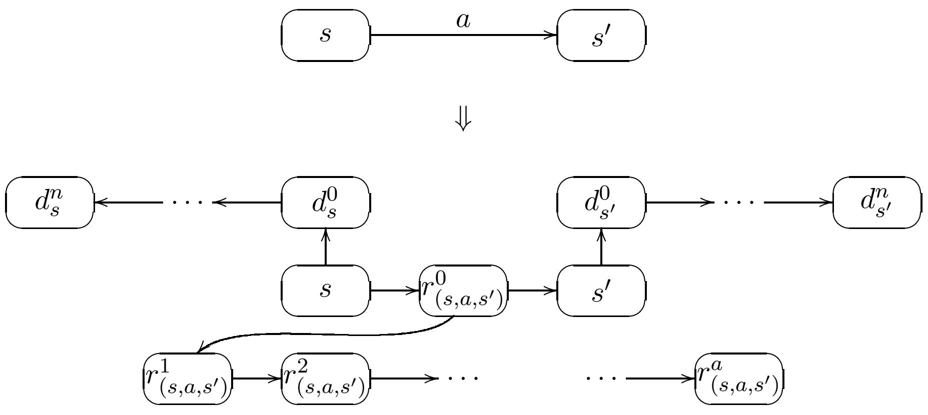


Figure 3.2: Transformation of a transition $s \xrightarrow{a} s'$

Definition 3.1 (Unlabelled transition system).
Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system. We call T *unlabelled transition system* whenever $\mathcal{A}ct$ is a singleton set, i.e., $|\mathcal{A}ct| = 1$.

Remark 3.1. If it is the case that $|\mathcal{A}ct| = 1$ then we simply write $\longrightarrow$ instead of $\xrightarrow{a}$. We also omit the second component in the definition of a labelled transition system, i.e., we can denote an unlabelled transition system simply by $T = (S, \longrightarrow)$ where $\longrightarrow \subseteq S \times S$.

Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system. We define a transformed *unlabelled* transition system $\widehat{T} \stackrel{\text{def}}{=} (\widehat{S}, \longrightarrow)$. We reuse the relation symbol $\longrightarrow$ without causing confusion since in the system T it is a ternary relation and in $\widehat{T}$ it is a binary relation. Without loss of generality we can assume that $\mathcal{A}ct = \{1, 2, \ldots, n\}$ for some $n > 0$. We define the system $\widehat{T} = (\widehat{S}, \longrightarrow)$ as follows:

$$\begin{aligned}
\widehat{S} \quad &\stackrel{\text{def}}{=} \quad S \cup \{r^k_{(s,a,s')} \mid 0 \leq k \leq a \ \wedge \ s \xrightarrow{a} s'\} \ \cup \\
&\qquad \{d^k_s \mid s \in S \ \wedge \ 0 \leq k \leq n\} \\
\longrightarrow \quad &\stackrel{\text{def}}{=} \quad \{(s, r^0_{(s,a,s')}),\ (r^0_{(s,a,s')}, s') \mid s \xrightarrow{a} s'\} \ \cup \\
&\qquad \{(r^k_{(s,a,s')}, r^{k+1}_{(s,a,s')}) \mid s \xrightarrow{a} s' \ \wedge \ 0 \leq k < a\} \ \cup \\
&\qquad \{(s, d^0_s) \mid s \in S\} \ \cup \\
&\qquad \{(d^k_s, d^{k+1}_s) \mid s \in S \ \wedge \ 0 \leq k < n\}.
\end{aligned}$$

For a better understanding of the transformation take a look at Figure 3.2 where a way how to transform a transition $s \xrightarrow{a} s'$ is drawn. The idea consists in splitting each transition $s \xrightarrow{a} s'$ labelled by $a \in \mathbb{N}_0$ with an intermediate state (the $r^0_{(s,a,s')}$ state) out of which goes a newly added linear path of length a. The d_s states add a linear path of length $n + 1$ to each state from S and serve for distinguishing the r-states from the original ones.

Remark 3.2. Notice that if T is a finite-state system then the size of $\widehat{T}$ is polynomially bounded by the size of T. In fact, we could add only one linear path of length $n+1$ with appropriate links into the path from the states in S and the r^0-states. However, for technical convenience in Section 3.3, we use the previously described construction.

Our aim is to show that for a pair of states s_1 and s_2 from the labelled transition system T it holds that $(s_1, T) \sim (s_2, T)$ if and only if $(s_1, \widehat{T}) \sim (s_2, \widehat{T})$.

Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system and let $s \in S$. We define a set of *finite norms* of s by

$$\mathcal{N}(s) \overset{\text{def}}{=} \{|w| \mid \exists s' \in S : s \xrightarrow{w} s' \not\longrightarrow\}$$

where $|w|$ is the length of w.

The following proposition is a standard one.

Proposition 3.1. Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system and let $s_1, s_2 \in S$. Then $(s_1, T) \sim (s_2, T)$ implies that $\mathcal{N}(s_1) = \mathcal{N}(s_2)$.

Lemma 3.1. Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system and $s_1, s_2 \in S$ be a pair of states. If $(s_1, T) \sim (s_2, T)$ then $(s_1, \widehat{T}) \sim (s_2, \widehat{T})$.

Proof. Suppose that the defender has a winning strategy in T starting from the pair s_1 and s_2. We show that the defender in $\widehat{T}$ has also a winning strategy starting from the pair s_1 and s_2. Any attacker's move in $\widehat{T}$ of the form $s_i \longrightarrow d^0_{s_i}$ (for $i \in \{1,2\}$) can be matched by the defender's move $s_{3-i} \longrightarrow d^0_{s_{3-i}}$. The states $d^0_{s_1}$ and $d^0_{s_2}$ are trivially bisimilar. Let the attacker's move in $\widehat{T}$ be $s_i \longrightarrow r^0_{(s_i,a,s'_i)}$ for some $i \in \{1,2\}$. Of course, the following attacker's move is possible in T as well: $s_i \xrightarrow{a} s'_i$. Let the defender's answer to this move in T be $s_{3-i} \xrightarrow{a} s'_{3-i}$ such that

$$(s'_1, T) \sim (s'_2, T). \tag{3.1}$$

The defender's response in $\widehat{T}$ is then $s_{3-i} \longrightarrow r^0_{(s_{3-i},a,s'_{3-i})}$. Now the game in $\widehat{T}$ continues from the states $r^0_{(s_1,a,s'_1)}$ and $r^0_{(s_2,a,s'_2)}$. Given an $i \in \{1,2\}$, only two transitions are possible from $r^0_{(s_i,a,s'_i)}$. Either the attacker can choose $r^0_{(s_i,a,s'_i)} \longrightarrow s'_i$ or $r^0_{(s_i,a,s'_i)} \longrightarrow r^1_{(s_i,a,s'_i)}$. If he chooses the second option then he looses since the defender answers by $r^0_{(s_{3-i},a,s'_{3-i})} \longrightarrow r^1_{(s_{3-i},a,s'_{3-i})}$ and the states $r^1_{(s_1,a,s'_1)}$ and $r^1_{(s_2,a,s'_2)}$ are easily seen to be strongly bisimilar. Should the attacker's choice be $r^0_{(s_i,a,s'_i)} \longrightarrow s'_i$ then the defender's answer is $r^0_{(s_{3-i},a,s'_{3-i})} \longrightarrow s'_{3-i}$. Since $s'_1, s'_2 \in S$ and the defender in T has a winning strategy from these states because of (3.1), we have established a winning strategy for the defender in $\widehat{T}$. □

Before showing the other implication, we prove the following property.

Property 3.1. The attacker in $\widehat{T}$ has a winning strategy from any pair of states $s_1, s_2 \in \widehat{S}$ such that $s_1 \notin S$ and $s_2 \in S$, or $s_1 \in S$ and $s_2 \notin S$.

Proof. Assume w.l.o.g. that $s_1 \notin S$ and $s_2 \in S$. The other case is symmetric. There are several possibilities for $s_1 \notin S$.

- Let $s_1 = d_s^k$ for some $s \in S$ and $0 \leq k \leq n$, or $s_1 = r^k_{(s,a,s')}$ for some $s, s' \in S$, $a \in \mathcal{A}ct$ and $0 < k \leq a$. In both cases $n + 1 \notin \mathcal{N}(s_1)$ and $n + 1 \in \mathcal{N}(s_2)$. Because of Proposition 3.1 we get $(s_1, \widehat{T}) \not\sim (s_2, \widehat{T})$ and the attacker in $\widehat{T}$ has a winning strategy.

- Let $s_1 = r^0_{(s,a,s')}$ for some $s, s' \in S$ and $a \in \mathcal{A}ct$. Now the attacker has the following winning strategy in $\widehat{T}$. He makes the move $r^0_{(s,a,s')} \longrightarrow r^1_{(s,a,s')}$. Assume a defender's answer $s_2 \longrightarrow s'_2$ for an arbitrary $s'_2 \in \widehat{S}$. Obviously either $n \in \mathcal{N}(s'_2)$ or $n+2 \in \mathcal{N}(s'_2)$. On the other hand $\max[\mathcal{N}(r^1_{(s,a,s')})] < n$. Again, using Proposition 3.1, the attacker has a winning strategy.

□

Lemma 3.2. Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system and $s_1, s_2 \in S$ be a pair of states. If $(s_1, \widehat{T}) \sim (s_2, \widehat{T})$ then $(s_1, T) \sim (s_2, T)$.

Proof. Knowing that the defender has a winning strategy in $\widehat{T}$ from s_1 and s_2, we establish a winning strategy for the defender in T from s_1 and s_2. Suppose that the attacker's move in T is $s_i \xrightarrow{a} s'_i$ for some $i \in \{1, 2\}$. Then it is possible to perform a series of two moves $s_i \longrightarrow r^0_{(s_i,a,s'_i)} \longrightarrow s'_i$ in $\widehat{T}$. Because of Property 3.1, the defender in $\widehat{T}$ has a response to these two moves only by performing $s_{3-i} \longrightarrow r^0_{(s_{3-i},b,s'_{3-i})} \longrightarrow s'_{3-i}$ for some $b \in \mathcal{A}ct$ and $s'_{3-i} \in S$ where

$$(s'_1, \widehat{T}) \sim (s'_2, \widehat{T}). \tag{3.2}$$

Notice that $a = b$, otherwise the attacker has a winning strategy in $\widehat{T}$ from $r^0_{(s_i,a,s'_i)}$ and $r^0_{(s_{3-i},b,s'_{3-i})}$ by performing the move $r^0_{(s_i,a,s'_i)} \longrightarrow r^1_{(s_i,a,s'_i)}$. Using Property 3.1, the defender must answer by $r^0_{(s_{3-i},b,s'_{3-i})} \longrightarrow r^1_{(s_{3-i},b,s'_{3-i})}$. However, the attacker has a winning strategy now since $a - 1 \in \mathcal{N}(r^1_{(s_i,a,s'_i)})$ and $a - 1 \notin \mathcal{N}(r^1_{(s_{3-i},b,s'_{3-i})})$ whenever $a \neq b$ — Proposition 3.1. This implies that the defender in T can perform $s_{3-i} \xrightarrow{a} s'_{3-i}$ and because of (3.2), the defender in T has a winning strategy from s'_1 and s'_2. Hence $(s_1, T) \sim (s_2, T)$. □

We can now state the main theorem of this section.

Theorem 3.1. Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a labelled transition system and let $s_1, s_2 \in S$ be a pair of states. Let $\widehat{T}$ be the corresponding unlabelled transition system. Then

$$(s_1, T) \sim (s_2, T) \quad \text{if and only if} \quad (s_1, \widehat{T}) \sim (s_2, \widehat{T}).$$

Proof. From Lemma 3.1 and Lemma 3.2. □

3.3 Applications

In this section we show that Theorem 3.1 successfully applies to bisimilarity checking of many infinite-state systems. We have to prove that the class of transition systems generated by a given model is closed under the transformation from labelled to unlabelled systems as presented in the previous section. We will focus in particular on a typical representative of parallel models — Petri nets — and sequential processes — pushdown systems.

First of all, note the fact that our transformation works immediately for finite-state processes.

Corollary 3.1. Let $T = (S, \mathcal{A}ct, \longrightarrow)$ be a finite-state labelled transition system, i.e., $|S|, |\mathcal{A}ct| < \infty$. There is a polynomial time reduction from strong bisimilarity checking problem for T to strong bisimilarity checking problem for $\widehat{T}$, where $\widehat{T}$ is an unlabelled (and finite-state) transition system.

Proof. Immediately from Theorem 3.1. $\square$

Remark 3.3. We remind the reader of the fact that the transformation from labelled to unlabelled transition systems does not only preserve the answer to strong bisimilarity. It moreover preserves the property of being strongly regular: a process in a labelled transition system is strongly regular iff the corresponding unlabelled process is strongly regular. This fact follows easily from our construction.

3.3.1 Petri Nets

In this section we show that the class of transition systems generated by labelled Petri nets is closed under the transformation from labelled to unlabelled systems.

Definition 3.2 (Unlabelled Petri net).
A labelled Petri net $N = (P, T, F, \mathcal{A}ct, \lambda)$ is called *unlabelled* Petri net whenever $|\mathcal{A}ct| = 1$.

Remark 3.4. Whenever $|\mathcal{A}ct| = 1$, let us say $\mathcal{A}ct = \{a\}$, we can omit $\mathcal{A}ct$ and λ from the definition of the net N and instead of $M \xrightarrow{a} M'$ in $T(N)$ we simply write $M \longrightarrow M'$. The net then generates an unlabelled transition system $T(N)$.

Let $N = (P, T, F, \mathcal{A}ct, \lambda)$ be a labelled Petri net. Without loss of generality assume that $\mathcal{A}ct = \{1, \ldots, n\}$ for some $n > 0$. We construct an unlabelled Petri net $N' = (P', T', F')$ and a mapping $\psi : [P \to \mathbb{N}_0] \to [P' \to \mathbb{N}_0]$ from markings in N to markings in N' such that $\widehat{T(N)}_{M_1}$ and $T(N')_{\psi(M_1)}$ are isomorphic unlabelled transition systems for any marking M_1 of N. Let us recall that $\widehat{T(N)}_{M_1}$ is the unlabelled transition system restricted to markings reachable from M_1 and $T(N')_{\psi(M_1)}$ is restricted to markings reachable from $\psi(M_1)$ — see Definition 2.3. The net N' is defined as follows:

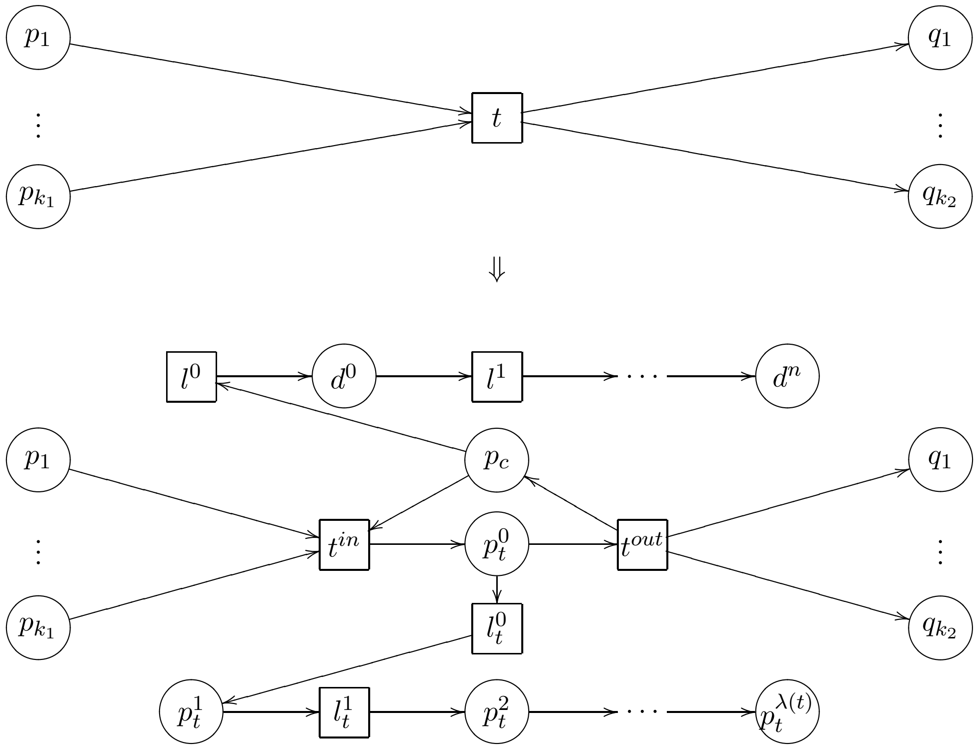


Figure 3.3: Transformation of a transition t

$$P' \overset{\text{def}}{=} P \cup \{p_t^k \mid t \in T \ \wedge\ 0 \le k \le \lambda(t)\} \cup \{p_c\} \cup \{d^k \mid 0 \le k \le n\}$$

$$T' \overset{\text{def}}{=} \{t^{in}, t^{out} \mid t \in T\} \cup \{l_t^k \mid t \in T \ \wedge\ 0 \le k < \lambda(t)\} \cup \{l^k \mid 0 \le k \le n\}$$

$$\begin{aligned} F' \overset{\text{def}}{=}\ & \{(p, t^{in}) \mid (p,t) \in F\} \cup \{(t^{out}, p) \mid (t,p) \in F\} \cup \\ & \{(t^{in}, p_t^0), (p_t^0, t^{out}) \mid t \in T\} \cup \\ & \{(p_t^k, l_t^k), (l_t^k, p_t^{k+1}) \mid t \in T \ \wedge\ 0 \le k < \lambda(t)\} \cup \\ & \{(p_c, t^{in}), (t^{out}, p_c) \mid t \in T\} \cup \\ & \{(p_c, l^0)\} \cup \{(l^k, d^k), (d^k, l^{k+1}) \mid 0 \le k < n\} \cup \{(l^n, d^n)\}. \end{aligned}$$

In this construction each transition t with input places $p_1, \dots, p_{k_1}$ and output places $q_1, \dots, q_{k_2}$ is transformed into a set of transitions shown in Figure 3.3.

Now, we give the mapping ψ. Let $M \in [P \to \mathbb{N}_0]$. Then $\psi(M) : P' \to \mathbb{N}_0$ is defined by

$$\psi(M)(p) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } p = p_c \\ M(p) & \text{if } p \in P \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 3.3. Let $N = (P, T, F, L, \lambda)$ be a labelled Petri net and $N' = (P', T', F')$ the unlabelled Petri net defined above. Then $\widehat{T(N)_{M_1}}$ and $T(N')_{\psi(M_1)}$ are isomorphic unlabelled transition systems for any $M_1 \in [P \to \mathbb{N}_0]$.

Proof. Assume that $\widehat{T(N)_{M_1}} = (S_1, \longrightarrow_1)$ and $T(N')_{\psi(M_1)} = (S_2, \longrightarrow_2)$. Recall that $S_1 \subseteq [P \rightarrow \mathbb{N}_0] \cup \{r^k_{(M,\lambda(t),M')} \mid M[t\rangle M' \ \wedge\ 0 \leq k \leq \lambda(t)\} \cup \{d^k_M \mid M \in [P \rightarrow \mathbb{N}_0] \ \wedge\ 0 \leq k \leq n\}$ and $S_2 \subseteq [P' \rightarrow \mathbb{N}_0]$. We define a mapping $f : S_1 \rightarrow S_2$ by

$$f(s_1) \stackrel{\text{def}}{=} \begin{cases} \psi(s_1) & \text{if } s_1 \in [P \rightarrow \mathbb{N}_0] \\ \overline{M} & \text{if } s_1 = r^k_{(M,\lambda(t),M')} \text{ such that } M[t\rangle M' \\ \overline{\overline{M}} & \text{if } s_1 = d^k_M \text{ such that } M \in [P \rightarrow \mathbb{N}_0] \end{cases}$$

where

$$\overline{M}(p) \stackrel{\text{def}}{=} \begin{cases} M(p) & \text{if } p \in P \smallsetminus {}^\bullet t \\ M(p) - 1 & \text{if } p \in {}^\bullet t \\ 1 & \text{if } p = p^k_t \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \overline{\overline{M}}(p) \stackrel{\text{def}}{=} \begin{cases} M(p) & \text{if } p \in P \\ 1 & \text{if } p = d^k \\ 0 & \text{otherwise.} \end{cases}$$

Let $s_1 \longrightarrow_1 s'_1$ for some $s_1, s'_1 \in S_1$. It can be easily seen that $f(s_1) \longrightarrow_2 f(s'_1)$. On the other hand, let $M_2 \longrightarrow_2 M'_2$ and $M_2 = f(s_1)$ for some $s_1 \in S_1$ and $M_2, M'_2 \in S_2$. Then there exists $s'_1 \in S_1$ such that $M'_2 = f(s'_1)$ and $s_1 \longrightarrow_1 s'_1$. Also note that f is injective. Hence, $\widehat{T(N)_{M_1}}$ and $T(N')_{\psi(M_1)}$ are isomorphic unlabelled transition systems. □

Theorem 3.2. Let N be a labelled Petri net and M_1, M_2 a pair of markings in N. There is a polynomial time reduction producing an unlabelled Petri net N' and a pair of markings $\psi(M_1), \psi(M_2)$ in N' such that

$$(M_1, N) \sim (M_2, N) \quad \text{if and only if} \quad (\psi(M_1), N') \sim (\psi(M_2, N').$$

Proof. Directly from Lemma 3.3 and Theorem 3.1. □

Since the strong bisimilarity checking problem for labelled Petri nets is undecidable [90], we obtain the following undecidability result for unlabelled Petri nets.

Corollary 3.2. Strong bisimilarity checking problem for unlabelled Petri nets is undecidable.

Remark 3.5. We mentioned this corollary explicitly because we consider it to be of a special interest. However, all other undecidability and lower bound results for (unnormed) Petri nets (see Chapter 8) hold also for unlabelled Petri nets.

3.3.2 Pushdown Processes

In this section we show that the class of PDA transition systems is closed under the transformation from labelled to unlabelled systems.

Let Δ be a PDA system where Q is a set of the control states, Γ is the stack alphabet, $\mathcal{A}ct$ is the set of actions and $\Delta \subseteq Q \times \Gamma \times \mathcal{A}ct \times Q \times \Gamma^*$.

Definition 3.3 (Unlabelled pushdown system).
A pushdown system Δ is called *unlabelled* whenever $|\mathcal{A}ct| = 1$.

Remark 3.6. Whenever $|\mathcal{A}ct| = 1$, let us say $\mathcal{A}ct = \{a\}$, we can omit $\mathcal{A}ct$ from the definition of a pushdown system Δ and instead of $pA \xrightarrow{a} q\alpha$ we simply write $pA \longrightarrow q\alpha$ assuming that $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma^*$.

Our aim is to transform a pushdown system Δ into an unlabelled pushdown system Δ' such that strong bisimilarity is preserved. For technical convenience, we assume from now on (in this chapter) that Γ contains a distinct "dummy" symbol Z such that $pZ \not\longrightarrow$ for any $p \in Q$. Then trivially

$$(p_1\beta_1, \Delta) \sim (p_2\beta_2, \Delta) \quad \text{if and only if} \quad (p_1\beta_1 Z, \Delta) \sim (p_2\beta_2 Z, \Delta) \qquad (3.3)$$

for any $p_1, p_2 \in Q$ and $\beta_1, \beta_2 \in \Gamma^*$. In particular, all states reachable from $p\beta Z$ are of the form $q\beta' Z$ where $q \in Q$ and $\beta' \in \Gamma^*$.

Let Δ be a pushdown system with control states Q, stack alphabet Γ and the set of actions $\mathcal{A}ct$. Without loss of generality assume that $\mathcal{A}ct = \{1, \ldots, n\}$ for some $n > 0$. Moreover let $Z \in \Gamma$ be the "dummy" stack symbol.

We will construct an unlabelled pushdown system Δ' with control states Q and stack alphabet Γ' where $\Gamma \subseteq \Gamma'$ such that $\widehat{T(\Delta)_{p_1\alpha_1 Z}}$ and $T(\Delta')_{p_1\alpha_1 Z}$ are isomorphic unlabelled transition systems for any $p_1 \in Q$ and $\alpha_1 \in \Gamma^*$. Again, see Definition 2.3 for the notation of transition systems restricted to reachable states from $p_1\alpha_1 Z$. The definitions of Γ' and Δ' are as follows:

$$\begin{array}{lcl} \Gamma' & \overset{\text{def}}{=} & \Gamma \cup \{X^k_{(pA,a,q\alpha)} \mid (pA \xrightarrow{a} q\alpha) \in \Delta \ \wedge\ 0 \le k \le a\} \cup \\ & & \{D^k \mid 0 \le k \le n\} \end{array}$$

$$\begin{array}{lcl} \Delta' & \overset{\text{def}}{=} & \{(p, A, p, X^0_{(pA,a,q\alpha)}),\ (p, X^0_{(pA,a,q\alpha)}, q, \alpha) \mid (pA \xrightarrow{a} q\alpha) \in \Delta\} \cup \\ & & \{(p, X^k_{(pA,a,q\alpha)}, p, X^{k+1}_{(pA,a,q\alpha)}) \mid (pA \xrightarrow{a} q\alpha) \in \Delta \ \wedge\ 0 \le k < a\} \cup \\ & & \{(p, A, p, D^0 A) \mid p \in Q \ \wedge\ A \in \Gamma\} \cup \\ & & \{(p, D^k, p, D^{k+1}) \mid p \in Q \ \wedge\ 0 \le k < n\}. \end{array}$$

Notice that in particular $pX^a_{(pA,a,q\alpha)}\beta Z \not\longrightarrow$ and $pD^n\beta Z \not\longrightarrow$ for any $\beta \in \Gamma'^*$. Graphical representation showing the transformation of $pA\beta Z \xrightarrow{a} q\alpha\beta Z$ where $\beta \in \Gamma^*$ and $(pA \xrightarrow{a} q\alpha) \in \Delta$ can be seen in Figure 3.4.

Lemma 3.4. Let Δ be a pushdown system and let Δ' be the unlabelled pushdown system defined above. Then $\widehat{T(\Delta)_{p_1\alpha_1 Z}}$ and $T(\Delta')_{p_1\alpha_1 Z}$ are isomorphic unlabelled transition systems for any $p_1 \in Q$ and $\alpha_1 \in \Gamma^*$.

Proof. Immediately from the construction. Notice that it is important that any reachable state in $T(\Delta')_{p_1\alpha_1 Z}$ ends with Z. In particular, from any state of the form $p\beta Z$ where $p \in Q$ and $\beta \in \Gamma^*$ (even if $\beta = \epsilon$) the following transition is possible in $T(\Delta')$: $p\beta Z \longrightarrow pD^0\beta Z$. □

Theorem 3.3. Let Δ be a pushdown system and $p_1\beta_1, p_2\beta_2$ a pair of states in $T(\Delta)$. There is a polynomial time reduction producing an unlabelled pushdown system Δ' (containing the "dummy" stack symbol Z) such that

$$(p_1\beta_1, \Delta) \sim (p_2\beta_2, \Delta) \quad \text{if and only if} \quad (p_1\beta_1 Z, \Delta') \sim (p_2\beta_2 Z, \Delta').$$

$$\begin{tikzcd}
pA\beta Z \arrow[r, "a"] & q\alpha\beta Z
\end{tikzcd}$$

$$\Downarrow$$

$$\begin{tikzcd}
pD^nA\beta Z & pD^0A\beta Z \arrow[l, dotted] & & qD^0\alpha\beta Z \arrow[r, dotted] & qD^n\alpha\beta Z \\
 & pA\beta Z \arrow[u] \arrow[r] & pX^0_{(pA,a,q\alpha)}\beta Z \arrow[r] \arrow[dl, bend left] & q\alpha\beta Z \arrow[u] & \\
 & pX^1_{(pA,a,q\alpha)}\beta Z \arrow[r] & pX^2_{(pA,a,q\alpha)}\beta Z \arrow[r, dotted] & pX^a_{(pA,a,q\alpha)}\beta Z &
\end{tikzcd}$$

Figure 3.4: Transformation of a transition $pA\beta Z \xrightarrow{a} q\alpha\beta Z$

Proof. Directly from Lemma 3.4 together with (3.3), and from Theorem 3.1. □

Since weak bisimilarity checking problem of pushdown processes is undecidable (see Chapter 7), we get the following corollary.

Corollary 3.3. Weak bisimilarity of unlabelled pushdown processes is undecidable.

Remark 3.7. We mentioned explicitly only one corollary which we consider to be of a special interest. However, all other undecidability and lower bound results for (unnormed) pushdown systems (see e.g. Chapter 8) hold also for unlabelled pushdown systems.

3.4 Concluding Remarks

By imposing a special restriction on the number of control states of a pushdown system to be a singleton set, let us say $Q \overset{\text{def}}{=} \{p\}$, we obtain a BPA system with deadlocks (see Subsection 2.5.2). The class of BPA systems with deadlocks is called BPA$_\delta$ in this section.

In the context of single-state PDA systems a *deadlock* δ is a stack symbol which has no defining equation, i.e., $p\delta \not\longrightarrow$. The class of labelled transition systems generated by BPA$_\delta$ is strictly more expressive (w.r.t. strong bisimilarity) than the BPA class without such deadlocks [169]. Observe that the BPA$_\delta$ class is closed under the transformation from labelled transition systems to unlabelled ones — the number of control states of a pushdown system Δ is the same as the number of control states of the transformed unlabelled pushdown system Δ'.

However, the BPA class (without deadlocks) is not closed under the transformation: let $\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{A\}$, $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{a, b\}$ and let Δ be a BPA system such that

$$\Delta \overset{\text{def}}{=} \{A \xrightarrow{a} A.A, \quad A \xrightarrow{b} \epsilon\}.$$

The minimal norm of any state from $\widehat{T(\Delta)}$ is less or equal to 3 where $3 = |\{a,b\}| + 1$. Moreover $(A^k, \widehat{T(\Delta)}) \not\sim (A^{k'}, \widehat{T(\Delta)})$ for any $k, k' \in \mathbb{N}_0$ such that $k \neq k'$. On the other hand there are only finitely many (even on the syntactical level) states with the minimal norm less or equal to 3 in any BPA system without deadlocks. Hence $(A, \widehat{T(\Delta)})$ cannot be bisimilar to any BPA system.

Similarly the BPP class is not closed under the transformation — it is enough to replace in our system Δ the sequential composition with the parallel one. This demonstrates that process algebras BPA (without deadlocks) and BPP are not strong enough to describe deadlock behaviour which is essential for our reduction.

On the other hand, the strong bisimilarity checking problem for BPA is in 2-EXPTIME [35] and it is known to be PSPACE-hard (see Chapter 6). In order to prove the containment of the problem in some lower complexity class (e.g. in PSPACE), it is enough to demonstrate a decision algorithm (running with the corresponding complexity) for *unlabelled* BPA$_\delta$ systems where moreover from any reachable state a deadlocked state (starting with δ) can be reached. In other words (using a construction from [169]) it is enough to show it for unnormed BPA systems in a very restricted form: there is a distinguished process constant D such that $D \xrightarrow{d} D$ for a distinguished action d (this is the only rewrite rule for D and this is the only place where the action d appears); every other rewrite rule is under a single action a and moreover from any reachable state there is a computation which ends in some state starting with D.

3.5 Bibliographical Remarks

This chapter is based on the paper "On the Power of Labels in Transition Systems" [171] (more precisely on the extended version of the paper [172]). In addition to the results about strong bisimilarity, the paper [171] also contains discussion about unlabelled transition systems with regard to model checking problems. In particular, it is shown that the presented transformation preserves the answers to checking system properties with modal μ-calculus and its sublogics like Hennessy-Milner logic, EF-logic, EG-logic, UB and CTL. This means that given a process (s, T) where T is a labelled transition system, and given a formula ϕ from one of the mentioned logics we can construct $\widehat{\phi}$ (a formula from the same logic as ϕ) such that $T, s \models \phi$ if and only if $\widehat{T}, s \models \widehat{\phi}$, where $\widehat{T}$ is the unlabelled transition system defined in this chapter.

Chapter 4

Extending Tableau Technique for BPP

In this chapter we define a class of transition systems called *effective commutative transition systems* (ECTS) and show, by generalising a tableau-based proof for BPP, that strong bisimilarity between any two states of such a transition system is decidable. This gives a general technique for extending decidability borders of strong bisimilarity for a wide class of infinite-state transition systems. This is demonstrated for several process formalisms, namely BPP systems, lossy BPP processes, BPP processes with interrupt and timed-arc BPP nets.

4.1 Motivation

Semantics to various formalisms for description of infinite-state processes is usually given in terms of labelled transition systems. This provides a common ground for studying such systems, and the usually considered problems like model checking and equivalence checking can be defined purely in terms of labelled transition systems (see Chapter 2). In this chapter we show a general approach for extending decidability borders of strong bisimilarity checking for commutative-based process formalisms.

It is known that strong bisimilarity is undecidable for a typical representative of fully parallel models — Petri nets [90]. Nevertheless, in [42, 43] Christensen, Hirshfeld and Moller proved (using a tableau technique) that strong bisimilarity is decidable for an important fragment of Petri nets called *communication-free Petri nets* (this fragment was introduced as BPP systems in Chapter 2). For a detailed overview of the tableau techniques consult e.g. [97].

We will abstract from the specific BPP syntax and generalize the tableau proof to a class of transition systems called effective commutative transition systems (ECTS). We give six simple conditions on a transition system to be an ECTS and if all of them are satisfied then strong bisimilarity between any two states of the transition system is decidable. There is no need to know the syntactic description of the system. Moreover, the generalisation is achieved in several ways: (i) states can be tuples of bounded multisets of natural numbers and not only tuples of natural numbers, (ii) we do not insist on a specific computation of successors of a given state — any effectively computable and finite set of successors is acceptable, and (iii) an auxiliary equivalence relation on states is introduced in order to check invariants of bisimilar pairs.

Semantics of many formalisms can be defined as an ECTS and this yields immediately decidability of strong bisimilarity. We will demonstrate this on four examples — BPP process algebra, lossy BPP processes, BPP systems with interrupt and timed-arc BPP nets — thus extending in several ways the known decidability border-line which lies somewhere between BPP systems and state-extended BPP systems (state-extended BPP systems, also called PPDA, are a strict subclass of Petri nets where strong bisimilarity is still undecidable [34, 97]).

4.2 Preliminary Definitions

Let $\mathbb{N}_0 = \{0, 1, \dots\}$ be the set of natural numbers. A *multiset* of $\mathbb{N}_0$ is a function $M : \mathbb{N}_0 \to \mathbb{N}_0$. Let $i \in \mathbb{N}_0$, then $M(i)$ denotes the number of occurrences of i in the multiset M. The *empty multiset* $\emptyset$ is a function such that $\emptyset(i) = 0$ for all $i \in \mathbb{N}_0$. The *multiset union* of two multisets M_1 and M_2 is defined by $(M_1 \uplus M_2)(i) = M_1(i) + M_2(i)$ for all $i \in \mathbb{N}_0$. By $\mathcal{B}_\infty$ we denote the set of all multisets of $\mathbb{N}_0$. Let $m \in \mathbb{N}_0$. We define a set $\mathcal{B}_m$ of all multisets of $\{0, 1, \dots, m\}$, i.e., $M \in \mathcal{B}_m$ iff $M \in \mathcal{B}_\infty$ and $M(i) = 0$ for all $i \in \mathbb{N}_0$ such that $i > m$. We call a multiset $M \in \mathcal{B}_\infty$ *finite* if there is some $m \in \mathbb{N}_0$ such that $M \in \mathcal{B}_m$.

Remark 4.1. For finite multisets we sometimes use an alternative set-like notation: e.g. a multiset $\{0, 1, 1, 4, 4, 4\}$ is the same as a multiset M such that $M(0) = 1$, $M(1) = 2$, $M(4) = 3$ and $M(i) = 0$ for $i \in \mathbb{N}_0 \smallsetminus \{0, 1, 4\}$.

Definition 4.1. Let $M, N \in \mathcal{B}_m$. We write $M \prec_\ell N$ iff there is k, $0 \le k \le m$, such that $M(k) < N(k)$ and $M(i) = N(i)$ for all i, $0 \le i < k$.

The following proposition follows from the definition.

Proposition 4.1. Let $M, N \in \mathcal{B}_m$. Then $M \neq N$ implies that either $M \prec_\ell N$ or $N \prec_\ell M$.

Definition 4.2. Let $M, N \in \mathcal{B}_m$. We write $M \preceq_c N$ iff $M(i) \le N(i)$ for every i, $1 \le i \le m$, i.e., iff there is $M' \in \mathcal{B}_m$ such that $N = M \uplus M'$.

Let $m, n \in \mathbb{N}_0$ and $n > 0$. We define a structure $S = (\mathcal{B}_m^n, \oplus, \emptyset^n)$ where $\mathcal{B}_m^n$ is a set of n-tuples of elements from $\mathcal{B}_m$. Let $\alpha = (M_1, M_2, \dots, M_n) \in \mathcal{B}_m^n$ and $\beta = (N_1, N_2, \dots, N_n) \in \mathcal{B}_m^n$, then $\alpha \oplus \beta = (M_1 \uplus N_1, M_2 \uplus N_2, \dots, M_n \uplus N_n)$. Of course, $\alpha \oplus \beta \in \mathcal{B}_m^n$. The structure S is a commutative monoid. If $\alpha \in \mathcal{B}_m^n$ then α_i, $1 \le i \le n$, is the i'th coordinate of α. We introduce two orderings on $\mathcal{B}_m^n$.

Definition 4.3. Let $\alpha, \beta \in \mathcal{B}_m^n$, then

$$\begin{array}{lll} \alpha <_\ell \beta & \text{iff} & \text{there is } k,\ 1 \le k \le n, \text{ such that} \\ & & \quad \alpha_k \prec_\ell \beta_k \text{ and } \alpha_i = \beta_i \text{ for every } i,\ 1 \le i < k \\ & & \\ \alpha \le_c \beta & \text{iff} & \alpha_i \preceq_c \beta_i \text{ for every } i,\ 1 \le i \le n. \end{array}$$

Observe that $<_\ell$ is a well-founded ordering (there is no infinite sequence $\alpha_1, \alpha_2, \ldots$ such that $\alpha_1 >_\ell \alpha_2 >_\ell \ldots$) since $\prec_\ell$ is well-founded. Moreover for any $\alpha \neq \beta$ either $\alpha <_\ell \beta$ or $\beta <_\ell \alpha$. Also notice that $\alpha \leq_c \beta$ iff there is $\alpha' \in \mathcal{B}^n_m$ such that $\beta = \alpha \oplus \alpha'$. We write $\alpha <_c \beta$ iff $\alpha \leq_c \beta$ and $\alpha \neq \beta$. The following lemma is a simple generalisation of Dickson's Lemma [54].

Lemma 4.1. Every infinite sequence of elements from $\mathcal{B}^n_m$ has an infinite non-decreasing subsequence w.r.t. $\leq_c$.

Now we extend the notion of labelled transition systems introduced is Chapter 2 with an equivalence relation in order to check invariants during strong bisimulation game. This means that a necessary condition for two states to be strongly bisimilar is that they are related by the equivalence relation. The definition of strong bisimilarity is then also appropriately generalized.

Definition 4.4 (Generalized labelled transition system).
A *(generalized) labelled transition system* is a 4-tuple $(S, \mathcal{A}ct, \longrightarrow, \mathcal{E}qv)$ where

- S is a set of *states* (or *processes*),
- $\mathcal{A}ct$ is a set of *labels* (or *actions*),
- $\longrightarrow \subseteq S \times \mathcal{A}ct \times S$ is a *transition relation*, and
- $\mathcal{E}qv \subseteq S \times S$ is an *equivalence relation* on states.

Remark 4.2. Our definition of labelled transition systems is a generalisation of labelled transition systems with final states — see the overview papers [137] and [34]. Let $F \subseteq S$ be a set of final states. In order to recover the definition from [137, 34] we define $(\alpha, \beta) \in \mathcal{E}qv$ iff $\alpha \in F$ and $\beta \in F$, or $\alpha \notin F$ and $\beta \notin F$.

Definition 4.5 (Generalized strong bisimilarity).
Let $T = (S, \mathcal{A}ct, \longrightarrow, \mathcal{E}qv)$ be a labelled transition system. A binary relation $R \subseteq S \times S$ is a *(generalized) strong bisimulation* iff whenever $(\alpha, \beta) \in R$ then for each $a \in \mathcal{A}ct$:

- if $\alpha \xrightarrow{a} \alpha'$ then $\beta \xrightarrow{a} \beta'$ for some β' such that $(\alpha', \beta') \in R$
- if $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ for some α' such that $(\alpha', \beta') \in R$
- $(\alpha, \beta) \in \mathcal{E}qv$.

States $\alpha, \beta \in S$ are *strongly bisimilar*, written $(\alpha, T) \sim (\beta, T)$, iff $(\alpha, \beta) \in R$ for some strong bisimulation R. If T is clear from the context, we write only $\alpha \sim \beta$ instead of $(\alpha, T) \sim (\beta, T)$.

In this chapter we will also need the notion of bisimulation approximations [135].

Definition 4.6. Let $T = (S, \mathcal{A}ct, \longrightarrow, \mathcal{E}qv)$ be a labelled transition system. The *stratified bisimulation relations* $\sim_k \subseteq S \times S$ for $k \in \mathbb{N}_0$ are defined as follows:

- $\alpha \sim_0 \beta$ for all $\alpha, \beta \in S$ such that $(\alpha, \beta) \in \mathcal{E}qv$, i.e., $\sim_0 = \mathcal{E}qv$
- $\alpha \sim_{k+1} \beta$ iff for each $a \in \mathcal{A}ct$:
  - if $\alpha \xrightarrow{a} \alpha'$ then $\beta \xrightarrow{a} \beta'$ for some β' such that $\alpha' \sim_k \beta'$
  - if $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ for some α' such that $\alpha' \sim_k \beta'$
  - $(\alpha, \beta) \in \mathcal{E}qv$.

Remark 4.3. We remind the reader of the fact that $\sim_k$ is an equivalence relation for every $k \in \mathbb{N}_0$.

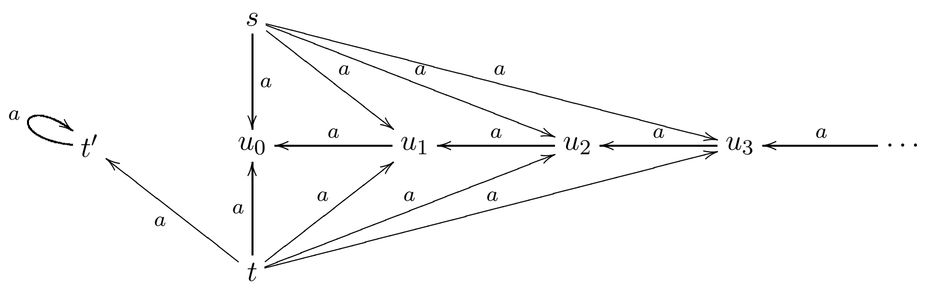
Given a labelled transition system $T = (S, \mathcal{A}ct, \longrightarrow, \mathcal{E}qv)$ we define the set $\mathsf{next}(\alpha, a) \overset{\text{def}}{=} \{\beta \in S \mid \alpha \xrightarrow{a} \beta\}$ for $\alpha \in S$ and $a \in \mathcal{A}ct$. We also define $\mathsf{next}(\alpha, *) \overset{\text{def}}{=} \bigcup_{a \in \mathcal{A}ct} \mathsf{next}(\alpha, a)$. The system T is *image-finite* iff the set $\mathsf{next}(\alpha, a)$ is finite for every $\alpha \in S$ and $a \in \mathcal{A}ct$. The following proposition is a standard one.

Proposition 4.2. Let $T = (S, \mathcal{A}ct, \longrightarrow, \mathcal{E}qv)$ be an image-finite labelled transition system and $\alpha, \beta \in S$. Then $\alpha \sim \beta$ if and only if $\alpha \sim_k \beta$ for all $k \in \mathbb{N}_0$.

Example 4.1. We demonstrate that image-finiteness of T is a necessary condition for validity of Proposition 4.2. Let us consider a transition system $T \overset{\text{def}}{=} (S, \mathcal{A}ct, \longrightarrow, S \times S)$ where $S \overset{\text{def}}{=} \{s, t, t'\} \cup \{u_i \mid i \in \mathbb{N}_0\}$, $\mathcal{A}ct \overset{\text{def}}{=} \{a\}$ and $\longrightarrow$ is given by

$$\begin{array}{rl} u_{i+1} \xrightarrow{a} u_i & \text{for all } i \in \mathbb{N}_0 \\ s \xrightarrow{a} u_i & \text{for all } i \in \mathbb{N}_0 \\ t \xrightarrow{a} u_i & \text{for all } i \in \mathbb{N}_0 \\ t \xrightarrow{a} t' & t' \xrightarrow{a} t'. \end{array}$$

A fragment of T is presented in the following picture. Observe that T is not image-finite since e.g. $\mathsf{next}(s, a) = \{u_i \mid i \in \mathbb{N}_0\}$ is not a finite set.


$$\begin{tikzcd} & & s \arrow[d, "a"] \arrow[dr, "a"] \arrow[drr, "a"] \arrow[drrr, "a"] & & & & \\ t' \arrow[loop left, "a"] & & u_0 & u_1 \arrow[l, "a"'] & u_2 \arrow[l, "a"'] & u_3 \arrow[l, "a"'] & \cdots \arrow[l, "a"'] \\ & & t \arrow[ull, "a"] \arrow[u, "a"] \arrow[ur, "a"] \arrow[urr, "a"] \arrow[urrr, "a"] & & & & \end{tikzcd}$$

It is now easy to see that $s \sim_k t$ for all $k \in \mathbb{N}_0$ but still $s \not\sim t$. □

4.3 The Method

In this section we introduce the class of effective commutative transition systems and show that strong bisimilarity is decidable for any pair of processes from this class.

Definition 4.7 (Effective commutative transition system). A labelled transition system $T = (S, \mathcal{A}ct, \longrightarrow, \mathcal{E}qv)$ is an *effective commutative transition system* (ECTS) iff there exist $n, m \in \mathbb{N}_0$, $n > 0$ such that the following conditions are satisfied:

(1) $S = \mathcal{B}_m^n$,

(2) $\mathcal{A}ct$ is a finite set,

(3) given $\alpha, \beta \in S$ it is decidable whether $(\alpha, \beta) \in \mathcal{E}qv$,

(4) $\mathsf{next}(\alpha, a)$ is effectively constructible for every $\alpha \in \mathcal{B}_m^n$ and $a \in \mathcal{A}ct$,

(5) T is image-finite, i.e., $\mathsf{next}(\alpha, a)$ is finite for every $\alpha \in \mathcal{B}_m^n$ and $a \in \mathcal{A}ct$,

(6) stratified bisimulation relations $\sim_k$ are *congruences* on $(\mathcal{B}_m^n, \oplus, \emptyset^n)$ for all $k \in \mathbb{N}_0$, i.e., if $\alpha \sim_k \beta$ then $(\alpha \oplus \gamma) \sim_k (\beta \oplus \gamma)$ for every $\alpha, \beta, \gamma \in \mathcal{B}_m^n$.

Since any ECTS is image-finite (5), the fact that $\sim_k$ are congruences (6) together with Proposition 4.2 implies:

(6') if $\alpha \sim \beta$ then $(\alpha \oplus \gamma) \sim (\beta \oplus \gamma)$ for every $\alpha, \beta, \gamma \in \mathcal{B}_m^n$.

We can now state the main theorem of this chapter.

Theorem 4.1 (Decidability of strong bisimilarity for ECTS).
Let $T = (\mathcal{B}_m^n, \mathcal{A}ct, \longrightarrow, \mathcal{E}qv)$ be an ECTS. Given $A, B \in \mathcal{B}_m^n$, it is decidable whether $(A, T) \sim (B, T)$.

Proof. The proof is by tableau-technique and it is a generalisation of the tableau-based proof used by Christensen, Hirshfeld and Moller in order to demonstrate decidability of strong bisimilarity for BPP [42, 43].

A tableau for $(A, B) \in \mathcal{B}_m^{2n}$ is a maximal proof tree rooted with (A, B) and built according to the following rules. Let (α, β) be a node in the tree. The node (α, β) is either *terminal (leaf)* or *nonterminal*. The following nodes are terminal:

- (α, α) is a *successful leaf* for any $\alpha \in \mathcal{B}_m^n$ (note that always $(\alpha, \alpha) \in \mathcal{E}qv$),
- (α, β) is a *successful leaf* if $\mathsf{next}(\alpha, *) \cup \mathsf{next}(\beta, *) = \emptyset$ and $(\alpha, \beta) \in \mathcal{E}qv$,
- (α, β) is an *unsuccessful leaf* if for some $a \in \mathcal{A}ct$ it is the case that $\mathsf{next}(\alpha, a) \cup \mathsf{next}(\beta, a) \neq \emptyset$, and either $\mathsf{next}(\alpha, a) = \emptyset$ or $\mathsf{next}(\beta, a) = \emptyset$,
- (α, β) is an *unsuccessful leaf* if $(\alpha, \beta) \notin \mathcal{E}qv$.

We say that a node is an *ancestor* of (α, β) if it is on the path from the root to (α, β) and at least one application of the rule EXPAND (defined later) separates them. If (α, β) is not a leaf then we reduce it using the following RED rules as long as possible.

$$\text{RED}_L \quad \frac{(\alpha, \beta)}{(\gamma \oplus \omega, \beta)} \quad \text{if there is an ancestor } (\gamma, \delta) \text{ or } (\delta, \gamma) \text{ of } (\alpha, \beta) \text{ such that } \gamma <_\ell \delta \text{ and } \alpha = \delta \oplus \omega \text{ for some } \omega \in \mathcal{B}_m^n$$

$$\text{RED}_R \quad \frac{(\alpha,\beta)}{(\alpha,\gamma\oplus\omega)} \quad \begin{array}{l}\text{if there is an ancestor } (\gamma,\delta) \text{ or } (\delta,\gamma) \text{ of } (\alpha,\beta) \text{ such}\\ \text{that } \gamma <_\ell \delta \text{ and } \beta = \delta\oplus\omega \text{ for some } \omega\in\mathcal{B}^n_m\end{array}$$

If no other reduction RED is applicable and the resulting node is not a leaf, we apply the rule EXPAND for a set of relations S_a, $a \in \mathcal{A}ct$, where $S_a \subseteq \mathsf{next}(\alpha,a)\times\mathsf{next}(\beta,a)$ such that $\forall\alpha'\in\mathsf{next}(\alpha,a).\exists\beta'\in\mathsf{next}(\beta,a).\ (\alpha',\beta')\in S_a$ and $\forall\beta'\in\mathsf{next}(\beta,a).\exists\alpha'\in\mathsf{next}(\alpha,a).\ (\alpha',\beta')\in S_a$.

$$\text{EXPAND} \quad \frac{(\alpha,\beta)}{\{(\alpha',\beta') \mid a\in\mathcal{A}ct \ \wedge\ (\alpha',\beta')\in S_a\}}$$

The set notation used in the rule EXPAND means that each element (α',β') in the conclusion of the rule becomes a new child in the proof tree. Now, we start again applying the RED-rules to every child which is not a leaf as long as possible. Note that reduction rules are applicable to a node iff the node is not a terminal one.

Lemma 4.2. Any tableau for (A,B) is finite and there are only finitely many tableaux.

Proof. Observe that any tableau for (A,B) is finitely branching because of the assumption (5) and the condition that $\mathcal{A}ct$ is finite (2), which implies that for a given $a\in\mathcal{A}ct$ any relation S_a is finite and there are finitely many such relations. Should the tableau be infinite, there is an infinite branch which gives an infinite sequence of vectors from $\mathcal{B}^{2n}_m$. Since the rules RED can be used only finitely many times in a sequence (they decrease the $<_\ell$ order, which is well founded), there must be an infinite subsequence of vectors on which the rule EXPAND was applied. Using Lemma 4.1, this sequence must contain an infinite nondecreasing subsequence $p_1 \leq_c p_2 \leq_c \ldots$. However, the rule EXPAND cannot be applied on p_2 since one of the rules RED is applicable. This is a contradiction.

Since there are only finitely many relations S_a (where $a\in\mathcal{A}ct$) available for the EXPAND rule and finitely many possibilities for applications of the RED rules, there are always finitely many possibilities how to extend already existing partial tableau. Suppose that there are infinitely many tableaux starting from (A,B). Then there must be a tableau for (A,B) with an infinite branch, which contradicts that every tableau is finite. □

We call a tableau for (A,B) *successful* if it is maximal (no further rules are applicable) and all its leaves are successful.

Lemma 4.3 (Completeness). If $A\sim B$ then there is a successful tableau for (A,B).

Proof. We shall construct a tableau from the root (A,B) such that every node (α,β) in the tableau satisfies $\alpha\sim\beta$. Hence this tableau cannot contain any unsuccessful leaf and it must be finite because of Lemma 4.2. Suppose that (α,β) is already a node in the tableau such that $\alpha\sim\beta$ and consider the rule RED_L applied on (α,β). We may assume that $\gamma\sim\delta$, which means using (6') that $(\gamma\oplus\omega)\sim(\delta\oplus\omega)=\alpha\sim\beta$. Hence $(\gamma\oplus\omega)\sim\beta$. Similarly for RED_R. From the definition of $\sim$ follows that the rule EXPAND is also forward sound, i.e., if $\alpha \sim \beta$ then we can choose for every $a \in \mathcal{A}ct$ a relation S_a such that $(\alpha', \beta') \in S_a$ implies that $\alpha' \sim \beta'$. □

Lemma 4.4 (Soundness). If there is a successful tableau for (A, B) then $A \sim B$.

Proof. For the sake of contradiction assume that there is a successful tableau for (A, B) and $A \not\sim B$. We will show that we can construct a path from the root (A, B) to some leaf such that for any pair (α, β) on this path $\alpha \not\sim \beta$.

If $A \not\sim B$ then using Proposition 4.2 there is a minimal k such that $A \not\sim_k B$. Notice that if $\alpha \not\sim_k \beta$ such that k is minimal and we apply the rule EXPAND on (α, β), then at least one of its children (α', β') satisfies that $\alpha' \not\sim_{k-1} \beta'$. We choose such a child to extend our path from the root.

If we apply RED$_L$ on (α, β) where $\alpha \not\sim_k \beta$ and k is minimal, then the corresponding ancestor (γ, δ) is separated by at least one application of EXPAND and so $\gamma \sim_k \delta$. This implies that $(\gamma \oplus \omega) \not\sim_k \beta$, otherwise using the assumption (6) we get that $\alpha = (\delta \oplus \omega) \sim_k (\gamma \oplus \omega) \sim_k \beta$, which is a contradiction with $\alpha \not\sim_k \beta$. The same is true for RED$_R$. Thus there must be a path from the root (A, B) to some leaf such that for any pair (α, β) on this path $\alpha \not\sim \beta$. This is a contradiction with the fact that the path contains a successful leaf. □

We proved that it is decidable whether $A \sim B$, since it is the case iff there is a successful tableau for (A, B). There are only finitely many tableaux and all of them are finite, moreover the conditions (3) and (4) ensure that they are effectively constructible. □

4.4 Applications

In this section we consider several specific classes of commutative transition systems. We study in particular BPP and lossy BPP processes, interrupt BPP systems and timed-arc BPP nets.

4.4.1 BPP and Deadlock-Sensitive BPP

It is a well known fact that strong bisimilarity is decidable for BPP [42, 43]. As our first application we reprove this result in a slightly more general way. Let us consider a BPP system Δ (as defined in Chapter 2). In addition let us recall that $\mathcal{P}(\mathcal{C}onst(\Delta))$ is the set of parallel process expressions over $\mathcal{C}onst(\Delta)$, and that $\equiv$ is the structural congruence over $\mathcal{P}(\mathcal{C}onst(\Delta))$ as introduced in Definition 2.10. Given a process expression E by $[E]_\equiv$ we denote the equivalence class represented by E, and $\mathcal{P}(\mathcal{C}onst(\Delta))/\!\equiv$ stands for the set of $\equiv$-equivalence classes of parallel process expressions over $\mathcal{C}onst(\Delta)$.

We extend now the notion of $T(\Delta)$, i.e., of the transition system generated by Δ, by defining the equivalence relation $\mathcal{E}qv$ (in what follows we explicitly treat the states of $T(\Delta)$ as equivalence classes w.r.t. $\equiv$).

There are two possibilities for defining the equivalence relation $\mathcal{E}qv$. In the usual setting

$$\mathcal{E}qv \stackrel{\text{def}}{=} (\mathcal{P}(\mathcal{C}onst(\Delta))/\!\equiv) \times (\mathcal{P}(\mathcal{C}onst(\Delta))/\!\equiv)$$

is the universal relation (thus it is in fact unused) and this is the notion of ordinary BPP systems. Another possibility is to define $\mathcal{E}qv$ by

$$\mathcal{E}qv \stackrel{\text{def}}{=} \{(E,F) \in (\mathcal{P}(\mathcal{C}onst(\Delta))/\!\equiv) \times (\mathcal{P}(\mathcal{C}onst(\Delta))/\!\equiv) \mid E = [\epsilon]_\equiv \text{ iff } F = [\epsilon]_\equiv\}.$$

We call this class *deadlock-sensitive* BPP. A study of strict (deadlock-sensitive) and nonstrict (deadlock-nonsensitive) bisimilarity for a sequential analogue of BPP — Basic Process Algebra (BPA) — is provided in [169].

Example 4.2. Let us consider a BPP system

$$\Delta \stackrel{\text{def}}{=} \{X \xrightarrow{a} \epsilon,\ Y \xrightarrow{a} Z\}.$$

Obviously (X, Δ) and (Y, Δ) are strongly bisimilar when Δ is interpreted as ordinary (deadlock-nonsensitive) BPP. When Δ is considered as deadlock-sensitive, (X, Δ) and (Y, Δ) are not strongly bisimilar any more since Z is a deadlock and it is not bisimilar to the empty process 'ϵ'. □

We show that given a BPP system Δ, we can interpret its semantics as a commutative transition system such that states are elements of $\mathcal{B}_{n-1} = \mathcal{B}^1_{n-1}$ where $n = |\mathcal{C}onst(\Delta)|$. Because of the structural congruence $\equiv$, any process expression E over $\mathcal{C}onst(\Delta)$ can be written as a vector of n natural numbers. Suppose a fixed ordering on $\mathcal{C}onst(\Delta) = \{X_0, X_1, \ldots, X_{n-1}\}$. Then the corresponding vector contains on i'th coordinate the number of occurrences of the process constant X_i in E. Formally, we define a mapping $\phi : \mathcal{P}(\mathcal{C}onst(\Delta)) \to \mathcal{B}^1_{n-1}$ by

$$\begin{array}{rcl}
\phi(\epsilon) & \stackrel{\text{def}}{=} & \emptyset \\
\phi(X_i) & \stackrel{\text{def}}{=} & M \text{ such that } M(i) = 1 \text{ and } M(j) = 0 \text{ for } j \neq i \\
\phi(E_1 \| E_2) & \stackrel{\text{def}}{=} & \phi(E_1) \oplus \phi(E_2).
\end{array}$$

The following proposition is an easy observation.

Proposition 4.3. Let Δ be a BPP system and $E, F \in \mathcal{P}(\mathcal{C}onst(\Delta))$. Then $E \equiv F$ iff $\phi(E) = \phi(F)$.

Hence any rule $(X \xrightarrow{a} E) \in \Delta$ can be represented by $\phi(X) \xrightarrow{a} \phi(E)$. The system Δ, where $n = |\mathcal{C}onst(\Delta)|$, generates a commutative labelled transition system $T^c(\Delta) \stackrel{\text{def}}{=} (\mathcal{B}^1_{n-1}, \mathcal{A}ct(\Delta), \longrightarrow, \mathcal{E}qv)$, where $\alpha \xrightarrow{a} \beta$ iff there exists a rule $(X \xrightarrow{a} E) \in \Delta$ such that $\alpha = \phi(X) \oplus \omega$ and $\beta = \phi(E) \oplus \omega$ for some $\omega \in \mathcal{B}^1_{n-1}$. The relation $\mathcal{E}qv$ for BPP and deadlock-sensitive BPP is defined in the same fashion as above, i.e., $\mathcal{E}qv \stackrel{\text{def}}{=} \mathcal{B}^1_{n-1} \times \mathcal{B}^1_{n-1}$ or $\mathcal{E}qv \stackrel{\text{def}}{=} \{(\alpha, \beta) \in \mathcal{B}^1_{n-1} \times \mathcal{B}^1_{n-1} \mid \alpha = \emptyset \text{ iff } \beta = \emptyset\}$.

Example 4.3. Let us consider the following system Δ where $\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{X_0, X_1, X_2\}$ and $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{a, b, c\}$.

$$X_0 \xrightarrow{a} X_0\|X_1\|X_2\|X_1,\ X_0 \xrightarrow{a} \epsilon,\ X_1 \xrightarrow{b} \epsilon,\ X_2 \xrightarrow{c} \epsilon$$

Then $n = 3$ and e.g. $\phi(X_0) = \{0\}$, $\phi(X_0\|X_1\|X_2\|X_1) = \{0, 1, 1, 2\}$ and $\phi(\epsilon) = \emptyset$. A sequence of transitions

$$X_0 \xrightarrow{a} X_0\|X_1\|X_2\|X_1 \xrightarrow{a} X_0\|X_1\|X_2\|X_1\|X_1\|X_2\|X_1 \xrightarrow{b}$$

$$X_0\|X_2\|X_1\|X_1\|X_2\|X_1 \xrightarrow{c} X_0\|X_1\|X_1\|X_2\|X_1$$

has a straightforward analogue in $\mathcal{B}_2^1$:

$$\{0\} \xrightarrow{a} \{0, 1, 1, 2\} \xrightarrow{a} \{0, 1, 1, 1, 1, 2, 2\} \xrightarrow{b}$$

$$\{0, 1, 1, 1, 2, 2\} \xrightarrow{c} \{0, 1, 1, 1, 2\}.$$

□

Obviously, $T(\Delta)$ and $T^c(\Delta)$ are isomorphic labelled transition systems. We can now apply Theorem 4.1 in order to show that strong bisimilarity for BPP is decidable.

Theorem 4.2. Given a BPP system Δ (or a deadlock-sensitive BPP system Δ) and a pair of processes (P_1, Δ) and (P_2, Δ), it is decidable whether $(P_1, \Delta) \sim (P_2, \Delta)$.

Proof. It can be easily verified that $T^c(\Delta)$ defined above is an ECTS. Since $T(\Delta)$ and $T^c(\Delta)$ are isomorphic labelled transition systems, we can use Theorem 4.1. □

4.4.2 Lossy BPP

The notion of unreliability, in particular lossiness, has been intensively studied with a number of interesting results. Let us mention e.g. models like *lossy channel systems* [1] or *lossy vector addition systems* [27, 127]. Lossy BPP systems were studied in [127] in the context of model checking problems. In lossy BPP we allow process constants disappear spontaneously at any time. We give a formal definition of lossy BPP systems first.

A *lossy* BPP system Δ is defined as an ordinary BPP system. Only the semantics to Δ is given in a different way: a lossy BPP system Δ determines a labelled transition system $T(\Delta) \overset{\text{def}}{=} (\mathcal{P}(\mathcal{C}onst(\Delta))/\equiv, \mathcal{A}ct(\Delta) \cup \{drop\}, \longrightarrow, \mathcal{E}qv)$ where states are $\equiv$-equivalence classes of process expressions over $\mathcal{C}onst(\Delta)$, $\mathcal{A}ct(\Delta) \cup \{drop\}$ is the set of labels with a distinguished label $drop \notin \mathcal{A}ct(\Delta)$ modelling lossiness, the transition relation $\longrightarrow$ is defined by the SOS rules in Figure 4.1 (we slightly abuse the notation and write only E instead of $[E]_\equiv$) and $\mathcal{E}qv$ for lossy BPP can be defined as in the case of BPP — deadlock sensitive or deadlock nonsensitive.

$$\frac{(X \xrightarrow{a} E) \in \Delta}{X \xrightarrow{a} E} \qquad \frac{E \xrightarrow{a} E'}{E\|F \xrightarrow{a} E'\|F} \qquad \frac{}{E \xrightarrow{drop} F} \text{ if } \exists F' \neq \epsilon \text{ s.t. } E = F\|F'$$

Figure 4.1: SOS rules for lossy BPP

Example 4.4. Let Δ be the following lossy BPP system such that $\mathcal{C}onst(\Delta) \stackrel{\text{def}}{=} \{X_0\}$ and $\mathcal{A}ct(\Delta) \stackrel{\text{def}}{=} \{a, b\}$.

$$X_0 \xrightarrow{a} X_0\|X_0, \;\; X_0 \xrightarrow{b} \epsilon$$

Then $X_0 \longrightarrow^* X_0^k$ for any $k \in \mathbb{N}_0$ where $X_0^0 \stackrel{\text{def}}{=} \epsilon$ and $X_0^{k+1} \stackrel{\text{def}}{=} X_0\|X_0^k$. Also, $X_0^k \xrightarrow{drop} X_0^{k'}$ for any k', $0 \leq k' < k$, in particular, $X_0^k \xrightarrow{drop} \epsilon$ and $\epsilon \not\longrightarrow$. This means that any reachable state in (X_0, Δ) has norm at most 1. Moreover, $X_0^k \not\sim X_0^{k'}$ for any $k \neq k'$. Hence there cannot be any BPP process bisimilar to X_0 (there are only finitely many strongly nonbisimilar BPP states of norm less or equal to 1). On the other hand this property in general disallows to find a bisimilar lossy BPP process for a given BPP process. Thus the classes BPP and lossy BPP are, as expected, incomparable w.r.t. strong bisimilarity. $\square$

We are now ready to define semantics of lossy BPP in terms of commutative transition systems, similarly as for BPP. Let Δ be a lossy BPP system, where $n = |\mathcal{C}onst(\Delta)|$. By $T^c(\Delta) \stackrel{\text{def}}{=} (\mathcal{B}^1_{n-1}, \mathcal{A}ct(\Delta) \cup \{drop\}, \longrightarrow, \mathcal{E}qv)$ we denote a commutative transition system, where $\alpha \xrightarrow{a} \beta$ iff either (i) there is a rule $(X \xrightarrow{a} E) \in \Delta$ such that $\alpha = \phi(X) \oplus \omega$ and $\beta = \phi(E) \oplus \omega$ for some $\omega \in \mathcal{B}^1_{n-1}$, or (ii) $\beta <_c \alpha$ and $a = drop$. The relation $\mathcal{E}qv$ for lossy BPP can be defined in the same fashion as above (deadlock sensitive or deadlock nonsensitive).

Obviously, $T(\Delta)$ and $T^c(\Delta)$ are isomorphic labelled transition systems. This implies the following theorem for lossy BPP systems.

Theorem 4.3. Given a lossy BPP system Δ (either deadlock sensitive or deadlock nonsensitive) and a pair of processes (P_1, Δ) and (P_2, Δ), it is decidable whether $(P_1, \Delta) \sim (P_2, \Delta)$.

Proof. We show that $T^c(\Delta)$ is an ECTS and then we use Theorem 4.1 and the isomorphism between $T(\Delta)$ and $T^c(\Delta)$.

To verify conditions (1) – (5) of Definition 4.7 is easy. Let us now examine the condition (6). Assume that $\alpha \sim_k \beta$ and let $\gamma \in \mathcal{B}^1_{n-1}$. By induction on k we show that also $\alpha \oplus \gamma \sim_k \beta \oplus \gamma$.
Base case: If $k = 0$ then it is enough to show that if $(\alpha, \beta) \in \mathcal{E}qv$ then also $(\alpha \oplus \gamma, \beta \oplus \gamma) \in \mathcal{E}qv$. This is true for both deadlock sensitive and nonsensitive $\mathcal{E}qv$.
Inductive step: Let $k > 0$ and $\alpha \sim_k \beta$. Of course, $(\alpha \oplus \gamma, \beta \oplus \gamma) \in \mathcal{E}qv$. We will analyse the transitions from $\alpha \oplus \gamma$ only (the arguments for $\beta \oplus \gamma$ are symmetric).

Let $\alpha \oplus \gamma \xrightarrow{a} \kappa$ and $a \neq \mathit{drop}$. Then either $\kappa = \alpha' \oplus \gamma$ and $\alpha \xrightarrow{a} \alpha'$, or $\kappa = \alpha \oplus \gamma'$ and $\gamma \xrightarrow{a} \gamma'$. In the first case, because of our assumption that $\alpha \sim_k \beta$, also $\beta \xrightarrow{a} \beta'$ such that $\alpha' \sim_{k-1} \beta'$. Thus $\beta \oplus \gamma \xrightarrow{a} \beta' \oplus \gamma$ and using the induction hypothesis $\alpha' \oplus \gamma \sim_{k-1} \beta' \oplus \gamma$. The second case where $\kappa = \alpha \oplus \gamma'$ is analogical.

Let $\alpha \oplus \gamma \xrightarrow{\mathit{drop}} \kappa$. Then $\kappa = \alpha' \oplus \gamma'$ such that $\alpha' \leq_c \alpha$ and $\gamma' \leq_c \gamma$, and $\alpha' \oplus \gamma' <_c \alpha \oplus \gamma$. If $\alpha' = \alpha$ then $\beta \oplus \gamma \xrightarrow{\mathit{drop}} \beta \oplus \gamma'$ and using the induction hypothesis and the fact that $\alpha \sim_{k-1} \beta$ we get that $\alpha \oplus \gamma' \sim_{k-1} \beta \oplus \gamma'$. Let $\alpha <_c \alpha'$. Since $\alpha \sim_k \beta$ and $\alpha \xrightarrow{\mathit{drop}} \alpha'$, also $\beta \xrightarrow{\mathit{drop}} \beta'$ such that $\alpha' \sim_{k-1} \beta'$. Hence $\beta \oplus \gamma \xrightarrow{\mathit{drop}} \beta' \oplus \gamma'$ and using the induction hypothesis we know that $\alpha' \oplus \gamma' \sim_{k-1} \beta' \oplus \gamma'$. □

4.4.3 Interrupt BPP

In this subsection we investigate *mode transfer operators* in BPP systems, in particular the interrupt operator. Quoting [9]: "A useful feature in programming languages and specification languages is the ability to denote mode switches. In particular, most languages have means to describe the disrupt and interrupt of the normal execution of a system." Various mode transfer operators were considered in the literature [16, 9, 22, 49, 55]. We define interrupt BPP systems that extend the pure BPP systems with an interrupt vector and a mechanism for handling the interrupt. The motivation is that every state is annotated with a set of allowed interrupts and if no interrupt appears, a normal execution of the process is performed. At any time an interrupt can be raised by performing the action *int*. A normal execution of the process is interrupted and the raised interrupt is handled. During this all interrupts are disallowed. After the interrupt handling has been completed, the action *iret* is performed and a normal execution of the interrupted process continues.

Formally, an *interrupt* BPP system Δ is a pair (Δ_1, Δ_2) such that Δ_1 is a finite set $\Delta_1 \subseteq \mathcal{C}\mathit{onst} \times \mathcal{A}\mathit{ct} \times \mathcal{P} \times 2^{\mathcal{C}\mathit{onst}(\Delta_2)}$ and Δ_2 is a BPP system, where $\mathcal{P}$ is the class of parallel process expressions over $\mathcal{C}\mathit{onst}$ and $2^{\mathcal{C}\mathit{onst}(\Delta_2)}$ is the powerset of $\mathcal{C}\mathit{onst}(\Delta_2)$. We write $(X \xrightarrow{a} E, \mathit{enable})$ for $(X, a, E, \mathit{enable}) \in \Delta_1$. By $\mathcal{C}\mathit{onst}(\Delta_1)$ we denote the set of process constants that occur in the first and the third component of Δ_1.

An interrupt system $\Delta = (\Delta_1, \Delta_2)$ determines a labelled transition system $T(\Delta) \stackrel{\text{def}}{=} (S, \mathcal{A}\mathit{ct}, \longrightarrow, \mathcal{E}\mathit{qv}^u)$ such that

$$\begin{aligned} S &\stackrel{\text{def}}{=} \left(\mathcal{P}(\mathcal{C}\mathit{onst}(\Delta_1))/\equiv\right) \times 2^{\mathcal{C}\mathit{onst}(\Delta_2)} \times \{0,1\} \times \left(\mathcal{P}(\mathcal{C}\mathit{onst}(\Delta_2))/\equiv\right) \\ \mathcal{A}\mathit{ct} &\stackrel{\text{def}}{=} \mathcal{A}\mathit{ct}(\Delta_1) \cup \mathcal{A}\mathit{ct}(\Delta_2) \cup \{\mathit{int}, \mathit{iret}\} \end{aligned}$$

where states are 4-tuples (E_1, IV, IF, E_2) such that E_1 is a BPP process, IV is an *interrupt vector*, IF is an *interrupt flag* (0 means normal execution and 1 means interrupt call) and E_2 is ϵ if $IF = 0$ or it contains the interrupt handling process in case $IF = 1$. We assume that $\mathit{int}, \mathit{iret} \notin \mathcal{A}\mathit{ct}(\Delta_1) \cup \mathcal{A}\mathit{ct}(\Delta_2)$. The SOS rules for $\longrightarrow$ are defined in Figure 4.2 (E again represents $[E]_\equiv$ and '$\|$'

$$\frac{(X \xrightarrow{a} E_1, \mathit{enable}) \in \Delta_1}{(X, IV, 0, \epsilon) \xrightarrow{a} (E_1, IV \cup \mathit{enable}, 0, \epsilon)}$$

$$\frac{(E_1, IV, 0, \epsilon) \xrightarrow{a} (E_1', IV', 0, \epsilon)}{(E_1 \| F_1, IV, 0, \epsilon) \xrightarrow{a} (E_1' \| F_1, IV', 0, \epsilon)}$$

$$\frac{X \in IV}{(E_1, IV, 0, \epsilon) \xrightarrow{int} (E_1, IV, 1, X)} \qquad \frac{}{(E_1, IV, 1, \epsilon) \xrightarrow{iret} (E_1, IV, 0, \epsilon)}$$

$$\frac{(X \xrightarrow{a} E_2) \in \Delta_2}{(E_1, IV, 1, X) \xrightarrow{a} (E_1, IV, 1, E_2)} \qquad \frac{(E_1, IV, 1, E_2) \xrightarrow{a} (E_1, IV, 1, E_2')}{(E_1, IV, 1, E_2 \| F_2) \xrightarrow{a} (E_1, IV, 1, E_2' \| F_2)}$$

Figure 4.2: SOS rules for interrupt BPP

is commutative) and for the sake of simplicity let us assume that $\mathcal{E}qv^u$ is the universal relation on states.

Example 4.5. Let $\Delta \overset{\text{def}}{=}$

$$\Big(\Big\{(X_0 \xrightarrow{a} X_0 \| X_0, \{Y_0\}),\ (X_0 \xrightarrow{b} \epsilon, \{Y_1\})\Big\},\ \Big\{Y_0 \xrightarrow{c} \epsilon,\ Y_1 \xrightarrow{d} Y_1\Big\}\Big).$$

Consider the initial state $(X_0, \emptyset, 0, \epsilon)$. Then the following sequence of transitions is possible in $T(\Delta)$:

$$(X_0, \emptyset, 0, \epsilon) \xrightarrow{a} (X_0 \| X_0, \{Y_0\}, 0, \epsilon) \xrightarrow{b} (X_0, \{Y_0, Y_1\}, 0, \epsilon) \xrightarrow{int}$$

$$(X_0, \{Y_0, Y_1\}, 1, Y_0) \xrightarrow{c} (X_0, \{Y_0, Y_1\}, 1, \epsilon) \xrightarrow{iret} (X_0, \{Y_0, Y_1\}, 0, \epsilon) \xrightarrow{int}$$

$$(X_0, \{Y_0, Y_1\}, 1, Y_1) \xrightarrow{d} (X_0, \{Y_0, Y_1\}, 1, Y_1) \xrightarrow{d} \cdots .$$

It is an easy observation that there is no BPP process strongly bisimilar to the initial state $(X_0, \emptyset, 0, \epsilon)$ of $T(\Delta)$ — we use similar arguments as in Example 4.4.

We remind the reader of the fact that for any BPP system we can find a bisimilar interrupt BPP system simply by disallowing interrupts at all — we define $\mathit{enable} = \emptyset$ in every rule of the BPP system. Hence the class of interrupt BPP is strictly more expressive (w.r.t. strong bisimilarity) than the class of BPP. $\square$

We demonstrate now how to give an alternative semantics in terms of a commutative transition system $T^c(\Delta)$. The idea is that a normal process execution is simulated one-to-one, and the interrupt calls are checked using the relation $\mathcal{E}qv$ — thus there are no actions *int* and *iret*. Let $\Delta = (\Delta_1, \Delta_2)$ be an interrupt BPP system such that $\mathcal{C}onst(\Delta_1) = \{X_0, \ldots, X_{n_1-1}\}$ and $\mathcal{C}onst(\Delta_2) = \{Y_0, \ldots, Y_{n_2-1}\}$. In what follows we denote by $T(\Delta_2)$ the deadlock sensitive transition system generated by the BPP system Δ_2. Since strong bisimilarity in $T(\Delta_2)$ is decidable (Theorem 4.2), we may assume w.l.o.g. that $(Y_i, \Delta_2) \not\sim (Y_j, \Delta_2)$ for all i, j such that $0 \le i < j \le n_2 - 1$. Let $n \overset{\text{def}}{=} \max\{n_1 - 1, n_2 - 1\}$.

We define $T^c(\Delta) \overset{\text{def}}{=} (\mathcal{B}_n^2, \mathcal{A}ct(\Delta_1) \cup \mathcal{A}ct(\Delta_2), \longrightarrow, \mathcal{E}qv)$. The intuition is that in the first component of a state $(M, N) \in \mathcal{B}_n^2$ we remember a BPP expression of normal process execution and in the second component we remember the interrupt vector IV in the following sense: $N(i) = 0$ if $Y_i \notin IV$, and $N(i) > 0$ if $Y_i \in IV$. For $\alpha = (M, N) \in \mathcal{B}_n^2$, let $IV(\alpha) \overset{\text{def}}{=} \{Y_i \mid 0 \leq i \leq n_2 - 1 \ \wedge \ N(i) > 0\}$ and let $cut(\alpha) \overset{\text{def}}{=} (M, N') \in \mathcal{B}_n^2$ such that

$$N'(i) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } N(i) = 0 \\ 1 & \text{if } N(i) > 0 \end{cases}$$

for all $i \in \mathbb{N}_0$. We define

$$\alpha = (M, N) \xrightarrow{a} (M', N') = \alpha'$$

iff $(E, IV(\alpha), 0, \epsilon) \xrightarrow{a} (E', IV(\alpha'), 0, \epsilon)$ such that $a \notin \{int, iret\}$, $\phi(E) = M$, $\phi(E') = M'$, and $cut(\alpha') = \alpha'$. The last condition ($cut(\alpha') = \alpha'$) ensures that $T^c(\Delta)$ becomes image-finite. Finally we define $\mathcal{E}qv$ as such a relation that for states $\alpha, \beta \in \mathcal{B}_n^2$: $(\alpha, \beta) \in \mathcal{E}qv$ iff $IV(\alpha) = IV(\beta)$.

The following property is an immediate consequence of the definition.

Property 4.1. Let $\alpha \in \mathcal{B}_n^2$. Then $(\alpha, T^c(\Delta)) \sim (cut(\alpha), T^c(\Delta))$.

Proposition 4.4. Let $\Delta = (\Delta_1, \Delta_2)$ be an interrupt BPP system. For any $E, F \in \mathcal{P}(\mathcal{C}onst(\Delta_1))$ it holds that $(E, \emptyset, 0, \epsilon) \sim (F, \emptyset, 0, \epsilon)$ in $T(\Delta)$ if and only if $(\phi(E), \emptyset) \sim (\phi(F), \emptyset)$ in $T^c(\Delta)$.

Proof. It is obvious that any transition under a where $a \notin \{int, iret\}$ in $T(\Delta)$ can be simulated naturally in the system $T^c(\Delta)$ and vice versa. An interrupt call in $T(\Delta)$ is checked using the relation $\mathcal{E}qv$ and whenever $(\alpha, \beta) \notin \mathcal{E}qv$ in $T^c(\Delta)$ then we can distinguish the corresponding states in $T(\Delta)$ by an appropriate interrupt call. $\square$

We can now present the following theorem for interrupt BPP.

Theorem 4.4. Given an interrupt BPP process rewrite system Δ and a pair of states $(E, \emptyset, 0, \epsilon)$ and $(F, \emptyset, 0, \epsilon)$ in $T(\Delta)$, it is decidable whether $(E, \emptyset, 0, \epsilon) \sim (F, \emptyset, 0, \epsilon)$.

Proof. Because of Proposition 4.4 it is enough to show that $T^c(\Delta)$ is an ECTS and then we use Theorem 4.1. The validity of conditions (1) – (5) of Definition 4.7 is straightforward. Remains to verify condition (6). Let $\alpha \sim_k \beta$ and $\gamma \in \mathcal{B}_n^2$. We show that $\alpha \oplus \gamma \sim_k \beta \oplus \gamma$. We proceed by induction on k.

Base case: If $k = 0$ then it is enough to show that if $(\alpha, \beta) \in \mathcal{E}qv$ then also $(\alpha \oplus \gamma, \beta \oplus \gamma) \in \mathcal{E}qv$. This is trivially true.

Inductive step: Let $k > 0$ and $\alpha \sim_k \beta$. Of course, $(\alpha \oplus \gamma, \beta \oplus \gamma) \in \mathcal{E}qv$. We will analyse the transitions from $\alpha \oplus \gamma$ only (the arguments for $\beta \oplus \gamma$ are symmetric).

Let $\alpha \oplus \gamma \xrightarrow{a} \kappa$. Then either $\kappa = cut(\alpha' \oplus \gamma)$ and $\alpha \xrightarrow{a} \alpha'$, or $\kappa = cut(\alpha \oplus \gamma')$ and $\gamma \xrightarrow{a} \gamma'$. In the first case, because of our assumption that $\alpha \sim_k \beta$, also

$\beta \xrightarrow{a} \beta'$ such that $\alpha' \sim_{k-1} \beta'$. Hence $\beta \oplus \gamma \xrightarrow{a} cut(\beta' \oplus \gamma)$. Using the induction hypothesis we get $\alpha' \oplus \gamma \sim_{k-1} \beta' \oplus \gamma$ and by Property 4.1 this implies that $cut(\alpha' \oplus \gamma) \sim_{k-1} cut(\beta' \oplus \gamma)$. The second case where $\kappa = cut(\alpha \oplus \gamma')$ is similar. □

Remark 4.4. We used BPP processes for interrupt handling (the system Δ_2). In fact, any process algebra where strong bisimilarity is decidable can be used.

4.4.4 Timed-Arc BPP

In this subsection we shall establish decidability of strong bisimilarity for a time extension of BPP systems, called timed-arc BPP.

It is worth mentioning other positive results for timed BPP. The authors in [20] show that performance equivalence (a version of timed bisimilarity) is decidable in polynomial time for BPP systems where actions have certain time durations. However, their definition of timed BPP does not allow to interpret ordinary BPP systems as timed ones since a duration of an action cannot be equal to 0 and must be strictly positive. Recently Lasota in [115] further extended this result in such a way that his notion of timed BPP subsumes the version with strictly positive time durations and even plain BPP systems. He proved that strong bisimilarity remains decidable. Our notion of time in BPP is, however, different from timed BPP used by Lasota, and it is inspired by a well studied model of timed-arc Petri nets. We define timed-arc BPP as a natural subclass of timed-arc Petri nets where time (age) is associated to tokens and transitions are labelled by time intervals which restrict the age of tokens available for firing a transition (see e.g. [25, 67]). Our definition implies that timed-arc BPP are a strict extension (w.r.t. strong bisimilarity) of ordinary BPP systems, as it is demonstrated later.

First, we introduce labelled timed-arc Petri nets, following definitions from [159] and then we define timed-arc BPP as its subclass where each transition has exactly one input place. A *labelled timed-arc Petri net* (LTAPN) is a tuple $N = (P, T, F, c, L, \lambda, \Sigma)$, where

- P is a finite set of *places*,
- T is a finite set of *transitions* such that $T \cap P = \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*,
- $c : F|_{P\times T} \to \mathbb{N}_0 \times (\mathbb{N}_0 \cup \{\infty\})$ is a *time constraint* on transitions such that for each arc $(p, t) \in F$ holds that $t_1 \leq t_2$ where $c(p, t) = (t_1, t_2)$,
- L is a finite set of *labels (actions)*,
- $\lambda : T \to L$ is *a labelling function*, and
- $\Sigma \subseteq \mathbb{N}_0$ is a recursive set of allowed *time-elapsing steps*.

Let $x \in \mathbb{N}_0$ and $c(p, t) = (t_1, t_2)$. We write $x \in c(p, t)$ whenever $t_1 \leq x \leq t_2$. We also define ${}^\bullet t = \{p \mid (p, t) \in F\}$ and $t^\bullet = \{p \mid (t, p) \in F\}$. A *marking* is a function $M\colon P \to \mathcal{B}$ where $\mathcal{B}$ denotes the set of all finite multisets on $\mathbb{N}_0$. Each place is thus assigned a certain number of tokens, and each token is annotated with a natural number (*age*). Let $x \in \mathcal{B}$ and $a \in \mathbb{N}_0$. We define $x \mathbin{<\!\!+} a$ such that we add the value a to every element of x, i.e., $x \mathbin{<\!\!+} a = \{b + a \mid b \in x\}$.

Let us now define the dynamics of LTAPNs. We introduce two types of transition rules: *firing* of a transition and *time-elapsing*. Let $N = (P, T, F, c, L, \lambda, \Sigma)$ be a LTAPN, M a marking and $t \in T$. We say that t is *enabled* in M iff

$$\forall p \in {}^\bullet t.\ \exists x \in M(p).\ x \in c(p, t).$$

If t is enabled in M then it can be *fired*, producing a marking M' (written $M[t\rangle M'$) such that

$$\forall p \in P.\ M'(p) = \Big(M(p) \smallsetminus C^-(p, t)\Big) \cup C^+(t, p)$$

where C^- and C^+ are chosen to satisfy the following equations (note that there may be more possibilities and that all the operations are on multisets):

$$C^-(p,t) = \begin{cases} \{x\} & \text{such that } x \in M(p) \text{ and } x \in c(p,t) \qquad \text{if } p \in {}^\bullet t \\ \emptyset & \text{otherwise} \end{cases}$$

$$C^+(t,p) = \begin{cases} \{0\} & \text{if } p \in t^\bullet \\ \emptyset & \text{otherwise.} \end{cases}$$

Note that the tokens added to places $t^\bullet$ are of age 0. We define also *time-elapsing* transitions τ_k, $k \in \Sigma$, as follows:

$$M[\tau_k\rangle M' \quad \text{iff} \quad \forall p \in P.\ M'(p) = M(p) \mathbin{<\!\!+} k.$$

Let $N = (P, T, F, c, L, \lambda, \Sigma)$ be a LTAPN. We define the corresponding labelled transition system $T(N) \overset{\text{def}}{=} ([P \to \mathcal{B}], L \cup \{\tau_k \mid k \in \Sigma\}, \longrightarrow, \mathcal{E}qv^u)$, where states are markings of N, actions are labels from L together with symbols for time-elapsing, and $M \xrightarrow{a} M'$ iff either $M[t\rangle M'$ and $a = \lambda(t)$, or $M[\tau_k\rangle M'$ and $a = \tau_k$ for some $k \in \Sigma$. For simplicity we define $\mathcal{E}qv^u$ to be the universal relation.

Definition 4.8 (Timed-arc BPP).
A *timed-arc* BPP is a LTAPN $(P, T, F, c, L, \lambda, \Sigma)$ such that $|{}^\bullet t| = 1$ for all $t \in T$.

Example 4.6. Consider a timed-arc BPP net

$$N \overset{\text{def}}{=} (\{p_1, p_2\}, \{t_1, t_2\}, F, c, \{a, b\}, \lambda, \{1\})$$

where F, c and λ are defined in Figure 4.3. Names of places (circles) are p_1 and p_2 (from left to right) and names of transitions (squares) are t_1 and t_2 (from left to right) such that $\lambda(t_1) = a$ and $\lambda(t_2) = b$. Notice that ${}^\bullet t_1 = \{p_1\}$ and ${}^\bullet t_2 = \{p_2\}$, so the net is indeed a timed-arc BPP net. Let $(\{0\}, \emptyset)$ be an initial marking — since $|P| = 2$ we can identify any marking $M : P \to \mathcal{B}$ with a pair $(M(p_1), M(p_2))$. Now e.g.

$$(\{0\}, \emptyset) \xrightarrow{a} (\{0\}, \{0\}) \xrightarrow{a} (\{0\}, \{0, 0\}) \xrightarrow{b}$$

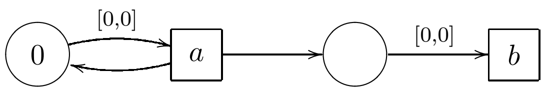


Figure 4.3: A timed-arc BPP net N

$$(\{0\},\{0\}) \xrightarrow{\tau_1} (\{1\},\{1\}) \xrightarrow{\tau_1} (\{2\},\{2\}) \xrightarrow{\tau_1} \dots .$$

Using similar arguments as in Example 4.4, there cannot be any BPP process bisimilar to the initial marking. On the other hand, for any BPP process there is a timed-arc BPP net bisimilar to it — we use the fact that any BPP process can alternatively be viewed as a Petri net where $|{}^\bullet t| = 1$ for every transition t and then we define all the time constrains as $[0,\infty]$ and set $\Sigma \overset{\text{def}}{=} \emptyset$. So the class of timed-arc BPP is strictly more expressive (w.r.t. strong bisimilarity) than the BPP class. □

Assuming a fixed ordering on $P = \{p_1,\dots,p_n\}$, there is a natural one-to-one correspondence between $[P \to \mathcal{B}]$ and $\mathcal{B}^n$. Let $M : P \to \mathcal{B}$ then we define $(N_1,\dots,N_n) \in \mathcal{B}^n$ by $N_i = M(p_i)$ for $1 \leq i \leq n$ and vice versa. In what follows we freely interchange these equivalent notations.

The system $T(N)$ is almost an ECTS. There are only two problems:

- states are not elements from $\mathcal{B}^n_m$ for some fixed $m \in \mathbb{N}_0$ and
- the set of actions can be infinite.

The following arguments show how to avoid these problems.

Definition 4.9. Let $N = (P,T,F,c,L,\lambda,\Sigma)$ be a LTAPN. We define its *maximal guard* $mg(N) \in \mathbb{N}_0$ as the maximal time constraint that appears in N, i.e.,

$$mg(N) \overset{\text{def}}{=} \max\Big(\{t_1,t_2 \mid \exists f \in F|_{P\times T}.\ c(f) = (t_1,t_2)\} \smallsetminus \{\infty\}\Big).$$

Let $M \in [P \to \mathcal{B}]$. We define a *compression* of M, $C_M \in [P \to \mathcal{B}_{mg(N)+1}]$, by

$$C_M(p)(k) \overset{\text{def}}{=} \begin{cases} M(p)(k) & \text{if } k < mg(N)+1 \\ \sum_{i=mg(N)+1}^{\infty} M(p)(i) & \text{if } k = mg(N)+1 \\ 0 & \text{if } k > mg(N)+1. \end{cases}$$

Lemma 4.5. Let $N = (P,T,F,c,L,\lambda,\Sigma)$ be a LTAPN and $M_1, M_2 \in [P \to \mathcal{B}]$. If $C_{M_1} = C_{M_2}$ then $(M_1, T(N)) \sim (M_2, T(N))$.

Proof. It is a routine exercise to verify that

$$R \overset{\text{def}}{=} \{(M_1,M_2) \in [P \to \mathcal{B}] \times [P \to \mathcal{B}] \mid C_{M_1} = C_{M_2}\}$$

is a strong bisimulation. □

Let $N = (P,T,F,c,L,\lambda,\Sigma)$ be a LTAPN. By m we denote the number $mg(N)+1$. We define a commutative transition system $T^c(N) = (\mathcal{B}^n_m, L \cup \{\tau_k \mid$

$k \in \Sigma \ \wedge \ k < m\} \cup T_m, \longrightarrow, \mathcal{E}qv^u)$ where $T_m = \{\tau_m\}$ if there is $k \in \Sigma$ such that $k \geq m$, otherwise $T_m = \emptyset$ (note that the construction of T_m is effective since Σ is a recursive set). We define $M \xrightarrow{a} M'$ for $M, M' \in \mathcal{B}^n_m$ iff either (i) $M[t\rangle M'$ and $a = \lambda(t)$, or (ii) $M[\tau_k\rangle M''$ where $m > k \in \Sigma$ or $\tau_k \in T_m$, such that $M' = C_{M''}$ and $a = \tau_k$.

Proposition 4.5. Let N be a LTAPN and M_1, M_2 a pair of markings in N. Then $(M_1, T(N)) \sim (M_2, T(N))$ if and only if $(C_{M_1}, T^c(N)) \sim (C_{M_2}, T^c(N))$.

Proof. From Lemma 4.5. Also note that in $T(N)$ for any $k \geq m = mg(N) + 1$ holds that if $M \xrightarrow{\tau_m} M'$ and $M \xrightarrow{\tau_k} M''$, then $C_{M'} = C_{M''}$. □

We are now ready to show decidability of strong bisimilarity for timed-arc BPP nets.

Theorem 4.5. Given a timed-arc BPP net $N = (P, T, F, c, L, \lambda, \Sigma)$ and a pair of markings M_1, M_2 on N, it is decidable whether $(M_1, T(N)) \sim (M_2, T(N))$.

Proof. By Proposition 4.5 it is enough to prove that $T^c(N)$ is an ECTS. To verify conditions (1) – (5) of Definition 4.7 is easy. Let us now examine the condition (6). Let $\alpha \sim_k \beta$ and $\gamma \in \mathcal{B}^n_m$. We show that $\alpha \oplus \gamma \sim_k \beta \oplus \gamma$. We proceed by induction on k.
Base case: If $k = 0$ then it is enough to show that if $(\alpha, \beta) \in \mathcal{E}qv^u$ then also $(\alpha \oplus \gamma, \beta \oplus \gamma) \in \mathcal{E}qv^u$. This is trivially true.
Inductive step: Let $k > 0$ and $\alpha \sim_k \beta$. Of course, $(\alpha \oplus \gamma, \beta \oplus \gamma) \in \mathcal{E}qv^u$. We will analyse the transitions from $\alpha \oplus \gamma$ only (the arguments for $\beta \oplus \gamma$ are symmetric).

Let $\alpha \oplus \gamma \xrightarrow{a} \kappa$ such that $a \neq \tau_l$ for $l \in \mathbb{N}_0$. Then either $\kappa = \alpha' \oplus \gamma$ and $\alpha \xrightarrow{a} \alpha'$, or $\kappa = \alpha \oplus \gamma'$ and $\gamma \xrightarrow{a} \gamma'$. (Note that there are not more possibilities since $|{}^\bullet t| = 1$ for any $t \in T$.) In the first case, because of our assumption that $\alpha \sim_k \beta$, also $\beta \xrightarrow{a} \beta'$ such that $\alpha' \sim_{k-1} \beta'$. Hence $\beta \oplus \gamma \xrightarrow{a} \beta' \oplus \gamma$. Using the induction hypothesis we get $\alpha' \oplus \gamma \sim_{k-1} \beta' \oplus \gamma$. The second case where $\kappa = \alpha \oplus \gamma'$ is similar.

Let $\alpha \oplus \gamma \xrightarrow{\tau_l} \alpha' \oplus \gamma'$ for some l, $m \geq l$. Of course, $\alpha \xrightarrow{\tau_l} \alpha'$ and since $\alpha \sim_k \beta$ also $\beta \xrightarrow{\tau_l} \beta'$ such that $\alpha' \sim_{k-1} \beta'$. Hence $\beta \oplus \gamma \xrightarrow{\tau_l} \beta' \oplus \gamma'$ and using the induction hypothesis we get $\alpha' \oplus \gamma' \sim_{k-1} \beta' \oplus \gamma'$. □

Remark 4.5. It remains an open problem whether strong bisimilarity is decidable for timed-arc BPP with continuous time, i.e., if we allow e.g. $\Sigma = \mathbb{R}^+_0$. On the other hand, if we keep the discrete time setting and consider *distributed* timed-arc BPP nets, bisimilarity remains decidable. For the definition of distributed timed-arc Petri nets see [140].

4.5 Concluding Remarks

We suggested a subclass of labelled transition systems called effective commutative transition systems (ECTS) where strong bisimilarity is decidable, and we showed that semantics of many extensions of BPP process algebra can be defined within the ECTS class. This approach seems to be feasible also for other natural extensions of BPP: the crucial condition to be satisfied is probably (6), saying that $\sim_k$ are congruences. This condition fails e.g. for Petri nets, and indeed strong bisimilarity becomes undecidable [90].

Decidability of weak bisimilarity of BPP is still a well known open problem, however, there is a promising approach to the problem recently introduced by Jančar [92]. In case of weak bisimilarity for BPP the problematic condition is (5), stating that the transition system is image-finite, which is not the case for weak bisimilarity. Nevertheless, we can still instead of potentially infinite set of successors $\mathsf{next}(\alpha, a)$ examine only its finite subset such that soundness and completeness of the tableau system is preserved. This possibility was exploited by Stirling in [187] for weak bisimilarity of normed BPP, however, with additional technical restrictions. To design finite subsets of $\mathsf{next}(\alpha, a)$ preserving soundness and completeness even in the general case might be a reasonable way to attack this problem.

4.6 Bibliographical Remarks

The content of this chapter is based on the paper "Note on the Tableau Technique for Commutative Transition Systems" [175] and on the technical report [170]. The notion of timed-arc Petri nets (including several ideas for more advanced time distribution over the net) was further studied in [141] and [140] as a joint research with Mogens Nielsen and Vladimiro Sassone. This work is, however, beyond the focus of this thesis and is not included.

Chapter 5

Lower Bounds for Weak Bisimilarity

In this chapter we address the problems of weak bisimilarity and regularity checking for BPA and BPP. We prove several complexity lower bounds for these problems with a particular focus on deriving results valid for normed processes. Some ideas described here will be further developed in Chapter 6 in order to adopt the techniques to the case of strong bisimilarity and strong regularity. However, during this process we have to sacrifice the validity of the lower bounds for normed systems. This should justify the inclusion of the hardness results for weak bisimilarity and regularity into this chapter — the presented lower bounds hold even for normed processes.

5.1 Motivation

Despite the fact that BPA and BPP are quite restricted classes, it appears nontrivial to introduce completeness of weak bisimilarity and regularity problems in some complexity class, or even to show decidability of these problems — decidability issues are still open[1]. Weak bisimilarity is known to be semi-decidable for BPP [57] and there are partial results by Hirshfeld [72] showing decidability of weak bisimilarity for restricted classes of totally normed BPA and BPP. Recently, Stirling [187] showed that weak bisimilarity is decidable for a large subclass of normed BPP. Unfortunately, the result does not immediately imply decidability of weak bisimilarity for the whole class of normed BPP.

There has also been some effort to provide at least hardness results. A PSPACE lower bound for weak bisimilarity of unnormed BPA and NP lower bound for weak bisimilarity of totally normed BPP were demonstrated by Stříbrná [189]. In case of BPA Stříbrná used a reduction from the totality problem for finite nondeterministic automata and hence unnormed process constants were needed. Mayr recently achieved a result, saying that weak bisimilarity of (unnormed) BPP is Π_2^P-hard [124] (in the polynomial hierarchy). In the same paper he also provided a construction showing that weak regularity of BPP is Π_2^P-hard. No lower bound was previously established for weak regularity of BPA.

[1]Very recently Jančar indicated [92] that he invented a new technique for strong bisimilarity of BPP which might be possibly used to prove also decidability of weak bisimilarity.

This chapter aims to contribute to the understanding of complexity issues of weak bisimilarity checking on simple process algebras. In Section 5.2 we improve Mayr's Π_2^P lower bound for weak bisimilarity of BPP to PSPACE, and show that this hardness result is valid even for the restricted subclass of *normed* BPP. The proof is by reduction from a PSPACE-complete problem of *quantified satisfiability* (also called quantified boolean formula, see e.g. [145]) which appears to be very useful for showing complexity lower bounds of bisimilarity. This problem will also be used in Chapter 6 in the context of strong bisimilarity. Concerning hardness results for BPA we announced [173] that weak bisimilarity of normed (and regular) BPA is PSPACE-hard. However, we do not include the proof in this thesis since very recently Mayr improved the result for normed (but non-regular) BPA even to EXPTIME [128].

The hardness results for regularity checking are usually derived from the lower bounds of bisimilarity problems. As first noticed by Mayr in [124], it is possible to reduce weak bisimilarity between a pair of regular BPP processes to weak regularity of BPP. In Section 5.3 we explicitly formulate this fact for BPP and moreover (using similar ideas as from [124]) we further extend this reduction also to BPA. An important observation is that these reductions preserve normedness. Hence weak regularity problems for normed BPP and normed BPA become PSPACE-hard. Mayr recently showed that weak regularity of (unnormed) BPA is even EXPTIME-hard [128].

The reader may wonder why we show PSPACE-hardness results for weak bisimilarity and regularity of BPP, and in Chapter 6 we further strengthen the results to the case of strong bisimilarity and regularity. The reason is that the techniques used in Chapter 6 require introduction of unnormed process constants and hence the results cannot be transfered to weak bisimilarity and regularity of normed processes. Moreover, it is unlikely (unless P=PSPACE) that strong bisimilarity and strong regularity of normed BPA and BPP are PSPACE-hard because there are polynomial time algorithms solving these problems, both for normed BPA [74, 106] and normed BPP [75, 106].

5.2 Weak Bisimilarity of Normed BPP

In this section we show that weak bisimilarity of normed BPP is PSPACE-hard. We prove it by reduction from the problem of quantified satisfiability (QSAT), which is known to be PSPACE-complete [145].

Problem:	<u>QSAT</u>
Instance:	A natural number n and a Boolean formula ϕ in conjunctive normal form with Boolean variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$.
Question:	Is $\forall x_1 \exists y_1 \forall x_2 \exists y_2 \ldots \forall x_n \exists y_n . \phi$ true?

A *literal* is a variable or the negation of a variable. Let

$$C \equiv \forall x_1 \exists y_1 \forall x_2 \exists y_2 \ldots \forall x_n \exists y_n . C_1 \wedge C_2 \wedge \ldots \wedge C_k$$

be an instance of QSAT, where each *clause* C_j, $1 \leq j \leq k$, is a disjunction of literals. We will define BPP processes (P_1, Δ) and (P_2, Δ), where

$$\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{Q_1, \dots, Q_k, X_1, \dots, X_n, Y_1, \dots, Y_n\}$$

and

$$\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{q_1, \dots, q_k, x_1, \dots, x_n, \overline{x}_1, \dots, \overline{x}_n, y\}.$$

Our aim is to define Δ such that $(P_1, \Delta) \approx (P_2, \Delta)$ if and only if C is true. For each i, $1 \leq i \leq n$, let

α_i be a parallel composition $Q_{i_1} \| Q_{i_2} \| \cdots \| Q_{i_\ell}$ such that
$1 \leq i_1 < i_2 < \cdots < i_\ell \leq k$ and
$C_{i_1}, C_{i_2}, \dots, C_{i_\ell}$ are all the clauses where x_i occurs positively,

$\overline{\alpha_i}$ be a parallel composition $Q_{i_1} \| Q_{i_2} \| \cdots \| Q_{i_\ell}$ such that
$1 \leq i_1 < i_2 < \cdots < i_\ell \leq k$ and
$C_{i_1}, C_{i_2}, \dots, C_{i_\ell}$ are all the clauses where x_i occurs negatively,

β_i be a parallel composition $Q_{i_1} \| Q_{i_2} \| \cdots \| Q_{i_\ell}$ such that
$1 \leq i_1 < i_2 < \cdots < i_\ell \leq k$ and
$C_{i_1}, C_{i_2}, \dots, C_{i_\ell}$ are all the clauses where y_i occurs positively, and

$\overline{\beta_i}$ be a parallel composition $Q_{i_1} \| Q_{i_2} \| \cdots \| Q_{i_\ell}$ such that
$1 \leq i_1 < i_2 < \cdots < i_\ell \leq k$ and
$C_{i_1}, C_{i_2}, \dots, C_{i_\ell}$ are all the clauses where y_i occurs negatively.

Example 5.1. Let us consider a quantified boolean formula

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 .\ (x_1 \vee \neg y_1 \vee y_2) \wedge (\neg x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee y_2 \vee \neg y_2)$$

where $n = 2$, $k = 3$, $C_1 = x_1 \vee \neg y_1 \vee y_2$, $C_2 = \neg x_1 \vee y_1 \vee y_2$ and $C_3 = x_1 \vee y_1 \vee y_2 \vee \neg y_2$. Then

$$\begin{array}{llll} \alpha_1 = Q_1 \| Q_3 & \overline{\alpha_1} = Q_2 & \beta_1 = Q_2 \| Q_3 & \overline{\beta_1} = Q_1 \\ \alpha_2 = \epsilon & \overline{\alpha_2} = \epsilon & \beta_2 = Q_1 \| Q_2 \| Q_3 & \overline{\beta_2} = Q_3. \end{array}$$

The intuition is that α_i contains all the clauses that become satisfied when x_i is set to true, and $\overline{\alpha_i}$ contains all the clauses that become satisfied when x_i is set to false. Similarly for β_i and $\overline{\beta_i}$. □

The set of transition rules Δ is given by

$$\begin{array}{lll}
X_i \xrightarrow{x_i} Y_i \| \alpha_i & X_i \xrightarrow{\overline{x}_i} Y_i \| \overline{\alpha_i} & \text{for } 1 \leq i \leq n \\
& & \\
Y_i \xrightarrow{y} X_{i+1} \| \beta_i & Y_i \xrightarrow{y} X_{i+1} \| \overline{\beta_i} & \text{for } 1 \leq i \leq n-1 \\
Y_n \xrightarrow{y} \beta_n & Y_n \xrightarrow{y} \overline{\beta_n} & \\
& & \\
X_i \xrightarrow{q_j} X_i & Y_i \xrightarrow{q_j} Y_i & \text{for } 1 \leq i \leq n \text{ and } 1 \leq j \leq k \\
& & \\
Q_j \xrightarrow{q_j} Q_j & Q_j \xrightarrow{\tau} \epsilon & \text{for } 1 \leq j \leq k.
\end{array}$$

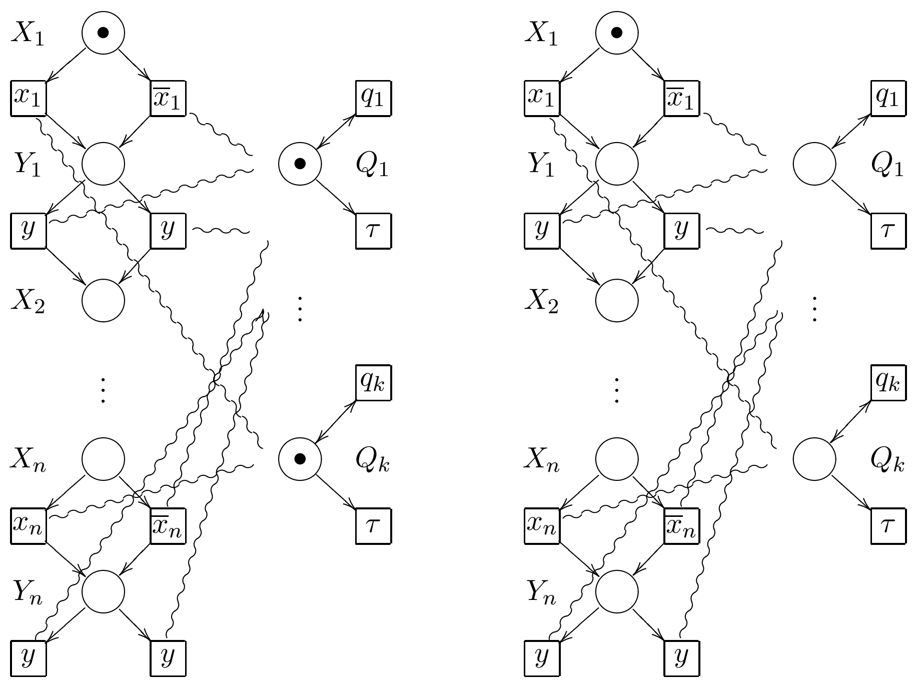


Figure 5.1: The processes (P_1, Δ) and (P_2, Δ) as Petri nets

Finally, let

$$P_1 \overset{\text{def}}{=} X_1 \| Q_1 \| Q_2 \| \ldots \| Q_k \qquad \text{and} \qquad P_2 \overset{\text{def}}{=} X_1.$$

We can see the processes P_1 and P_2 using Petri net notation in Figure 5.1. This figure is only illustrative, and some transitions, namely $X_i \xrightarrow{q_j} X_i$ and $Y_i \xrightarrow{q_j} Y_i$ for $1 \leq i \leq n$, $1 \leq j \leq k$ are missing. The curly lines stand for the corresponding sets of arrows for α_i, $\overline{\alpha_i}$, β_i resp. $\overline{\beta_i}$. The intuition is that the attacker will be forced to play only in the process P_1 and if C is true then the defender will have the possibility to add all the process constants $\{Q_1, \ldots, Q_k\}$.

Let γ be a parallel composition of elements from $\mathcal{C}onst(\Delta)$. We define the set of process constants that occur in γ as $\mathsf{set}(\gamma) \overset{\text{def}}{=} \{X \in \mathcal{C}onst(\Delta) \mid X \text{ occurs in } \gamma\}$ and we also define $\mathsf{set}_Q(\gamma) \overset{\text{def}}{=} \mathsf{set}(\gamma) \cap \{Q_1, \ldots, Q_k\}$. The following proposition is an immediate consequence of the definition of Δ.

Proposition 5.1. Let γ and γ' be parallel compositions of some process constants from $\{Q_1, \ldots, Q_k\}$. Then $\mathsf{set}_Q(\gamma) = \mathsf{set}_Q(\gamma')$ if and only if $(\gamma, \Delta) \approx (\gamma', \Delta)$.

We need to show that C is true if and only if $(P_1, \Delta) \approx (P_2, \Delta)$.

Lemma 5.1. If $(P_1, \Delta) \approx (P_2, \Delta)$ then C is true.

Proof. We show that $(P_1, \Delta) \not\approx (P_2, \Delta)$ assuming that C is false. If C is false then $C' \stackrel{\text{def}}{=} \exists x_1 \forall y_1 \exists x_2 \forall y_2 \ldots \exists x_n \forall y_n . \neg(C_1 \wedge C_2 \wedge \ldots \wedge C_k)$ is true and from this we claim that the attacker has a winning strategy in the bisimulation game for (P_1, Δ) and (P_2, Δ). The attacker plays only in the process P_1 (without using τ actions) performing the following sequence of actions

$$\widetilde{x}_1, y, \widetilde{x}_2, y, \ldots, \widetilde{x}_n, y$$

where $\widetilde{x}_i$, $1 \leq i \leq n$, corresponds to either x_i or $\overline{x}_i$, depending on the truth values for which the formula C' is true. It does not matter how the choice of the rule for the action y is done. The defender can only respond by performing the same actions $\widetilde{x}_1, y, \widetilde{x}_2, y, \ldots, \widetilde{x}_n, y$ (possibly using also some τ actions). The actions $\widetilde{x}_1, \ldots, \widetilde{x}_n$ are forced. For the action y there are always two possibilities, corresponding to assigning a truth value for y_i, $1 \leq i \leq n$. Finally the processes P_1 and P_2 are in states P_1' and P_2', respectively, such that $\mathsf{set}(P_1') = \{Q_1, \ldots, Q_k\}$ and $\mathsf{set}(P_2') \subseteq \{Q_1, \ldots, Q_k\}$. Since we assume that C' is true, the attacker can ensure that there is a clause C_j, $1 \leq j \leq k$, which is not satisfied. Hence $Q_j \notin \mathsf{set}(P_2')$ and P_2' cannot perform q_j. However, q_j is enabled in P_1' and thus the attacker has a winning strategy. This implies that $(P_1, \Delta) \not\approx (P_2, \Delta)$. □

For the proof of the opposite direction let us first observe the following property of (P_1, Δ) and (P_2, Δ). Let δ be some state such that $\mathsf{set}(\delta) \cap \{Q_1, \ldots, Q_k\} = \emptyset$ and let γ and γ' be parallel compositions of some process constants from $\{Q_1, \ldots, Q_k\}$ satisfying the condition that $\mathsf{set}_Q(\gamma) \supseteq \mathsf{set}_Q(\gamma')$. Let us consider the processes $\delta \| \gamma$ and $\delta \| \gamma'$. Whenever the attacker chooses any move in the second one, the defender has an answer which makes these two processes weakly bisimilar (using τ actions to eliminate the extra process constants Q_j from the first process and then by Proposition 5.1). We are now ready to prove the following lemma.

Lemma 5.2. If C is true then $(P_1, \Delta) \approx (P_2, \Delta)$.

Proof. Let P_1' and P_2' denote successors of P_1 and P_2, respectively, in the bisimulation game. The defender's strategy is to satisfy the following conditions during the game:

- $\mathsf{set}_Q(P_1') \supseteq \mathsf{set}_Q(P_2')$ and
- never delete (using τ actions) any process constant Q_j, $1 \leq j \leq k$, in the process P_2', unless it is necessary for satisfying the first condition.

Of course these conditions are true at the beginning of the game. Using the argument above this lemma, we can see that whenever the attacker makes a move in the process P_2', he immediately loses since the defender can make the resulting processes weakly bisimilar. This means that the only possible winning strategy for the attacker is to keep playing in P_1'. However, now the defender can always fulfill the conditions of his strategy. On a move under x_i resp. $\overline{x}_i$ there is only one possible response for the defender. Whenever the attacker makes a move under y, the defender chooses one of the rules $Y_i \xrightarrow{y} X_{i+1} \| \beta_i$

and $Y_i \xrightarrow{y} X_{i+1}\|\overline{\beta_i}$, such that the formula $\forall x_{i+1}\exists y_{i+1}\ldots\forall x_n\exists y_n.C_1\wedge\ldots\wedge C_k$ is still true. Since we have the rules $X_i \xrightarrow{q_j} X_i$ and $Y_i \xrightarrow{q_j} Y_i$ for any i,j such that $1\le i\le n$ and $1\le j\le k$, the only possibility for the attacker to win is to perform a sequence

$$\widetilde{x}_1, y, \widetilde{x}_2, y, \ldots, \widetilde{x}_n, y$$

possibly including also some τ actions and then reach a state P_1' such that $\mathsf{set}(P_1') \subseteq \{Q_1,\ldots,Q_k\}$. Since C is true the defender can always reach a corresponding state P_2' where $\mathsf{set}(P_2') = \{Q_1,\ldots,Q_k\}$ and then using τ actions he ensures that $\mathsf{set}(P_1') = \mathsf{set}(P_2')$. Hence (using Proposition 5.1) the attacker loses again. This means that the defender has a winning strategy and $(P_1,\Delta)\approx(P_2,\Delta)$. □

Theorem 5.1. Weak bisimilarity of normed BPP is PSPACE-hard.

Proof. Observe that all the process constants in Δ are normed and that the reduction is in polynomial time. The theorem is then an immediate consequence of Lemma 5.1 and Lemma 5.2. □

Remark 5.1. Theorem 5.1 can be easily extended to 1-safe Petri nets where *each transition has exactly one input place* (for the definition of 1-safe Petri nets see e.g. [98]). It is enough to introduce for each $\alpha_i/\overline{\alpha_i}$ and $\beta_i/\overline{\beta_i}$, $1\le i\le n$, a new set of process constants $\{Q_1,\ldots,Q_k\}$ to ensure that in each reachable marking there is at most one token in every place. Related results about 1-safe Petri nets can be found in [98].

5.3 Weak Regularity of Normed BPA and BPP

In this section we analyze the problem of weak regularity of BPP and BPA processes. Mayr proved that weak regularity of BPP is Π_2^P-hard [124] by giving a reduction from the weak bisimilarity problem between a pair of special processes with finitely many reachable states. It can be easily seen that his proof works also for a general pair of weakly regular processes and moreover it preserves normedness.

Theorem 5.2 ([124]). Let (P_1,Δ) and (P_2,Δ) be weakly regular BPP processes. We can construct in polynomial time a BPP process (P,Δ') such that

$$(P_1,\Delta)\approx(P_2,\Delta) \text{ if and only if } (P,\Delta') \text{ is weakly regular.}$$

Moreover, if (P_1,Δ) and (P_2,Δ) are normed, so is (P,Δ').

Observe that the processes (P_1,Δ) and (P_2,Δ) from the proof of PSPACE-hardness of weak bisimilarity for BPP (Theorem 5.1) are weakly regular (they are even finite-state processes). Moreover the processes are normed. This gives the following theorem.

Theorem 5.3. Weak regularity of normed BPP is PSPACE-hard.

Let us now consider the problem of weak regularity for BPA. There was no lower bound known before. We show that there is a reduction from weak bisimilarity of regular BPA to weak regularity. The idea of the proof is similar to the case of BPP mentioned above from [124].

Theorem 5.4. Let (P_1, Δ) and (P_2, Δ) be weakly regular BPA processes. We can construct in polynomial time a BPA process (P, Δ') such that

$$(P_1, \Delta) \approx (P_2, \Delta) \text{ if and only if } (P, \Delta') \text{ is weakly regular.}$$

Moreover, if (P_1, Δ) and (P_2, Δ) are normed, so is (P, Δ').

Proof. Assume that (P_1, Δ) and (P_2, Δ) are weakly regular BPA processes. We construct a BPA process (P, Δ') with

$$\mathcal{C}onst(\Delta') \stackrel{\text{def}}{=} \mathcal{C}onst(\Delta) \cup \{A, B, C, B_1, B_2\}$$

and

$$\mathcal{A}ct(\Delta') \stackrel{\text{def}}{=} \mathcal{A}ct(\Delta) \cup \{a\}$$

where A, B, C, B_1, B_2 are new process constants and a is a new action. Then $\Delta' \stackrel{\text{def}}{=} \Delta \cup \Delta^1 \cup \Delta^2$, where Δ^1 and Δ^2 are defined as follows. The set of transition rules Δ^1 is given by

$$\begin{array}{ll} A \xrightarrow{a} A.B & A \xrightarrow{\tau} \epsilon \\ B \xrightarrow{a} \epsilon & B \xrightarrow{\tau} \epsilon \\ & \\ C \xrightarrow{a} B_1 & C \xrightarrow{a} P_1 \\ B_1 \xrightarrow{a} B_1 & B_1 \xrightarrow{a} P_1 \end{array}$$

and Δ^2 is given by

$$\begin{array}{ll} C \xrightarrow{a} B_2 & C \xrightarrow{a} P_2 \\ B_2 \xrightarrow{a} B_2 & B_2 \xrightarrow{a} P_2. \end{array}$$

Let $P \stackrel{\text{def}}{=} A.C$. Observe that if (P_1, Δ) and (P_2, Δ) are normed, so is (P, Δ'). We show now that our reduction is correct.

Lemma 5.3. If $(P_1, \Delta) \not\approx (P_2, \Delta)$ then (P, Δ') is not weakly regular.

Proof. Suppose that $(P_1, \Delta) \not\approx (P_2, \Delta)$. Then we demonstrate that there are infinitely many weakly nonbisimilar states reachable from P. Let us consider $B^i.C$ for any natural number i. Of course $P \longrightarrow^* B^i.C$ and we claim that $(B^i.C, \Delta') \not\approx (B^j.C, \Delta')$ for any $i \neq j$. Without loss of generality assume that $i < j$. The attacker has the following winning strategy (playing only in the second process — see Figure 5.2). He performs a sequence of j actions a in $B^j.C$, thus reaching C. Since B^i cannot do this sequence, the defender has to reach C eventually (let us say after i' rounds where $i' \leq i$). As neither P_1 nor P_2 can perform a, he has only two choices when responding to the action a. He can play either $C \xrightarrow{a} B_1$ or $C \xrightarrow{a} B_2$. Assume that he chooses B_1 (the

$$\begin{tikzcd}
B^i.C \arrow[d, Rightarrow, "a^{i'}"] & \approx^? & B^j.C \arrow[d, "a^{i'}"] \\
C \arrow[d, "a"] & & B^{j-i'}.C \arrow[d, "a"] \\
B_1 \arrow[d, "a^{j-i'-1}"] & & B^{j-i'-1}.C \arrow[d, "a^{j-i'-1}"] \\
B_1 \arrow[d, "a"] & & C \arrow[d, "a"] \\
P_1 & \not\approx & P_2
\end{tikzcd}$$

Figure 5.2: Winning strategy for the attacker ($i < j$)

other case is symmetric). Now the defender's only possibility is to stay in B_1 for another $a^{j-i'-1}$ moves of the attacker. After the attacker has reached C (in the second process), he chooses to go to P_2 in the next round. If the defender stays in B_1 he loses immediately and if he moves to P_1 he loses as well, since $(P_1, \Delta') \not\approx (P_2, \Delta')$. $\square$

Lemma 5.4. If $(P_1, \Delta) \approx (P_2, \Delta)$ then (P, Δ') is weakly regular.

Proof. Assume that $(P_1, \Delta) \approx (P_2, \Delta)$ which implies that $(P, \Delta') \approx (P, \Delta'')$ where $\Delta'' = \Delta' \smallsetminus \Delta^2$. Notice that (B_1, Δ'') is weakly regular, so it is enough to show that $(A.C, \Delta'') \approx (B_1, \Delta'')$. Obviously $(C, \Delta'') \approx (B_1, \Delta'')$ which implies that for any $n \geq 0$, $(B^n.C, \Delta'') \approx (B_1, \Delta'')$ since $B^n \xrightarrow{\tau^*} \epsilon$.

This gives that $(A.B^n.C, \Delta'') \approx (B_1, \Delta'')$ for any $n \geq 0$, which in particular means that $(A.C, \Delta'') \approx (B_1, \Delta'')$. $\square$

Theorem 5.4 is an immediate consequence of Lemma 5.3 and Lemma 5.4. $\square$

In the paper by Stříbrná [189] it is shown (Theorem 2.5) that weak bisimilarity of totally normed BPA is NP-hard. The proof is by reduction from a variant of the bin-packing (knapsack) problem and the processes in this proof have finitely many reachable states (and so they are weakly regular). Thus we can use Theorem 5.4 to obtain the following result.

Theorem 5.5. Weak regularity of normed BPA is NP-hard.

Remark 5.2. We announced [173] that weak bisimilarity of normed BPA is PSPACE-hard. Since a reduction from QSAT can be given such that the processes are weakly regular, we get that weak regularity of normed BPA is also PSPACE-hard. Recently Mayr achieved EXPTIME lower bound for weak bisimilarity of regular (but unnormed) BPA [128], which means that the problem of weak regularity for unnormed BPA is EXPTIME-hard.

5.4 Concluding Remarks

The results proved in this chapter indicate that weak bisimilarity and regularity checking problems for normed process algebras are significantly (unless P=PSPACE) more difficult than those for strong bisimilarity and regularity.

We proved that weak bisimilarity of normed BPP is PSPACE-hard by describing a reduction from the problem of quantified satisfiability (QSAT). By another reduction from QSAT (using different techniques) we can prove [173] that the weak bisimilarity problem is PSPACE-hard also for normed BPA. Nevertheless, Mayr recently improved the result to EXPTIME [128] and hence we did not include our PSPACE-hardness proof into this thesis.

In the following chapter we will further extend the ideas of reducing QSAT to the strong bisimilarity problems for BPA and BPP. We will develop so called *existential quantification technique* which enables the defender to choose truth values of variables during the process of assignment generation. The crucial difference is that this technique for strong bisimilarity requires unnormed process constants whereas in case of weak bisimilarity we showed that normed processes are sufficient for obtaining PSPACE-hardness.

Concerning weak regularity questions, we explicitly formulated a reduction from weak bisimilarity of regular processes to weak regularity. The reduction idea for BPP is due to Mayr [124]. We modified this idea and showed that it works also for BPA. An important observation is that the reductions preserve normedness.

This allows to transfer the hardness results for weak bisimilarity checking to weak regularity checking, provided that the involved processes are weakly regular.

5.5 Bibliographical Remarks

This chapter is based on the paper "Complexity of Weak Bisimilarity and Regularity for BPA and BPP" [167]. The paper is available also as a technical report [168] and an extended journal version [174] will appear in Mathematical Structures in Computer Science. The journal version also contains techniques for proving DP-hardness of weak bisimilarity for BPA and strong bisimilarity for BPP. These results are, however, not included because in Chapter 6 we further improve the DP lower bounds to PSPACE.

Chapter 6

Lower Bounds for Strong Bisimilarity

In this chapter we study strong bisimilarity problems for simple process algebras. In particular, we show PSPACE-hardness of the following problems: (i) strong bisimilarity of basic parallel processes (BPP), (ii) strong bisimilarity of basic process algebra (BPA), (iii) strong regularity of BPP, and (iv) strong regularity of BPA. A novel technique called existential quantification is developed for this purpose.

We also demonstrate NL-hardness of strong regularity problems for the normed subclasses of BPP and BPA, thus introducing NL-completeness of the problems.

6.1 Motivation

It is a well known fact that unlike all other equivalences in van Glabbeek's linear/branching time spectrum (see [64, 82]), strong bisimilarity is decidable for BPA [46] and BPP [43]. This challenging phenomenon was a motivation for further investigation of strong bisimilarity in the class of simple process algebras. Restricted classes of normed processes were studied, with the surprising results that even though language equivalence is still undecidable for normed BPA [80] and BPP [82], strong bisimilarity becomes decidable even in polynomial time [74, 75]. The situation is, however, different for the unrestricted classes of simple process algebras.

Despite the fact that strong bisimilarity of BPA is decidable in 2-EXPTIME [35], it is still an open problem whether there exists an elementary decision algorithm for BPP[1]. The conjecture that strong bisimilarity of unnormed BPP is decidable (like in the normed case) in polynomial time was only recently proved false (unless P=NP) by Mayr. He showed that strong bisimilarity of BPP is co-NP-hard [124]. No nontrivial lower bound was known for unnormed BPA.

We will improve Mayr's co-NP lower bound for BPP and show that the complexity of bisimilarity checking of BPA is indeed different (unless P=PSPACE) from the case of normed BPA by demonstrating that strong bisimilarity of

[1]Very recently Jančar indicated [92] that he invented a new technique for strong bisimilarity of BPP which will probably imply the containment of the problem in PSPACE.

BPA and BPP is PSPACE-hard. In order to show that we describe polynomial time reductions from the quantified satisfiability (QSAT) problem to the strong bisimilarity checking problems of BPA and BPP. Given an instance C of QSAT, we construct a pair of BPA (BPP) processes P_1 and P_2 such that P_1 and P_2 are strongly bisimilar if and only if C is true.

Except for the PSPACE-hardness results, a new contribution achieved in these reductions is a general technique which enables to imitate a generation of quantified assignments of boolean formulas in the context of bisimulation games of an attacker and a defender. While the truth value of a variable prefixed by the universal quantifier is being chosen, the attacker has the possibility to decide between two alternatives in the continuation of the bisimulation game. On the other hand, while choosing the truth value of an existentially quantified variable, the defender can force the attacker to continue the bisimulation game according to his decision — we call this technique *existential quantification technique*. (Similar ideas (not explicitly formulated) appeared independently due to Jančar in the context of high undecidability of weak bisimilarity for Petri nets [89].) Satisfied clauses of a given formula are remembered by means of process constants that are present in the current states of BPA and BPP systems. After the whole assignment of boolean variables was generated, the attacker can make a final check whether all clauses are indeed satisfied. This is easier to verify for BPP because we have a parallel (and thus simultaneous) access to all process constants contained in the current state. To achieve the same result for BPA, we will have to encode satisfied clauses in a unary way.

Another decidability problem that has attracted much attention is that of regularity checking. Strong regularity checking is decidable in 2-EXPTIME for BPA [36, 35] and in polynomial time for normed BPA and BPP [106] where it coincides with boundedness. Decidability of strong regularity for BPP follows from the fact that the problem is decidable even for Petri nets [93], a proper superclass of BPP. However, no elementary upper bound has been established so far. It is known that strong regularity is co-NP-hard for BPP [124] and no hardness result was available for BPA. We will describe polynomial time reductions from strong bisimilarity of regular BPA (BPP) processes to strong regularity checking of BPA (BPP), using Mayr's idea from [124]. By applying our PSPACE-hardness of strong bisimilarity for BPA and BPP, and by the fact that the involved processes are strongly regular, we can conclude that strong regularity of BPA and BPP are PSPACE-hard problems.

Finally, we will investigate the complexity of regularity checking problems for normed BPA and BPP and show their NL-completeness.

6.2 Existential Quantification Technique

In this section we explain the basic idea of the PSPACE-hardness proofs presented in this chapter, namely the existential quantification technique. Our aim is to make the rewrite rules defined in the following sections more readable, by demonstrating a general pattern of existential quantification used heavily (with small modifications) later on.

$$\begin{tikzcd}
 & & X \arrow[dl, "a"'] \arrow[drr, "a"] \arrow[drrrr, "a"] & & & & \\
 & Y^{\texttt{choice}} \arrow[dl, "\texttt{one}"'] \arrow[dr, "\texttt{two}"] & & Y^{\texttt{one}} \arrow[dl, dotted, no head] \arrow[dr, dotted, no head] & & Y^{\texttt{two}} \arrow[dl, dotted, no head] \arrow[dr, dotted, no head] & \\
Z \oplus \alpha_{\texttt{one}} & & Z \oplus \alpha_{\texttt{two}} & & {} & & {}
\end{tikzcd}
\qquad
\begin{tikzcd}
 & & X' \arrow[dl, "a"'] \arrow[dr, "a"] & & \\
 & Y^{\texttt{one}} \arrow[dl, "\texttt{one}"'] \arrow[d, "\texttt{two}"] & & Y^{\texttt{two}} \arrow[d, "\texttt{two}"'] \arrow[dr, "\texttt{one}"] & \\
Z' \oplus \alpha_{\texttt{one}} & Z \oplus \alpha_{\texttt{two}} & & Z' \oplus \alpha_{\texttt{two}} & Z \oplus \alpha_{\texttt{one}}
\end{tikzcd}$$

Figure 6.1: Processes (X, Δ) and (X', Δ)

The existential quantification technique in its simplest form can be explained as follows: the defender can force the attacker to perform a certain move by threatening to enter bisimilar states otherwise. Take a look at processes s and t such that

$$s \xrightarrow{a} s', \;\; t \xrightarrow{a} t', \;\; \text{and} \;\; t \xrightarrow{a} t'',$$

where s' and t' are assumed to be bisimilar.

$$\begin{tikzcd}
 & s \arrow[dl, "a"'] \\
s' &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
 & t \arrow[dl, "a"'] \arrow[dr, "a"] & \\
t' & & t''
\end{tikzcd}$$

In the bisimulation game played from s and t, the only option for the attacker is to play $t \xrightarrow{a} t''$. Should the attacker choose any other move, he would lose immediately since s' and t' are bisimilar.

Let us now consider the following process rewrite system Δ where $\alpha_{\texttt{one}}$ and $\alpha_{\texttt{two}}$ are some fixed process expressions and $\oplus$ is either a sequential or parallel composition.

$$\begin{array}{ll}
X \xrightarrow{a} Y^{\texttt{choice}} & \\
X \xrightarrow{a} Y^{\texttt{one}} & X' \xrightarrow{a} Y^{\texttt{one}} \\
X \xrightarrow{a} Y^{\texttt{two}} & X' \xrightarrow{a} Y^{\texttt{two}} \\
 & \\
Y^{\texttt{choice}} \xrightarrow{\texttt{one}} Z \oplus \alpha_{\texttt{one}} & Y^{\texttt{one}} \xrightarrow{\texttt{one}} Z' \oplus \alpha_{\texttt{one}} \\
Y^{\texttt{choice}} \xrightarrow{\texttt{two}} Z \oplus \alpha_{\texttt{two}} & Y^{\texttt{two}} \xrightarrow{\texttt{two}} Z' \oplus \alpha_{\texttt{two}} \\
Y^{\texttt{one}} \xrightarrow{\texttt{two}} Z \oplus \alpha_{\texttt{two}} & Y^{\texttt{two}} \xrightarrow{\texttt{one}} Z \oplus \alpha_{\texttt{one}}
\end{array}$$

The transition systems generated by the processes (X, Δ) and (X', Δ) are depicted in Figure 6.1.

The intuition behind the construction can be nicely explained in terms of bisimulation games. Consider a strong bisimulation game starting from the pair X and X'.

The attacker is forced to make the first move by playing $X \xrightarrow{a} Y^{\texttt{choice}}$ because in all other possible moves, either from X or X', the defender can make the resulting processes syntactically equal and hence bisimilar. The defender's answer to the move $X \xrightarrow{a} Y^{\texttt{choice}}$ is either (i) $X' \xrightarrow{a} Y^{\texttt{one}}$ or (ii) $X' \xrightarrow{a} Y^{\texttt{two}}$.

In the next round starting from (i) $Y^{\texttt{choice}}$ and $Y^{\texttt{one}}$ or (ii) $Y^{\texttt{choice}}$ and $Y^{\texttt{two}}$, the attacker can use either the action `one` or `two` — obviously it is irrelevant whether the chosen action is performed in the first or in the second process. In case (i), if the attacker chooses the action `one` then the players reach the pair $Z \oplus \alpha_{\texttt{one}}$ and $Z' \oplus \alpha_{\texttt{one}}$. If he chooses the action `two` then the players reach a pair of syntactically equal states, namely $Z \oplus \alpha_{\texttt{two}}$ and $Z \oplus \alpha_{\texttt{two}}$, from which the defender has an obvious winning strategy. In case (ii), if the attacker chooses the action `two` then the players reach the pair $Z \oplus \alpha_{\texttt{two}}$ and $Z' \oplus \alpha_{\texttt{two}}$. If he chooses the action `one` then he loses as in case (i).

Now, either the defender won by reaching syntactically equal states, or the resulting processes after two rounds are (i) $Z \oplus \alpha_{\texttt{one}}$ and $Z' \oplus \alpha_{\texttt{one}}$ or (ii) $Z \oplus \alpha_{\texttt{two}}$ and $Z' \oplus \alpha_{\texttt{two}}$. Note that it was the defender who had the possibility to decide between adding $\alpha_{\texttt{one}}$ or $\alpha_{\texttt{two}}$.

We can repeat this construction several times in a row, which is explained in more detail in the following sections.

6.3 Hardness of Strong Bisimilarity

The hardness results are proved again by polynomial reductions from a PSPACE-complete problem of quantified satisfiability (QSAT) [145].

In this section we will use the version where the prefix of quantifiers starts with the existential one.

Problem:	QSAT
Instance:	A natural number $n > 0$ and a boolean formula ϕ in conjunctive normal form with boolean variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$.
Question:	Is $\exists x_1 \forall y_1 \exists x_2 \forall y_2 \ldots \exists x_n \forall y_n . \phi$ true?

A *literal* is a variable or the negation of a variable. An instance of QSAT is a formula C of the form

$$C \equiv \exists x_1 \forall y_1 \exists x_2 \forall y_2 \ldots \exists x_n \forall y_n .\ C_1 \wedge C_2 \wedge \ldots \wedge C_k$$

where each *clause* C_j, $1 \leq j \leq k$, is a disjunction of literals.

6.3.1 Basic Parallel Processes

In this subsection we show that strong bisimilarity of BPP is a PSPACE-hard problem. We prove it by reduction from QSAT. Let

$$C \equiv \exists x_1 \forall y_1 \exists x_2 \forall y_2 \dots \exists x_n \forall y_n.\ C_1 \wedge C_2 \wedge \dots \wedge C_k$$

be an instance of QSAT.

We define the following BPP processes (X_1, Δ) and (X_1', Δ) where

$$\begin{aligned}\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \ & \{Q_1, \dots, Q_k\} \cup \\ & \{X_1, \dots, X_{n+1}, Y_1^{\texttt{choice}}, \dots, Y_n^{\texttt{choice}}, Z_1, \dots, Z_n\} \cup \\ & \{X_1', \dots, X_{n+1}', Y_1^{\texttt{tt}}, \dots, Y_n^{\texttt{tt}}, Y_1^{\texttt{ff}}, \dots, Y_n^{\texttt{ff}}, Z_1', \dots, Z_n'\}\end{aligned}$$

and $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{q_1, \dots, q_k, a, \texttt{tt}, \texttt{ff}\}$.

For each i, $1 \leq i \leq n$, let

α_i be a parallel composition $Q_{i_1} \| Q_{i_2} \| \cdots \| Q_{i_\ell}$ such that
$1 \leq i_1 < i_2 < \cdots < i_\ell \leq k$ and
$C_{i_1}, C_{i_2}, \dots, C_{i_\ell}$ are all the clauses where x_i occurs positively,

$\overline{\alpha_i}$ be a parallel composition $Q_{i_1} \| Q_{i_2} \| \cdots \| Q_{i_\ell}$ such that
$1 \leq i_1 < i_2 < \cdots < i_\ell \leq k$ and
$C_{i_1}, C_{i_2}, \dots, C_{i_\ell}$ are all the clauses where x_i occurs negatively,

β_i be a parallel composition $Q_{i_1} \| Q_{i_2} \| \cdots \| Q_{i_\ell}$ such that
$1 \leq i_1 < i_2 < \cdots < i_\ell \leq k$ and
$C_{i_1}, C_{i_2}, \dots, C_{i_\ell}$ are all the clauses where y_i occurs positively, and

$\overline{\beta_i}$ be a parallel composition $Q_{i_1} \| Q_{i_2} \| \cdots \| Q_{i_\ell}$ such that
$1 \leq i_1 < i_2 < \cdots < i_\ell \leq k$ and
$C_{i_1}, C_{i_2}, \dots, C_{i_\ell}$ are all the clauses where y_i occurs negatively.

Example 6.1. Let us consider a quantified boolean formula

$$\exists x_1 \forall y_1 \exists x_2 \forall y_2.\ (x_1 \vee \neg y_1 \vee y_2) \wedge (\neg x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee y_2 \vee \neg y_2)$$

where $n = 2$, $k = 3$, $C_1 = x_1 \vee \neg y_1 \vee y_2$, $C_2 = \neg x_1 \vee y_1 \vee y_2$ and $C_3 = x_1 \vee y_1 \vee y_2 \vee \neg y_2$. Then

$$\begin{array}{llll}\alpha_1 = Q_1 \| Q_3 & \overline{\alpha_1} = Q_2 & \beta_1 = Q_2 \| Q_3 & \overline{\beta_1} = Q_1 \\ \alpha_2 = \epsilon & \overline{\alpha_2} = \epsilon & \beta_2 = Q_1 \| Q_2 \| Q_3 & \overline{\beta_2} = Q_3.\end{array}$$

□

The set Δ is given by the following rewrite rules:

- for all j such that $1 \leq j \leq k$:

$$Q_j \xrightarrow{q_j} Q_j$$

- for all i such that $1 \leq i \leq n$:

$$\begin{array}{ll}
X_i \xrightarrow{a} Y_i^{\texttt{choice}} & \\
X_i \xrightarrow{a} Y_i^{\texttt{tt}} & X'_i \xrightarrow{a} Y_i^{\texttt{tt}} \\
X_i \xrightarrow{a} Y_i^{\texttt{ff}} & X'_i \xrightarrow{a} Y_i^{\texttt{ff}} \\
 & \\
Y_i^{\texttt{choice}} \xrightarrow{\texttt{tt}} Z_i \| \alpha_i & Y_i^{\texttt{tt}} \xrightarrow{\texttt{tt}} Z'_i \| \alpha_i \\
Y_i^{\texttt{choice}} \xrightarrow{\texttt{ff}} Z_i \| \overline{\alpha_i} & Y_i^{\texttt{ff}} \xrightarrow{\texttt{ff}} Z'_i \| \overline{\alpha_i} \\
Y_i^{\texttt{tt}} \xrightarrow{\texttt{ff}} Z_i \| \overline{\alpha_i} & Y_i^{\texttt{ff}} \xrightarrow{\texttt{tt}} Z_i \| \alpha_i \\
 & \\
Z_i \xrightarrow{\texttt{tt}} X_{i+1} \| \beta_i & Z'_i \xrightarrow{\texttt{tt}} X'_{i+1} \| \beta_i \\
Z_i \xrightarrow{\texttt{ff}} X_{i+1} \| \overline{\beta_i} & Z'_i \xrightarrow{\texttt{ff}} X'_{i+1} \| \overline{\beta_i}
\end{array}$$

- and

$$X_{n+1} \xrightarrow{a} Q_1 \| Q_2 \| \ldots \| Q_{k-1} \| Q_k \qquad X'_{n+1} \xrightarrow{a} \epsilon.$$

We can see the processes (X_1, Δ) and (X'_1, Δ) in Figure 6.2 (if we set $i = 1$ and $\gamma_1 = \epsilon$). The intuition behind the construction will be explained in terms of bisimulation games and follows the main idea from Section 6.2. Consider a strong bisimulation game starting from the pair of processes X_1 and X'_1.

The attacker is forced to make the first move by playing $X_1 \xrightarrow{a} Y_1^{\texttt{choice}}$ because in all other possible moves, either from X_1 or X'_1, the defender can make the resulting processes syntactically equal and hence bisimilar. The defender's answer to the move $X_1 \xrightarrow{a} Y_1^{\texttt{choice}}$ is either (i) $X'_1 \xrightarrow{a} Y_1^{\texttt{tt}}$ (this corresponds to setting the variable x_1 to true) or (ii) $X'_1 \xrightarrow{a} Y_1^{\texttt{ff}}$ (this corresponds to setting the variable x_1 to false).

In the next round the attacker is forced to take the action (i) tt or (ii) ff, according to the defender's choice in the first round. Otherwise the attacker loses. The defender can only imitate the same action in the other process. The resulting processes after two rounds are (i) $Z_1 \| \alpha_1$ and $Z'_1 \| \alpha_1$ or (ii) $Z_1 \| \overline{\alpha_1}$ and $Z'_1 \| \overline{\alpha_1}$. Note that it was the defender who had the possibility to decide between adding α_1 (i.e. setting x_1 to true) or $\overline{\alpha_1}$ (i.e. setting x_1 to false).

In the third round the attacker has the choice of playing either along the action tt or ff, which corresponds to the universal quantifier in front of y_1. It does not matter in which process the attacker performs the move. The defender has only one possibility how to answer to this move — he must imitate the corresponding move in the other process. The resulting processes are $X_2 \| \gamma_2$ and $X'_2 \| \gamma_2$ such that $\gamma_2 = \widetilde{\alpha}_1 \| \widetilde{\beta}_1$ where $\widetilde{\alpha}_1 \in \{\alpha_1, \overline{\alpha_1}\}$ and $\widetilde{\beta}_1 \in \{\beta_1, \overline{\beta_1}\}$ according to the truth values chosen for x_1 (by the defender) and for y_1 (by the attacker). Now the game continues in a similar way from $X_2 \| \gamma_2$ and $X'_2 \| \gamma_2$. Playing some of the actions $q_1, \ldots, q_k$ cannot make the attacker win since the defender has always the possibility to imitate the same move in the other processes.

Hence if the attacker wants to win he has to reach eventually the states $X_{n+1} \| \gamma_{n+1}$ and $X'_{n+1} \| \gamma_{n+1}$, and then he performs the move $X_{n+1} \| \gamma_{n+1} \xrightarrow{a}$

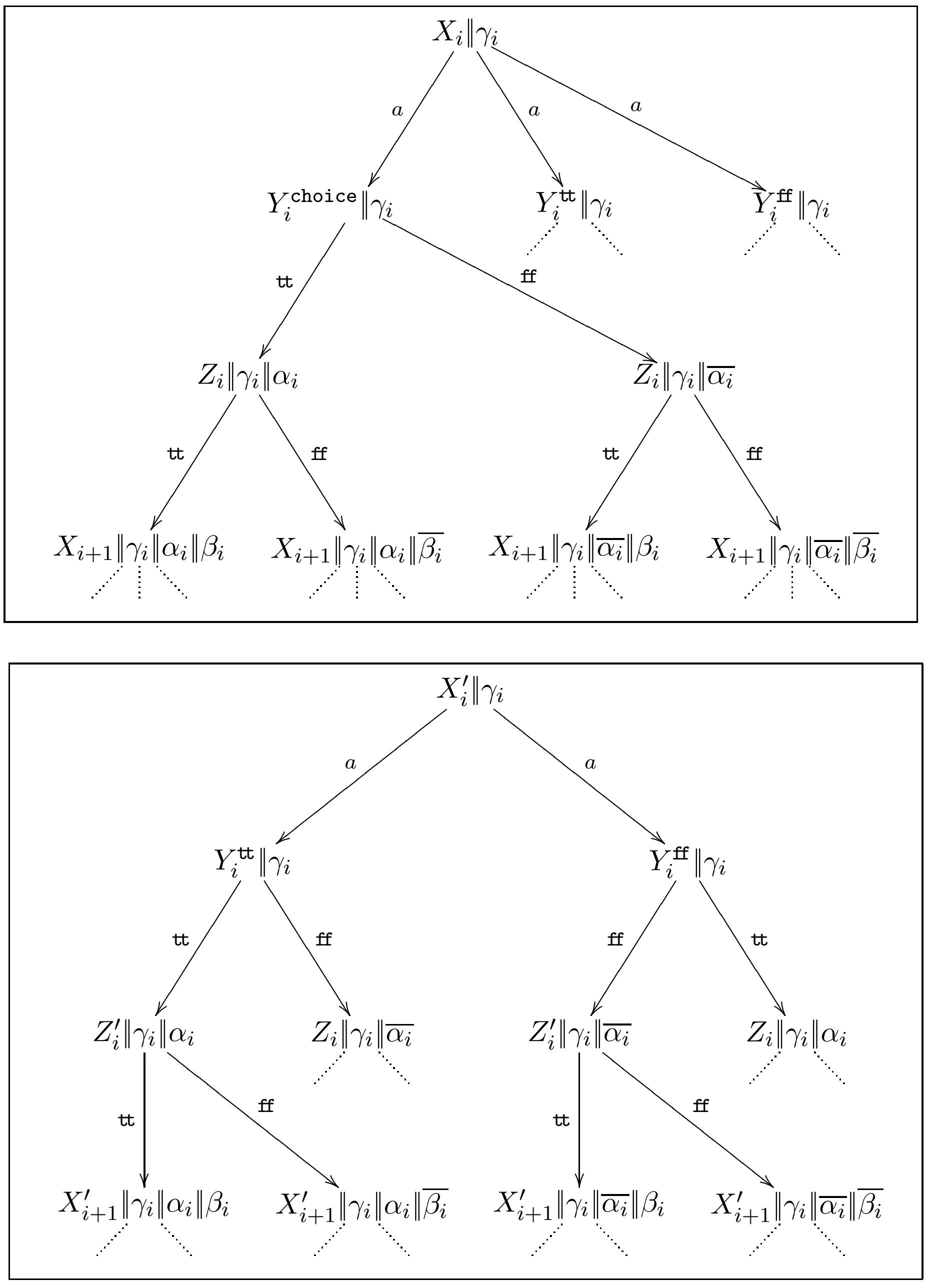

$X_i \| \gamma_i$
a
$Y_i^{\texttt{choice}} \| \gamma_i$
$Y_i^{\texttt{tt}} \| \gamma_i$
$Y_i^{\texttt{ff}} \| \gamma_i$
$\texttt{tt}$
$\texttt{ff}$
$Z_i \| \gamma_i \| \alpha_i$
$Z_i \| \gamma_i \| \overline{\alpha_i}$
$X_{i+1} \| \gamma_i \| \alpha_i \| \beta_i$
$X_{i+1} \| \gamma_i \| \alpha_i \| \overline{\beta_i}$
$X_{i+1} \| \gamma_i \| \overline{\alpha_i} \| \beta_i$
$X_{i+1} \| \gamma_i \| \overline{\alpha_i} \| \overline{\beta_i}$
$X'_i \| \gamma_i$
$Z'_i \| \gamma_i \| \alpha_i$
$Z'_i \| \gamma_i \| \overline{\alpha_i}$
$X'_{i+1} \| \gamma_i \| \alpha_i \| \beta_i$
$X'_{i+1} \| \gamma_i \| \alpha_i \| \overline{\beta_i}$
$X'_{i+1} \| \gamma_i \| \overline{\alpha_i} \| \beta_i$
$X'_{i+1} \| \gamma_i \| \overline{\alpha_i} \| \overline{\beta_i}$

Figure 6.2: Processes $(X_i \| \gamma_i, \Delta)$ and $(X'_i \| \gamma_i, \Delta)$

$Q_1 \| Q_2 \| \ldots \| Q_{k-1} \| Q_k \| \gamma_{n+1}$ to which the defender has only one answer, namely $X'_{n+1} \| \gamma_{n+1} \xrightarrow{a} \gamma_{n+1}$. From the states $Q_1 \| Q_2 \| \ldots \| Q_{k-1} \| Q_k \| \gamma_{n+1}$ and γ_{n+1} the attacker has the possibility to check whether every clause C_j, $1 \leq j \leq k$, in C is indeed satisfied under the generated truth assignment by using the rule $Q_j \xrightarrow{q_j} Q_j$ in the first process. If the clause C_j is not satisfied then Q_j does not appear in γ_{n+1} and the defender loses. If all the clauses in C are satisfied then $Q_1 \| Q_2 \| \ldots \| Q_{k-1} \| Q_k \| \gamma_{n+1} \sim \gamma_{n+1}$ and the defender wins.

In what follows we formally prove that C is true iff $(X_1, \Delta) \sim (X'_1, \Delta)$.

Lemma 6.1. If $(X_1, \Delta) \sim (X'_1, \Delta)$ then the formula C is true.

Proof. We show that $(X_1, \Delta) \not\sim (X'_1, \Delta)$ under the assumption that C is false. If C is false then C' defined by

$$C' \stackrel{\text{def}}{=} \forall x_1 \exists y_1 \forall x_2 \exists y_2 \ldots \forall x_n \exists y_n.\ \neg(C_1 \wedge C_2 \wedge \ldots \wedge C_k)$$

is true and we claim that the attacker has a winning strategy in the bisimulation game starting from (X_1, Δ) and (X'_1, Δ). The attacker's strategy starts with performing a sequence of actions

$$a, \widetilde{x}_1, \widetilde{y}_1, \ldots, a, \widetilde{x}_i, \widetilde{y}_i, \ldots, a, \widetilde{x}_n, \widetilde{y}_n, a$$

where $\widetilde{x}_i, \widetilde{y}_i \in \{\mathtt{tt}, \mathtt{ff}\}$ for all i, $1 \leq i \leq n$. The attacker is playing only in the first process (X_1, Δ). The choice of $\widetilde{x}_i$ is done by the defender and of $\widetilde{y}_i$ by the attacker — see the discussion above. This means that whatever values for $\widetilde{x}_1, \ldots, \widetilde{x}_n$ are chosen by the defender, the attacker can still decide on values for $\widetilde{y}_1, \ldots, \widetilde{y}_n$ such that the generated assignment satisfies the formula $\neg(C_1 \wedge C_2 \wedge \ldots \wedge C_k)$. Hence there must be some j, $1 \leq j \leq k$, such that the clause C_j is not satisfied. This implies that Q_j does not occur in the second process. However, the attacker can perform the action q_j in the first process by using the rule $Q_j \xrightarrow{q_j} Q_j$. Thus the attacker has a winning strategy in the bisimulation game and $(X_1, \Delta) \not\sim (X'_1, \Delta)$. □

Lemma 6.2. If the formula C is true then $(X_1, \Delta) \sim (X'_1, \Delta)$.

Proof. Let us define sets $\mathcal{AS}_i$, corresponding to the assignments of variables from x_1, y_1 to x_i, y_i, $1 \leq i \leq n$, such that the formula

$$\exists x_{i+1} \forall y_{i+1} \ldots \exists x_n \forall y_n.\ C_1 \wedge C_2 \wedge \ldots \wedge C_k$$

is still true.

For $i \in \{1, \ldots, n\}$ let $\mathcal{AS}_i \stackrel{\text{def}}{=}$

$\Big\{ \widetilde{\alpha}_1 \| \widetilde{\beta}_1 \| \widetilde{\alpha}_2 \| \widetilde{\beta}_2 \| \ldots \| \widetilde{\alpha}_i \| \widetilde{\beta}_i \ \mid$

such that for all j, $1 \leq j \leq i$, it holds that $\widetilde{\alpha}_j \in \{\alpha_j, \overline{\alpha_j}\}$ and $\widetilde{\beta}_j \in \{\beta_j, \overline{\beta_j}\}$, and under the assignment

$$x_j = \begin{cases} \mathtt{tt} & \text{if } \widetilde{\alpha}_j = \alpha_j \\ \mathtt{ff} & \text{if } \widetilde{\alpha}_j = \overline{\alpha_j} \end{cases} \quad \text{and} \quad y_j = \begin{cases} \mathtt{tt} & \text{if } \widetilde{\beta}_j = \beta_j \\ \mathtt{ff} & \text{if } \widetilde{\beta}_j = \overline{\beta_j} \end{cases}$$

the formula $\exists x_{i+1} \forall y_{i+1} \ldots \exists x_n \forall y_n.\ C_1 \wedge C_2 \wedge \ldots \wedge C_k$ is true $\Big\}$.

By definition $\mathcal{AS}_0 \overset{\text{def}}{=} \{\epsilon\}$. In particular, $\mathcal{AS}_n$ contains all the assignments for which the unquantified formula $C_1 \wedge C_2 \wedge \ldots \wedge C_k$ is true. The following relation is a strong bisimulation (assuming that the index i ranges over the set $\{1, \ldots, n\}$).

$$\begin{array}{l}
\{(X_i \| \gamma_i, X'_i \| \gamma_i) \mid \gamma_i \in \mathcal{AS}_{i-1}\} \cup \\
\{(Y_i^{\mathtt{choice}} \| \gamma_i, Y_i^{\mathtt{tt}} \| \gamma_i) \mid \gamma_i \| \alpha_i \| \beta_i \in \mathcal{AS}_i \ \wedge \ \gamma_i \| \alpha_i \| \overline{\beta_i} \in \mathcal{AS}_i\} \cup \\
\{(Y_i^{\mathtt{choice}} \| \gamma_i, Y_i^{\mathtt{ff}} \| \gamma_i) \mid \gamma_i \| \overline{\alpha_i} \| \beta_i \in \mathcal{AS}_i \ \wedge \ \gamma_i \| \overline{\alpha_i} \| \overline{\beta_i} \in \mathcal{AS}_i\} \cup \\
\{(Z_i \| \gamma_i \| \alpha_i, Z'_i \| \gamma_i \| \alpha_i) \mid \gamma_i \| \alpha_i \| \beta_i \in \mathcal{AS}_i \ \wedge \ \gamma_i \| \alpha_i \| \overline{\beta_i} \in \mathcal{AS}_i\} \cup \\
\{(Z_i \| \gamma_i \| \overline{\alpha_i}, Z'_i \| \gamma_i \| \overline{\alpha_i}) \mid \gamma_i \| \overline{\alpha_i} \| \beta_i \in \mathcal{AS}_i \ \wedge \ \gamma_i \| \overline{\alpha_i} \| \overline{\beta_i} \in \mathcal{AS}_i\} \cup \\
\{(X_{n+1} \| \gamma_{n+1}, X'_{n+1} \| \gamma_{n+1}) \mid \gamma_{n+1} \in \mathcal{AS}_n\} \cup \\
\{(Q_1 \| Q_2 \| \ldots \| Q_{k-1} \| Q_k \| \gamma_{n+1}, \gamma_{n+1}) \mid \gamma_{n+1} \in \mathcal{AS}_n\} \cup \\
\{(E, E) \mid E \in \mathcal{P}\big(\mathit{Const}(\Delta)\big)\}
\end{array}$$

Since $\mathcal{AS}_0 = \{\epsilon\}$, we get that the pair (X_1, X'_1) is an element of this relation. Hence we proved that $(X_1, \Delta) \sim (X'_1, \Delta)$. □

Theorem 6.1. Strong bisimilarity of BPP is PSPACE-hard.

Proof. By Lemma 6.1 and Lemma 6.2. □

Remark 6.1. Notice that there are only finitely many reachable states from (X_1, Δ) and (X'_1, Δ). Hence (X_1, Δ) and (X'_1, Δ) are strongly regular processes.

Remark 6.2. Theorem 6.1 can be easily extended to 1-safe Petri nets where each transition has exactly one input place (for related results about 1-safe Petri nets see e.g. [98]). It is enough to introduce for each α_i, $\overline{\alpha_i}$, β_i and $\overline{\beta_i}$, $1 \leq i \leq n$, a new set of process constants $\{Q_1, \ldots, Q_k\}$ to ensure that in each reachable marking there is at most one token in every place.

6.3.2 Basic Process Algebra

In this subsection we show that strong bisimilarity of BPA is PSPACE-hard. The proof is again by reduction from QSAT, using the main idea from Section 6.2. However, there is a substantial difference from the proof for BPP explained in the previous subsection.

In case of BPP it is easier to check which clauses of a given boolean formula are satisfied because we have a parallel (and thus simultaneous) access to all process constants contained in the current state. This technique has to be modified to work for BPA since we have only a sequential access to the process constants contained in the current state, and there is no possibility of remembering any information in e.g. a finite-state control unit as in pushdown systems. Hence we have to encode the information about satisfied clauses in a unary way to achieve our result.

Let

$$C \equiv \exists x_1 \forall y_1 \exists x_2 \forall y_2 \ldots \exists x_n \forall y_n.\ C_1 \wedge C_2 \wedge \ldots \wedge C_k$$

be an instance of QSAT. Assume that $Q_1, \ldots, Q_k$ are process constants.

For each i, $1 \le i \le n$, let

α_i be a sequential composition $Q_{i_1}.Q_{i_2}.\cdots.Q_{i_\ell}$ such that
$1 \le i_1 < i_2 < \cdots < i_\ell \le k$ and
$C_{i_1}, C_{i_2}, \ldots, C_{i_\ell}$ are all the clauses where x_i occurs positively,

$\overline{\alpha_i}$ be a sequential composition $Q_{i_1}.Q_{i_2}.\cdots.Q_{i_\ell}$ such that
$1 \le i_1 < i_2 < \cdots < i_\ell \le k$ and
$C_{i_1}, C_{i_2}, \ldots, C_{i_\ell}$ are all the clauses where x_i occurs negatively,

β_i be a sequential composition $Q_{i_1}.Q_{i_2}.\cdots.Q_{i_\ell}$ such that
$1 \le i_1 < i_2 < \cdots < i_\ell \le k$ and
$C_{i_1}, C_{i_2}, \ldots, C_{i_\ell}$ are all the clauses where y_i occurs positively, and

$\overline{\beta_i}$ be a sequential composition $Q_{i_1}.Q_{i_2}.\cdots.Q_{i_\ell}$ such that
$1 \le i_1 < i_2 < \cdots < i_\ell \le k$ and
$C_{i_1}, C_{i_2}, \ldots, C_{i_\ell}$ are all the clauses where y_i occurs negatively.

Example 6.2. Let us consider a quantified formula

$$\exists x_1 \forall y_1 \exists x_2 \forall y_2.\ (x_1 \vee \neg y_1 \vee y_2) \wedge (\neg x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee y_2 \vee \neg y_2)$$

where $n = 2$, $k = 3$, $C_1 = x_1 \vee \neg y_1 \vee y_2$, $C_2 = \neg x_1 \vee y_1 \vee y_2$ and $C_3 = x_1 \vee y_1 \vee y_2 \vee \neg y_2$. Then

$$\begin{array}{llll} \alpha_1 = Q_1.Q_3 & \overline{\alpha_1} = Q_2 & \beta_1 = Q_2.Q_3 & \overline{\beta_1} = Q_1 \\ \alpha_2 = \epsilon & \overline{\alpha_2} = \epsilon & \beta_2 = Q_1.Q_2.Q_3 & \overline{\beta_2} = Q_3. \end{array}$$

□

Let $\mathcal{SF}(\gamma)$ be the set of all suffixes of a given sequential composition $\gamma \in \mathcal{S}(\{Q_1, \ldots, Q_k\})$, i.e., $\mathcal{SF}(\gamma) \overset{\text{def}}{=} \{\gamma' \mid \exists \gamma'' \text{ such that } \gamma''.\gamma' = \gamma\}$. Let M be the least natural number such that $M \ge 2n + 1$ and $M = 2^K$ for some natural number $K > 0$. Of course, $M > 1$.

We will define processes $(X_1.S, \Delta)$ and $(X'_1.S, \Delta)$ such that $\mathcal{C}onst(\Delta) \overset{\text{def}}{=}$

$$\begin{array}{l}
\{A_0, A_1, A_2, \ldots, A_{kK-1}\} \ \cup\ \{Q_1, \ldots, Q_k\} \ \cup \\
\{V_i^{\gamma}, V_i^{\gamma,\texttt{choice}} \mid 1 \le i \le n \ \wedge\ \gamma \in \mathcal{SF}(\alpha_i) \cup \mathcal{SF}(\overline{\alpha_i})\} \ \cup \\
\{V_i^{'\gamma}, V_i^{\gamma,\texttt{yes}}, V_i^{\gamma,\texttt{no}} \mid 1 \le i \le n \ \wedge\ \gamma \in \mathcal{SF}(\alpha_i) \cup \mathcal{SF}(\overline{\alpha_i})\} \ \cup \\
\{W_i^{\gamma}, W_i^{\gamma,\texttt{choice}} \mid 1 \le i \le n \ \wedge\ \gamma \in \mathcal{SF}(\beta_i) \cup \mathcal{SF}(\overline{\beta_i})\} \ \cup \\
\{W_i^{'\gamma}, W_i^{\gamma,\texttt{yes}}, W_i^{\gamma,\texttt{no}} \mid 1 \le i \le n \ \wedge\ \gamma \in \mathcal{SF}(\beta_i) \cup \mathcal{SF}(\overline{\beta_i})\} \ \cup \\
\{X_i, Y_i^{\texttt{choice}}, Z_i, X'_i, Y_i^{\texttt{tt}}, Y_i^{\texttt{ff}}, Z'_i \mid 1 \le i \le n\} \ \cup\ \{X_{n+1}, X'_{n+1}, S\}
\end{array}$$

and $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{a, c, \texttt{tt}, \texttt{ff}, \texttt{yes}, \texttt{no}, s\}$. The first part of the rewrite system Δ consists of the rewrite rules:

$$\begin{array}{ll} A_0 \xrightarrow{c} \epsilon & \\ A_\ell \xrightarrow{c} A_{\ell-1}.A_{\ell-2}.\cdots.A_1.A_0 & \text{for all } \ell,\ 1 \le \ell \le kK-1 \\ & \\ Q_j \xrightarrow{c} A_{jK-1}.A_{jK-2}.\cdots.A_1.A_0 & \text{for all } j,\ 1 \le j \le k. \end{array}$$

Remark 6.3. Notice that the size of the previously introduced rewrite rules is polynomial w.r.t. the size of the formula C. Moreover $A_\ell \xrightarrow{c^{2^\ell}} \epsilon$ for all ℓ, $0 \le \ell \le kK-1$, which implies by using the equation $2^{jK} = M^j$ that $Q_j \xrightarrow{c^{M^j}} \epsilon$ for all j, $1 \le j \le k$. Hence Q_j can perform exactly M^j transitions labelled by the "counting" action c and then disappears.

The intuition is that each clause C_j, $1 \le j \le k$, is coded by the process constant Q_j, which enables to perform exactly M^j of c actions. The key idea of our proof is then that the defender and the attacker will choose truth values for the variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, respectively. During this process some of the clauses $C_1, \ldots, C_k$ become satisfied, and the defender will have the possibility to add the corresponding process constants $Q_1, \ldots, Q_k$ to the current state.

Moreover, the defender will be able to select which of the process constants (corresponding to the satisfied clauses) appear in the current state in such a way that each of them appears there exactly once.

The following lemma shall be essential for proving our reduction correct.

Lemma 6.3. Assume that M and k are the constants introduced above, i.e., $M > 1$ and $k > 0$. Let a_j, $1 \le j \le k$, be natural numbers such that $0 \le a_j \le M-1$ for all j. The following two statements are equivalent:

$$(i)\ \sum_{j=1}^{k} a_j M^j = \sum_{j=1}^{k} M^j \qquad\qquad (ii)\ \ a_j = 1 \text{ for all } j,\ 1 \le j \le k.$$

Proof. By uniqueness of M-ary representations.

Obviously (ii) implies (i). By induction on k we prove the other direction. If $k = 1$ then $a_1 M = M$ immediately gives that $a_1 = 1$. Let $\sum_{j=1}^{k+1} a_j M^j = \sum_{j=1}^{k+1} M^j$. Since $a_j \le M-1$ for all j, $1 \le j \le k+1$, we get that

$$\sum_{j=1}^{k} a_j M^j \le \sum_{j=1}^{k} (M-1) M^j = (M-1)M \cdot \sum_{j=0}^{k-1} M^j =$$

$$(M-1)M \cdot \frac{M^k - 1}{M-1} = M^{k+1} - M < M^{k+1}.$$

Let us now consider the equation

$$\sum_{j=1}^{k} a_j M^j + a_{k+1} M^{k+1} = \sum_{j=1}^{k} M^j + M^{k+1}.$$

The fact that $\sum_{j=1}^{k} a_j M^j < M^{k+1}$ gives that $a_{k+1} \geq 1$. On the other hand

$$\sum_{j=1}^{k} M^j = M \cdot \sum_{j=0}^{k-1} M^j = M \cdot \frac{M^k - 1}{M - 1} < M^{k+1},$$

which implies that $a_{k+1} \leq 1$.

Hence $a_{k+1} = 1$ and the following equation must be satisfied

$$\sum_{j=1}^{k} a_j M^j = \sum_{j=1}^{k} M^j.$$

By induction hypothesis $a_j = 1$ also for all j, $1 \leq j \leq k$. $\square$

We continue with the definition of the set of rewrite rules Δ. For all i, $1 \leq i \leq n$, and $Q.\gamma \in \mathcal{SF}(\alpha_i) \cup \mathcal{SF}(\overline{\alpha_i})$ where $Q \in \{Q_1, \ldots, Q_k\}$ and $\gamma \in \mathcal{S}(\{Q_1, \ldots, Q_k\})$, Δ contains the rules:

$$\begin{array}{ll}
V_i^{Q.\gamma} \xrightarrow{a} V_i^{Q.\gamma,\mathtt{choice}} & \\
V_i^{Q.\gamma} \xrightarrow{a} V_i^{Q.\gamma,\mathtt{yes}} & V_i^{\prime Q.\gamma} \xrightarrow{a} V_i^{Q.\gamma,\mathtt{yes}} \\
V_i^{Q.\gamma} \xrightarrow{a} V_i^{Q.\gamma,\mathtt{no}} & V_i^{\prime Q.\gamma} \xrightarrow{a} V_i^{Q.\gamma,\mathtt{no}} \\
 & \\
V_i^{Q.\gamma,\mathtt{choice}} \xrightarrow{\mathtt{yes}} V_i^{\gamma}.Q & V_i^{Q.\gamma,\mathtt{yes}} \xrightarrow{\mathtt{yes}} V_i^{\prime\gamma}.Q \\
V_i^{Q.\gamma,\mathtt{choice}} \xrightarrow{\mathtt{no}} V_i^{\gamma} & V_i^{Q.\gamma,\mathtt{no}} \xrightarrow{\mathtt{no}} V_i^{\prime\gamma} \\
V_i^{Q.\gamma,\mathtt{yes}} \xrightarrow{\mathtt{no}} V_i^{\gamma} & V_i^{Q.\gamma,\mathtt{no}} \xrightarrow{\mathtt{yes}} V_i^{\gamma}.Q.
\end{array}$$

Similarly, for all i, $1 \leq i \leq n$, and $Q.\gamma \in \mathcal{SF}(\beta_i) \cup \mathcal{SF}(\overline{\beta_i})$ where $Q \in \{Q_1, \ldots, Q_k\}$ and $\gamma \in \mathcal{S}(\{Q_1, \ldots, Q_k\})$, Δ contains the rules:

$$\begin{array}{ll}
W_i^{Q.\gamma} \xrightarrow{a} W_i^{Q.\gamma,\mathtt{choice}} & \\
W_i^{Q.\gamma} \xrightarrow{a} W_i^{Q.\gamma,\mathtt{yes}} & W_i^{\prime Q.\gamma} \xrightarrow{a} W_i^{Q.\gamma,\mathtt{yes}} \\
W_i^{Q.\gamma} \xrightarrow{a} W_i^{Q.\gamma,\mathtt{no}} & W_i^{\prime Q.\gamma} \xrightarrow{a} W_i^{Q.\gamma,\mathtt{no}} \\
 & \\
W_i^{Q.\gamma,\mathtt{choice}} \xrightarrow{\mathtt{yes}} W_i^{\gamma}.Q & W_i^{Q.\gamma,\mathtt{yes}} \xrightarrow{\mathtt{yes}} W_i^{\prime\gamma}.Q \\
W_i^{Q.\gamma,\mathtt{choice}} \xrightarrow{\mathtt{no}} W_i^{\gamma} & W_i^{Q.\gamma,\mathtt{no}} \xrightarrow{\mathtt{no}} W_i^{\prime\gamma} \\
W_i^{Q.\gamma,\mathtt{yes}} \xrightarrow{\mathtt{no}} W_i^{\gamma} & W_i^{Q.\gamma,\mathtt{no}} \xrightarrow{\mathtt{yes}} W_i^{\gamma}.Q.
\end{array}$$

Assume now a strong bisimulation game starting from the pair $(V_i^{\alpha_i}, \Delta)$ and $(V_i^{\prime\alpha_i}, \Delta)$. As shown in Section 6.2, either in some round the states become syntactically equal, or the defender has the possibility to choose in the first round the next states (i) $V_i^{\alpha_i,\mathtt{choice}}$ and $V_i^{\alpha_i,\mathtt{yes}}$ or (ii) $V_i^{\alpha_i,\mathtt{choice}}$ and $V_i^{\alpha_i,\mathtt{no}}$. This means that in the next round a process constant Q such that $\alpha_i = Q.\alpha'_i$ for some α'_i is either (i) added to the current state or (ii) left out.

Now the game continues either from (i) $V_i^{\alpha'_i}.Q$ and $V_i^{\prime\alpha'_i}.Q$ or from (ii) $V_i^{\alpha'_i}$ and $V_i^{\prime\alpha'_i}$. This repeats in similar fashion until the states $V_i^{\epsilon}.\gamma_i$ and $V_i^{\prime\epsilon}.\gamma_i$ are reached, such that γ_i is some subsequence of α_i (in a reverse order) and it was the defender who had the possibility to decide which of the process constants contained in α_i appear also in γ_i.

The same happens if we start playing the strong bisimulation game from $(V_i^{\overline{\alpha_i}}, \Delta)$ and $(V_i'^{\overline{\alpha_i}}, \Delta)$, or $(W_i^{\beta_i}, \Delta)$ and $(W_i'^{\beta_i}, \Delta)$, or $(W_i^{\overline{\beta_i}}, \Delta)$ and $(W_i'^{\overline{\beta_i}}, \Delta)$.

We finish the definition of Δ by adding the rules:

- for all i, $1 \le i \le n$:

$$\begin{array}{ll}
X_i \xrightarrow{a} Y_i^{\mathtt{choice}} & \\
X_i \xrightarrow{a} Y_i^{\mathtt{tt}} & X_i' \xrightarrow{a} Y_i^{\mathtt{tt}} \\
X_i \xrightarrow{a} Y_i^{\mathtt{ff}} & X_i' \xrightarrow{a} Y_i^{\mathtt{ff}} \\
 & \\
Y_i^{\mathtt{choice}} \xrightarrow{\mathtt{tt}} V_i^{\alpha_i} & Y_i^{\mathtt{tt}} \xrightarrow{\mathtt{tt}} V_i'^{\alpha_i} \\
Y_i^{\mathtt{choice}} \xrightarrow{\mathtt{ff}} V_i^{\overline{\alpha_i}} & Y_i^{\mathtt{ff}} \xrightarrow{\mathtt{ff}} V_i'^{\overline{\alpha_i}} \\
Y_i^{\mathtt{tt}} \xrightarrow{\mathtt{ff}} V_i^{\overline{\alpha_i}} & Y_i^{\mathtt{ff}} \xrightarrow{\mathtt{tt}} V_i^{\alpha_i} \\
 & \\
V_i^{\epsilon} \xrightarrow{a} Z_i & V_i'^{\epsilon} \xrightarrow{a} Z_i' \\
 & \\
Z_i \xrightarrow{\mathtt{tt}} W_i^{\beta_i} & Z_i' \xrightarrow{\mathtt{tt}} W_i'^{\beta_i} \\
Z_i \xrightarrow{\mathtt{ff}} W_i^{\overline{\beta_i}} & Z_i' \xrightarrow{\mathtt{ff}} W_i'^{\overline{\beta_i}} \\
 & \\
W_i^{\epsilon} \xrightarrow{a} X_{i+1} & W_i'^{\epsilon} \xrightarrow{a} X_{i+1}'
\end{array}$$

- and

$$X_{n+1} \xrightarrow{a} Q_1.Q_2.\ldots.Q_{k-1}.Q_k.S \qquad X_{n+1}' \xrightarrow{a} \epsilon \qquad S \xrightarrow{s} S.$$

Lemma 6.4. If $(X_1.S, \Delta) \sim (X_1'.S, \Delta)$ then the formula C is true.

Proof. We show that $(X_1.S, \Delta) \not\sim (X_1'.S, \Delta)$ under the assumption that C is false. If C is false then C' defined by

$$C' \stackrel{\text{def}}{=} \forall x_1 \exists y_1 \forall x_2 \exists y_2 \ldots \forall x_n \exists y_n.\ \neg(C_1 \wedge C_2 \wedge \ldots \wedge C_k)$$

is true and we claim that the attacker has a winning strategy in the bisimulation game starting from $(X_1.S, \Delta)$ and $(X_1'.S, \Delta)$. As mentioned in Section 6.2, in the first round the attacker is forced to perform the move $X_1.S \xrightarrow{a} Y_1^{\mathtt{choice}}.S$. The defender can respond either by (i) $X_1'.S \xrightarrow{a} Y_1^{\mathtt{tt}}.S$ (which corresponds to setting the variable x_1 to true) or by (ii) $X_1'.S \xrightarrow{a} Y_1^{\mathtt{ff}}.S$ (which corresponds to setting the variable x_1 to false). In the second round the attacker performs the action (i) $\mathtt{tt}$ or (ii) $\mathtt{ff}$, and the defender must answer by the same action in the other process. Now the game continues from (i) $V_1^{\alpha_1}.S$ and $V_1'^{\alpha_1}.S$ or (ii) $V_1^{\overline{\alpha_1}}.S$ and $V_1'^{\overline{\alpha_1}}.S$. Within the next (i) $2 \cdot |\alpha_1|$ or (ii) $2 \cdot |\overline{\alpha_1}|$ rounds (where $|w|$ is the length of w) the defender has the possibility to choose some subsequence of (i) α_1 or (ii) $\overline{\alpha_1}$ and add it in a reverse order to the current state. Then the game continues either from (i) $V_1^{\epsilon}.\gamma_1.S$ and $V_1'^{\epsilon}.\gamma_1.S$ or (ii) $V_1^{\epsilon}.\overline{\gamma_1}.S$ and $V_1'^{\epsilon}.\overline{\gamma_1}.S$, such that (i) γ_1 is a subsequence (in a reverse order and chosen by the defender) of α_1 or (ii) $\overline{\gamma_1}$ is a subsequence (in a reverse order and chosen by the defender) of $\overline{\alpha_1}$. The players have only one possible continuation of the game by using the rewrite rules $V_1^\epsilon \xrightarrow{a} Z_1$ and $V_1'^\epsilon \xrightarrow{a} Z_1'$, thus reaching the states (i) $Z_1.\gamma_1.S$ and $Z_1'.\gamma_1.S$ or (ii) $Z_1.\overline{\gamma_1}.S$ and $Z_1'.\overline{\gamma_1}.S$.

Now, it is the attacker who has the possibility of making a choice between the rewrite rules $Z_1 \xrightarrow{\mathtt{tt}} W_1^{\beta_1}$ or $Z_1 \xrightarrow{\mathtt{ff}} W_1^{\overline{\beta_1}}$ in the first process. This corresponds to setting the variable y_1 to true or false. The defender can only imitate the same action by using the rules $Z_1' \xrightarrow{\mathtt{tt}} W_1'^{\beta_1}$ or $Z_1' \xrightarrow{\mathtt{ff}} W_1'^{\overline{\beta_1}}$ in the other process. From the current states starting with $W_1^{\beta_1}$ and $W_1'^{\beta_1}$, or $W_1^{\overline{\beta_1}}$ and $W_1'^{\overline{\beta_1}}$, the same happens as before: the defender has the possibility of choosing a subsequence δ_1 (in a reverse order) of β_1 or a subsequence $\overline{\delta_1}$ (in a reverse order) of $\overline{\beta_1}$. So precisely after $2 \cdot |\beta_1|$ or $2 \cdot |\overline{\beta_1}|$ rounds the following four possible pairs of states can be reached: (1) $W_1^\epsilon.\delta_1.\gamma_1.S$ and $W_1'^\epsilon.\delta_1.\gamma_1.S$, or (2) $W_1^\epsilon.\delta_1.\overline{\gamma_1}.S$ and $W_1'^\epsilon.\delta_1.\overline{\gamma_1}.S$, or (3) $W_1^\epsilon.\overline{\delta_1}.\gamma_1.S$ and $W_1'^\epsilon.\overline{\delta_1}.\gamma_1.S$, or (4) $W_1^\epsilon.\overline{\delta_1}.\overline{\gamma_1}.S$ and $W_1'^\epsilon.\overline{\delta_1}.\overline{\gamma_1}.S$. We have now only one possible continuation of the game in the next round, reaching the states (1) $X_2.\delta_1.\gamma_1.S$ and $X_2'.\delta_1.\gamma_1.S$, or (2) $X_2.\delta_1.\overline{\gamma_1}.S$ and $X_2'.\delta_1.\overline{\gamma_1}.S$, or (3) $X_2.\overline{\delta_1}.\gamma_1.S$ and $X_2'.\overline{\delta_1}.\gamma_1.S$, or (4) $X_2.\overline{\delta_1}.\overline{\gamma_1}.S$ and $X_2'.\overline{\delta_1}.\overline{\gamma_1}.S$.

We remind the reader of the fact that the defender had the possibility to set the variable x_1 to true or false, and the attacker decided on the truth value for the variable y_1. In the meantime, all the process constants from $\{Q_1, \ldots, Q_k\}$ corresponding to the clauses that became satisfied by this assignment could have been potentially added to the current state, but it was the defender who had the possibility to filter some of them out.

In the next rounds the same schema of the game repeats, until we reach the states $X_{n+1}.\omega.S$ and $X_{n+1}'.\omega.S$. The defender decides on the truth values for each of the variables $x_2, \ldots, x_n$, and the attacker has the possibility to respond by choosing the truth values for the variables $y_2, \ldots, y_n$. During this some of the clauses appear to be satisfied and ω consists of a selection (made by the defender) of process constants corresponding to these clauses.

Since we assume that the formula C' is true, the attacker can decide on the truth values for $y_1, \ldots, y_n$ in such a way that at least one of the clauses $C_1, \ldots, C_k$ is not satisfied. Let us suppose that it is C_m for some m, $1 \leq m \leq k$, that is not satisfied. Hence Q_m cannot appear in ω and the attacker has the following winning strategy. He plays $X_{n+1}.\omega.S \xrightarrow{a} Q_1.Q_2.\ldots.Q_{k-1}.Q_k.S.\omega.S$, to which the defender can only answer by $X_{n+1}'.\omega.S \xrightarrow{a} \omega.S$.

The state $Q_1.Q_2.\ldots.Q_{k-1}.Q_k.S.\omega.S$ can perform exactly $\sum_{j=1}^{k} M^j$ of actions c (see Remark 6.3) followed by an infinite sequence of actions s. On the other hand, $\omega.S$ can never perform exactly $\sum_{j=1}^{k} M^j$ of actions c and then the infinite sequence of actions s. This follows from the fact that Q_m does not appear in ω and from Lemma 6.3 — obviously, any process constant from $\{Q_1, \ldots, Q_k\}$ can occur at most $2n$ times in ω ($2n \leq M - 1$), which satisfies the assumption of Lemma 6.3.

Hence the attacker has a winning strategy which implies that $(X_1.S, \Delta) \not\sim (X_1'.S, \Delta)$. $\square$

Lemma 6.5. If the formula C is true then $(X_1.S, \Delta) \sim (X'_1.S, \Delta)$.

Proof. Consider a strong bisimulation game starting from the pair $(X_1.S, \Delta)$ and $(X'_1.S, \Delta)$. We show that the defender has a winning strategy. As mentioned in Section 6.2 and in the proof above, the attacker is forced to play according to a strictly defined strategy, otherwise the defender can make the resulting processes immediately syntactically equal and hence bisimilar. As shown before the defender can make the choices between setting the variables $x_1, \ldots, x_n$ to true or false, whereas the attacker can decide on truth values for $y_1, \ldots, y_n$. Thus the defender can play the bisimulation game such that finally every clause $C_1, \ldots, C_k$ in C is satisfied. The defender has the possibility to add the corresponding process constants $Q_1, \ldots, Q_k$ to the current state in such a way that when reaching the states $X_{n+1}.\omega.S$ and $X'_{n+1}.\omega.S$, the sequential composition ω contains every Q_j exactly once for each j, $1 \leq j \leq k$. This can be easily achieved by following the strategy: "add Q_j to the current state if and only if it is not already present there". After performing the moves $X_{n+1}.\omega.S \xrightarrow{a} Q_1.Q_2.\ldots.Q_{k-1}.Q_k.S.\omega.S$ and $X'_{n+1}.\omega.S \xrightarrow{a} \omega.S$, the defender wins since $S.\omega.S \sim S$ and both $Q_1.Q_2.\ldots.Q_{k-1}.Q_k$ and ω can perform the same number of actions c and then become ϵ. Hence $(X_1.S, \Delta) \sim (X'_1.S, \Delta)$. □

Theorem 6.2. Strong bisimilarity of BPA is PSPACE-hard.

Proof. By Lemma 6.4 and Lemma 6.5. □

Remark 6.4. Notice that there are only finitely many reachable states from $(X_1.S, \Delta)$ and $(X'_1.S, \Delta)$. Hence $(X_1.S, \Delta)$ and $(X'_1.S, \Delta)$ are strongly regular processes.

6.4 Hardness of Strong Regularity

As already mentioned in Chapter 5, the idea to reduce bisimilarity to regularity first appeared in the literature due to Mayr [124]. He showed a technique for reducing weak bisimilarity of regular BPP to weak regularity of BPP. However, in his reduction τ actions are used.

Building upon Mayr's approach we will provide polynomial time reductions from strong bisimilarity of regular BPP (BPA) to strong regularity of BPP (BPA).

6.4.1 Basic Parallel Processes

Theorem 6.3 (Reduction from bisimilarity to regularity).
Let (P_1, Δ) and (P_2, Δ) be strongly regular BPP processes. We can construct in polynomial time a BPP process (P, Δ') such that

$$(P_1, \Delta) \sim (P_2, \Delta) \quad \text{if and only if} \quad (P, \Delta') \text{ is strongly regular.}$$

$$\begin{tikzcd}
X\|A^i\|B_c \arrow[d, "a^{i'}"] & \sim^? & X\|A^j\|B_c \arrow[d, "a^{i'}"] \\
P'_1\|A^{i-i'+1}\|A_c\|B_c \arrow[d, "a^{j-i'}"] & & X\|A^{j-i'}\|B_c \arrow[d, "a^{j-i'}"] \\
P'_1\|A^{i''}\|A_c\|B_c \arrow[d, "a"] & & X\|B_c \arrow[d, "a"] \\
P_1\|A^{i''}\|A_c\|B_c & \not\sim & P_2\|A_c\|B_c
\end{tikzcd}$$

Figure 6.3: Winning strategy for the attacker ($i < j$)

Proof. Assume that (P_1, Δ) and (P_2, Δ) are strongly regular. We construct a BPP process (P, Δ') with

$$\mathcal{C}onst(\Delta') \stackrel{\text{def}}{=} \mathcal{C}onst(\Delta) \cup \{X, A, A_c, B_c, P'_1, P'_2\}$$

and

$$\mathcal{A}ct(\Delta') \stackrel{\text{def}}{=} \mathcal{A}ct(\Delta) \cup \{a, b\}$$

where $X, A, A_c, B_c, P'_1, P'_2$ are new process constants and a, b are new actions. We define $\Delta' \stackrel{\text{def}}{=} \Delta \cup \Delta^1 \cup \Delta^2$ where the set of rewrite rules Δ^1 is given by

$$\begin{array}{llll}
X \xrightarrow{b} X\|A & A \xrightarrow{a} \epsilon & A_c \xrightarrow{a} A_c & B_c \xrightarrow{b} B_c \\
X \xrightarrow{a} P'_1\|A_c & X \xrightarrow{a} P_1\|A_c & P'_1 \xrightarrow{a} P_1 &
\end{array}$$

and Δ^2 is given by

$$X \xrightarrow{a} P'_2\|A_c \qquad X \xrightarrow{a} P_2\|A_c \qquad P'_2 \xrightarrow{a} P_2.$$

Let $P \stackrel{\text{def}}{=} X\|B_c$.

Lemma 6.6. If $(P_1, \Delta) \not\sim (P_2, \Delta)$ then (P, Δ') is not strongly regular.

Proof. Let $(P_1, \Delta) \not\sim (P_2, \Delta)$. For simplicity (and without loss of generality) we assume that $P_1 \not\sim \epsilon$ and $P_2 \not\sim \epsilon$. We demonstrate that there are infinitely many strongly nonbisimilar states reachable from (P, Δ').

Let us consider an infinite number of states of the form $X\|A^i\|B_c$ for any natural number i. Of course $P \longrightarrow^* X\|A^i\|B_c$ and we claim that $(X\|A^i\|B_c, \Delta') \not\sim (X\|A^j\|B_c, \Delta')$ for any $i \neq j$. Without loss of generality assume that $i < j$. The attacker has the following winning strategy (playing only in the second process — see Figure 6.3).

He performs a sequence of j actions a from $X\|A^j\|B_c$, thus reaching a state $X\|B_c$. The defender playing from $X\|A^i\|B_c$ cannot do this sequence of a-actions without using some rule for X. This is because $B_c \stackrel{a}{\not\longrightarrow}$ and A^i can perform at most i a-actions ($i < j$). As we assume that $P_1 \not\sim \epsilon$ and $P_2 \not\sim \epsilon$, process constants P_1 and P_2 cannot appear in the defender's process during the first j rounds, otherwise he loses immediately. So the defender has to make a choice between the rules $X \xrightarrow{a} P_1' \| A_c$ and $X \xrightarrow{a} P_2' \| A_c$ sometime within the first j moves (let us say in round i' where $i' \leq i+1$). Assume that the defender chooses $X \xrightarrow{a} P_1' \| A_c$ — the other case is symmetric. Now, the defender must perform $j - i'$ of a-actions by using the rules $A_c \xrightarrow{a} A_c$ or $A \xrightarrow{a} \epsilon$.

After j rounds the resulting states are $P_1' \| A^{i''} \| A_c \| B_c$ for $i'' \leq i - i' + 1$, and $X \| B_c$. The attacker wins by performing the move $X \| B_c \xrightarrow{a} P_2 \| A_c \| B_c$. Again, since we assume that $P_2 \not\sim \epsilon$ the defender has to answer with $P_1' \| A^{i''} \| A_c \| B_c \xrightarrow{a} P_1 \| A^{i''} \| A_c \| B_c$. The attacker has now a winning strategy from $P_1 \| A^{i''} \| A_c \| B_c$ and $P_2 \| A_c \| B_c$: the fact that $P_1 \not\sim P_2$ and that the actions a and b are fresh ones implies that $P_1 \| A^{i''} \| A_c \| B_c \not\sim P_2 \| A_c \| B_c$. $\square$

Lemma 6.7. If $(P_1, \Delta) \sim (P_2, \Delta)$ then (P, Δ') is strongly regular.

Proof. Assume that $(P_1, \Delta) \sim (P_2, \Delta)$ which implies that $(P, \Delta') \sim (P, \Delta'')$ where $\Delta'' \overset{\text{def}}{=} \Delta' \smallsetminus \Delta^2$. It is enough to show that (P, Δ'') is strongly regular. Observe that $(A^i \| A_c, \Delta'') \sim (A_c, \Delta'')$ for any i such that $0 \leq i$. Then also $(P_1 \| A^i \| A_c \| B_c, \Delta'') \sim (P_1 \| A_c \| B_c, \Delta'')$ and $(P_1' \| A^i \| A_c \| B_c, \Delta'') \sim (P_1' \| A_c \| B_c, \Delta'')$ for any i such that $0 \leq i$. This implies that

$$(X \| A^i \| B_c, \Delta'') \sim (P_1' \| A_c \| B_c, \Delta'') \tag{6.1}$$

for any i such that $0 \leq i$. Since (P_1, Δ) is a strongly regular process then $(P_1' \| A_c \| B_c, \Delta'')$ is also strongly regular. This by using (6.1) in particular gives that $(X \| A^0 \| B_c, \Delta'') = (X \| B_c, \Delta'') = (P, \Delta'')$ is strongly regular. $\square$

Theorem 6.3 follows from Lemma 6.6 and Lemma 6.7. $\square$

Theorem 6.4. Strong regularity of BPP is PSPACE-hard.

Proof. By Theorem 6.1, Remark 6.1 and Theorem 6.3. $\square$

6.4.2 Basic Process Algebra

Theorem 6.5 (Reduction from bisimilarity to regularity).
Let (P_1, Δ) and (P_2, Δ) be strongly regular BPA processes. We can construct in polynomial time a BPA process (P, Δ') such that

$$(P_1, \Delta) \sim (P_2, \Delta) \quad \text{if and only if} \quad (P, \Delta') \text{ is strongly regular.}$$

Proof. Assume that (P_1, Δ) and (P_2, Δ) are strongly regular BPA processes such that Δ contains no deadlocks (if Δ contains deadlocks, we can use deadlock elimination technique from [169]).

We construct a BPA process (P, Δ') with

$$\mathcal{C}onst(\Delta') \overset{\text{def}}{=} \mathcal{C}onst(\Delta) \cup \{X, A, C, S, P_1', P_2'\}$$

and

$$\mathcal{A}ct(\Delta') \stackrel{\text{def}}{=} \mathcal{A}ct(\Delta) \cup \{a, s\}$$

where X, A, C, S, P_1', P_2' are new process constants and a, s are new actions. We define $\Delta' \stackrel{\text{def}}{=} \Delta \cup \Delta^1 \cup \Delta^2$ where the set of transition rules Δ^1 is given by

$$\begin{array}{llll}
X \xrightarrow{a} X.A & X \xrightarrow{a} \epsilon & A \xrightarrow{a} \epsilon & S \xrightarrow{s} S \\[1em]
A \xrightarrow{a} P_1'.S & A \xrightarrow{a} P_1.S & P_1' \xrightarrow{a} P_1' & P_1' \xrightarrow{a} P_1 \\
X \xrightarrow{a} P_1'.S & X \xrightarrow{a} P_1.S & & \\
C \xrightarrow{a} P_1'.S & C \xrightarrow{a} P_1.S & &
\end{array}$$

and Δ^2 is given by

$$\begin{array}{llll}
A \xrightarrow{a} P_2'.S & A \xrightarrow{a} P_2.S & P_2' \xrightarrow{a} P_2' & P_2' \xrightarrow{a} P_2 \\
X \xrightarrow{a} P_2'.S & X \xrightarrow{a} P_2.S & & \\
C \xrightarrow{a} P_2'.S & C \xrightarrow{a} P_2.S. & &
\end{array}$$

Let $P \stackrel{\text{def}}{=} X.C$.

Lemma 6.8. If $(P_1, \Delta) \not\sim (P_2, \Delta)$ then (P, Δ') is not strongly regular.

Proof. Let $(P_1, \Delta) \not\sim (P_2, \Delta)$. Without loss of generality assume that $P_1 \not\sim \epsilon$ and $P_2 \not\sim \epsilon$. We show that there are infinitely many strongly nonbisimilar states reachable from (P, Δ'). Consider the states of the form $A^i.C$ for any natural number i. Of course, $P \longrightarrow^* A^i.C$. In order to prove that (P, Δ') is not strongly regular, it is enough to show that $(A^i.C, \Delta') \not\sim (A^j.C, \Delta')$ for any $i < j$. The next paragraph describes attacker's winning strategy from the states $A^i.C$ and $A^j.C$.

The attacker is playing only in the second process $A^j.C$. He performs a sequence of actions a of length j by using the rule $A \xrightarrow{a} \epsilon$ and reaches the state C. By examining all possible moves of the defender from the process $A^i.C$, we get that a rule different from $A \xrightarrow{a} \epsilon$ must be used within the first j rounds since $i < j$. Using the assumption that $P_1 \not\sim \epsilon$ and $P_2 \not\sim \epsilon$ we derive that only four types of states (reachable by the defender from $A^i.C$ after j rounds) must be considered. Namely

- $P_1'.S.A^{i'}.C$ for some i', $0 \le i' < i$
- $P_2'.S.A^{i'}.C$ for some i', $0 \le i' < i$
- $P_1'.S$ or
- $P_2'.S$.

Notice that $S.\alpha \sim S$ for any process expression α, which in particular means that $P_1'.S.A^{i'}.C \sim P_1'.S$ and $P_2'.S.A^{i'}.C \sim P_2'.S$. Hence it is enough to examine only the states $P_1'.S$ and $P_2'.S$. Let us consider the bisimulation game continuing from the pair of states $P_1'.S$ and C — the other case (from states $P_2'.S$ and C) is symmetric. The attacker wins by performing the move $C \xrightarrow{a} P_2.S$. The defender has to answer by $P_1'.S \xrightarrow{a} P_1.S$ since the move $P_1'.S \xrightarrow{a} P_1'.S$ means an immediate loss for the defender (we assume that $P_1 \not\sim \epsilon$ and $P_2 \not\sim \epsilon$). Now the resulting states after $j+1$ rounds are $P_1.S$ and $P_2.S$. The attacker has a winning strategy from this pair by our assumption that $P_1 \not\sim P_2$ and by the fact that $S \xrightarrow{s} S$ is the only rewrite rule for S and the action s is a fresh one. □

Lemma 6.9. If $(P_1, \Delta) \sim (P_2, \Delta)$ then (P, Δ') is strongly regular.

Proof. Assume that $(P_1, \Delta) \sim (P_2, \Delta)$ which implies that $(P, \Delta') \sim (P, \Delta'')$ where $\Delta'' \overset{\text{def}}{=} \Delta' \smallsetminus \Delta^2$ (here the assumption that Δ contains no deadlocks in necessary). It is enough to show that (P, Δ'') is strongly regular. In what follows we often use (without explicitly mentioning it) the fact that $S.\alpha \sim S$ for any process expression α.

Let us first observe that $(C, \Delta'') \sim (P_1'.S, \Delta'')$ which implies that also $(A^i.C, \Delta'') \sim (P_1'.S, \Delta'')$ for all $i \geq 0$. Using this fact we get that

$$(X.A^i.C, \Delta'') \sim (P_1'.S, \Delta'') \tag{6.2}$$

for all $i \geq 0$. Recall that (P_1, Δ) is a strongly regular process. It is easily seen now that $(P_1'.S, \Delta'')$ is also a strongly regular process. Hence (6.2) in particular gives that $(X.A^0.C, \Delta'') = (X.C, \Delta'') = (P, \Delta'')$ is strongly regular. □

Theorem 6.5 follows from Lemma 6.8 and Lemma 6.9. □

Theorem 6.6. Strong regularity of BPA is PSPACE-hard.

Proof. By Theorem 6.2, Remark 6.4 and Theorem 6.5. □

6.4.3 Normed Processes

In this subsection we show that under the condition of normedness, strong regularity of BPA and BPP are complete problems for nondeterministic logarithmic space (NL). Kučera in [106] argues that strong regularity of BPA and BPP is decidable in polynomial time but it is easy to see that a test whether a BPA (BPP) process contains an *accessible* and *growing* process constant (a condition equivalent to regularity) can be performed even in nondeterministic logarithmic space. In [107] the previous results are extended to normed PA processes, and again it can be shown that the decision algorithm for strong regularity of normed PA can be implemented in NL.

Theorem 6.7. Strong regularity of normed BPA and normed BPP is NL-hard.

Proof. In order to prove NL-hardness, we reduce the reachability problem for directed acyclic graphs (for NL-completeness see e.g. [145]) to strong regularity checking of normed BPA (BPP).

Problem:	Reachability for directed acyclic graphs
Instance:	A directed acyclic graph $G = (V, E)$ such that $V = \{v_1, \ldots, v_n\}$, $1 \leq n$, and $E \subseteq V \times V$.
Question:	Is it the case that $(v_1, v_n) \in E^*$ where E^* is the reflexive and transitive closure of E?

Let $G = (V, E)$ be an instance of the reachability problem for acyclic directed graphs. For $u \in V$ we define its out-degree by

$$u^+ \overset{\text{def}}{=} |\{v \in V \mid (u, v) \in E\}|$$

and without loss of generality assume that $v_1^+, v_n^+ > 0$. Let Δ be a finite-state system where $\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{X_u \mid u \in V \ \wedge \ u^+ > 0\} \cup \{X\}$, $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{a\}$ and

$$\begin{aligned} \Delta \overset{\text{def}}{=} \ & \{X_u \xrightarrow{a} X_v \mid (u, v) \in E \ \wedge \ v^+ > 0\} \ \cup \\ & \{X_u \xrightarrow{a} \epsilon \mid (u, v) \in E \ \wedge \ v^+ = 0\} \ \cup \\ & \{X_{v_n} \xrightarrow{a} X\}. \end{aligned}$$

Obviously, $(v_1, v_n) \in E^*$ if and only if $X_{v_1} \longrightarrow^* X$. Let us define a BPA system

$$\Delta_1 \overset{\text{def}}{=} \Delta \ \cup \ \{X \xrightarrow{a} X.X, \ X \xrightarrow{a} \epsilon\}$$

and a BPP system

$$\Delta_2 \overset{\text{def}}{=} \Delta \ \cup \ \{X \xrightarrow{a} X \| X, \ X \xrightarrow{a} \epsilon\}.$$

It is an easy observation that (X, Δ_1) and (X, Δ_2) are normed and nonregular processes. This implies that (X_{v_1}, Δ_1) and (X_{v_1}, Δ_2) are also normed processes (G is acyclic) such that $(v_1, v_n) \in E^*$ iff (X_{v_1}, Δ_1) is not strongly regular, and $(v_1, v_n) \in E^*$ iff (X_{v_1}, Δ_2) is not strongly regular. Recall that NL=co-NL (see e.g. [145]). Hence the problems of strong regularity for normed BPA and BPP are NL-hard (our reductions are obviously in logarithmic space). $\square$

6.5 Concluding Remarks

We have shown that the strong bisimilarity and regularity problems for BPA and BPP are PSPACE-hard. Our proofs are done by reduction from the problem of quantified satisfiability (QSAT). The general idea (Section 6.2) for generating quantified instances of QSAT applies to both BPA and BPP. However, the proofs for BPA and BPP differ in the method of checking that all clauses are indeed satisfied. This is due to the fact that BPP enables parallel access to all process constants contained in the current state whereas BPA does not.

An interesting observation is that only one unnormed process constant (namely S) is used in the hardness proofs for BPA. In contrast, the hardness proofs for strong bisimilarity of BPP (see [124] and Subsection 6.3.1) require a polynomial number of unnormed process constants.

We claim that the existential quantification technique can be used in similar contexts. As we pointed out in [173], there are several applications of the existential quantification technique. The technique is beneficial for e.g. showing PSPACE-hardness of weak bisimilarity for normed and regular BPA, and PSPACE-hardness of strong bisimilarity for normed and regular PDA. The second result for normed PDA was recently improved even to EXPTIME [112] — again using the existential quantification technique.

The usefulness of the main idea of existential quantification will be demonstrated also in the following chapter where we prove that weak bisimilarity problems for PDA and PA are undecidable. The technique of existential quantification brings surprisingly highly understandable (we hope) proofs of these results.

6.6 Bibliographical Remarks

This chapter is based on the technical report “Strong Bisimilarity of Simple Process Algebras: Complexity Lower Bounds” [179]. The report is an extended and unified version of two papers: “Strong Bisimilarity and Regularity of Basic Process Algebra is PSPACE-Hard” [178] and “Strong Bisimilarity and Regularity of Basic Parallel Processes is PSPACE-Hard” [177].

Chapter 7

Undecidability of Weak Bisimilarity

In this chapter we prove undecidability of weak bisimilarity for pushdown processes and PA-processes. The existential quantification technique introduced before will be useful also in this chapter.

In case of PDA, we reduce the halting problem of 2-counter Minsky machines to weak bisimilarity checking. Here the existential quantification technique enables to rearrange stack contents so that we have an access to both the counters stored on the stack. Moreover, we argue that weak bisimilarity is highly undecidable (outside of the arithmetical hierarchy of undecidable problems) even for a restricted subclass of normed PDA. One application of the presented reduction is a proof of high undecidability of strong bisimilarity for prefix-recognizable graphs.

In case of PA, we reduce Post's correspondence problem to weak bisimilarity checking. The existential quantification technique makes possible to generate arbitrarily long solutions of Post's correspondence problem. However, decidability of weak bisimilarity for normed PA-processes still remains open. It is also unclear whether weak bisimilarity of unnormed PA-processes lies inside the arithmetical hierarchy or not.

7.1 Motivation

Strong bisimilarity is decidable for basic process algebra [46] and basic parallel processes [43], two basic models of purely sequential, respectively parallel, computations. There are even polynomial time algorithms for *normed* subclasses of BPA and BPP [74, 75]. The techniques used to show the results are quite involved and exploit nice structural properties of these simple process algebras.

The answers to the strong bisimilarity problems for processes generated by pushdown systems are even more involved than those for BPA and BPP. A pushdown process can be seen as a BPA process extended with a finite-state control unit. Let us recall that from the language point of view, there is no difference between PDA and BPA since both formalisms describe the class of context-free languages.

On the other hand the situation is different when considering strong bisimilarity as the equivalence relation. The PDA class is strictly more expressive than BPA w.r.t. strong (and weak) bisimilarity, and hence the decidability problems are more difficult to handle. Nevertheless, Stirling proved decidability of strong bisimilarity for normed PDA [184] and the same question for the whole class of PDA was positively answered by Sénizergues [163].

Similar extension of BPP with finite-state control unit defines the class of parallel PDA (PPDA), a strict subclass of Petri nets. However, strong bisimilarity is still undecidable for PPDA [34, 97].

Another extension of BPA and BPP is simply by considering PA, the algebra that contains both sequential and parallel operator. In this case the problem of strong bisimilarity is open, however, for the normed subclass it is decidable in 2-NEXPTIME [73].

Let us now draw our attention to weak bisimilarity. The positive development in strong bisimilarity checking for many classes of infinite-state systems had led to the hope that extending the existing techniques to the case of weak bisimilarity might be a feasible step. Some of the recent results, however, contradict this hope.

Opposed to the fact that strong bisimilarity is decidable between Petri nets and finite-state systems [96], Jančar and Esparza proved in [93] that weak bisimilarity is undecidable. Similarly, strong bisimilarity is decidable for pushdown processes (PDA) [163], whereas weak bisimilarity will be shown undecidable in this chapter. Strong bisimilarity of Petri nets is undecidable [90], however, it is at the first level of the arithmetical hierarchy (Π^0_1-complete). On the other hand, weak bisimilarity of Petri nets lies beyond the arithmetical hierarchy [89]. In this chapter we also prove that weak bisimilarity of PA-processes is undecidable. Since PA allows neither communication nor global-state control, the proof is more difficult than for PDA and PN: the undecidability argument for PDA uses a finite-state control unit and the proof for PN relies on the possibility of communication.

The situation appears slightly more promising on the second level of the PRS-hierarchy. Decidability of weak bisimilarity for BPA and BPP are well known open problems. The issues are open even for normed BPA and BPP. However, some partial positive results were achieved e.g. in [72, 187], and the technique of Jančar [92] may give a positive answer for the whole class of BPP. It is also known that weak bisimilarity is decidable for *deterministic* PDA (follows from [162] as mentioned in [125]).

7.2 Pushdown Processes

In this section we provide an evidence of high undecidability of weak bisimilarity for normed PDA. The proof technique is based on an effective encoding of the halting problem for 2-counter Minsky machines [136] into the bisimilarity checking problem of a pair of pushdown processes. We use again the game-theoretic characterization of weak bisimilarity to make the proof more understandable. The intuition of our encoding is that a configuration of a Minsky machine, consisting of an instruction label and the values of counters, is represented by a pair of pushdown processes. In each process the label is remembered in the control state and the values of counters are stored in the stack. The problem is, of course, that we have only sequential access to the stack but we need to enable (at least a limited) parallel access to both counters. The key idea is a technique how to manage these stack contents in such a way that the players faithfully simulate the computation of the Minsky machine. The goal is to establish that the attacker has a winning strategy in the bisimulation game iff the machine halts, or equivalently that the defender has a winning strategy iff the machine diverges. The technique of existential quantification will be used to show this.

Definition 7.1 (Minsky machine with two counters).
A *Minsky machine* R with two counters c_1 and c_2 is a finite sequence

$$R = (L_1 : I_1,\quad L_2 : I_2,\quad \ldots,\quad L_{n-1} : I_{n-1},\quad L_n : \texttt{halt})$$

where $n \geq 1$, $L_1, \ldots, L_n$ are pairwise different *labels*, and $I_1, \ldots, I_{n-1}$ are *instructions* of the following two types:

- *increment:*

$$c_r := c_r + 1\texttt{;}\ \texttt{goto}\ L_j$$

- *test and decrement:*

$$\texttt{if}\ c_r = 0\ \texttt{then}\ \texttt{goto}\ L_j\ \texttt{else}\ c_r := c_r - 1\texttt{;}\ \texttt{goto}\ L_k$$

where $1 \leq r \leq 2$ and $1 \leq j, k \leq n$.

A *configuration* of a Minsky machine R is a triple (L_i, v_1, v_2) where L_i is the instruction label ($1 \leq i \leq n$), and $v_1, v_2 \in \mathbb{N}_0$ are nonnegative integers representing the values of counters c_1 and c_2, respectively. Let $\mathcal{C}onf$ be the set of all configurations of R. The transition relation $\hookrightarrow\ \subseteq \mathcal{C}onf \times \mathcal{C}onf$ between configurations is defined in the obvious and natural way. We remind the reader of the fact that the computation of the machine R is deterministic, i.e., if $c \hookrightarrow d$ and $c \hookrightarrow e$ then $d = e$ for all $c, d, e \in \mathcal{C}onf$.

It is a well known fact that the problem whether a Minsky machine R *halts* with the initial counter values set to zero (in other words the problem whether $(L_1, 0, 0) \hookrightarrow^* (L_n, v_1, v_2)$ for some $v_1, v_2 \in \mathbb{N}_0$) is undecidable [136]. If R does not halt we say that it *diverges*.

Our aim is to show that there is an effective construction such that given a Minsky machine R it defines a pushdown automaton Δ and a pair of processes $p_1\alpha_1$ and $p_2\alpha_2$ with the property that R halts if and only if $(p_1\alpha_1, \Delta) \not\approx (p_2\alpha_2, \Delta)$. This proves that weak bisimilarity of pushdown processes is an undecidable problem.

Let us fix a Minsky machine

$$R = (L_1 : I_1,\quad L_2 : I_2,\quad \ldots,\quad L_{n-1} : I_{n-1},\quad L_n : \texttt{halt}).$$

We construct Δ in stages. First, we define the sets of control states, stack symbols and actions. Let $\mathcal{I}nc \overset{\text{def}}{=} \{i \mid 1 \leq i < n \text{ and } I_i \text{ is of the type `increment'}\}$ and $\mathcal{D}ec \overset{\text{def}}{=} \{i \mid 1 \leq i < n \text{ and } I_i \text{ is of the type `test and decrement'}\}$.

$$\begin{aligned}
Q &\overset{\text{def}}{=} \{\mathsf{equal}, \mathsf{equal}_1, \mathsf{equal}_2, \mathsf{empty}_1, \mathsf{empty}_2, \mathsf{empty}'_1, \mathsf{empty}'_2\} \cup \\
&\quad \bigcup_{i \in \mathcal{I}nc} \{p_i, p'_i\} \cup \bigcup_{i \in \mathcal{D}ec} \{p_i, p'_i, u_i, u'_i, q_i, q'_i, t_i, t'_i\} \cup \{p_n, p'_n\} \\
\Gamma &\overset{\text{def}}{=} \{C_1, C_2, S\} \\
\mathcal{A}ct &\overset{\text{def}}{=} \{a, b, c, d, e, c_1, c_2, c'_1, c'_2, \mathsf{halt}, \tau\}
\end{aligned}$$

The intuition is that a configuration $(L_i, v_1, v_2) \in \mathcal{C}onf$ is represented by a pair of processes $p_i\gamma S$ and $p'_i\gamma' S$ where $\gamma, \gamma' \in \{C_1, C_2\}^*$ such that the number of occurrences of C_1 and C_2 in γ (and also in γ') is equal to v_1 and v_2, respectively. Using this representation, our task is now to design rewrite rules to simulate step by step the computation of R. Let us define formally a mapping $\mathit{value} : \{C_1, C_2\}^* \to \mathbb{N}_0 \times \mathbb{N}_0$ by the following inductive definition (the operation of addition is component-wise).

$$\begin{aligned}
\mathit{value}(\epsilon) &\overset{\text{def}}{=} (0, 0) \\
\mathit{value}(C_1\gamma) &\overset{\text{def}}{=} \mathit{value}(\gamma) + (1, 0) && \text{for all } \gamma \in \{C_1, C_2\}^* \\
\mathit{value}(C_2\gamma) &\overset{\text{def}}{=} \mathit{value}(\gamma) + (0, 1) && \text{for all } \gamma \in \{C_1, C_2\}^*
\end{aligned}$$

As a part of the bisimulation game we will find useful the following rewrite rules which enable to check whether two given stacks contain the same number of occurrences of C_1 and C_2. In the rules below X ranges over the set $\{C_1, C_2, S\}$.

$$\begin{array}{ll}
\mathsf{equal}\ X \xrightarrow{a} \mathsf{equal}_1\ X & \mathsf{equal}\ X \xrightarrow{b} \mathsf{equal}_2\ X \\[1em]
\mathsf{equal}_1\ C_1 \xrightarrow{c_1} \mathsf{equal}_1 & \mathsf{equal}_1\ C_2 \xrightarrow{\tau} \mathsf{equal}_1 \\
\mathsf{equal}_2\ C_2 \xrightarrow{c_2} \mathsf{equal}_2 & \mathsf{equal}_2\ C_1 \xrightarrow{\tau} \mathsf{equal}_2
\end{array}$$

Proposition 7.1. Let $\gamma, \gamma' \in \{C_1, C_2\}^*$. Then

$$\mathsf{equal}\ \gamma S \approx \mathsf{equal}\ \gamma' S \quad \text{iff} \quad \mathit{value}(\gamma) = \mathit{value}(\gamma').$$

Proof. Easy. □

We continue by defining further rewrite rules to check whether the number of occurences of C_1 (or C_2) is zero.

$$\begin{array}{ll}
\mathsf{empty}_1\ C_1 \xrightarrow{c_1} \mathsf{empty}_1 & \mathsf{empty}_1\ C_2 \xrightarrow{c_2} \mathsf{empty}_1 \\
\mathsf{empty}'_1\ C_1 \xrightarrow{c'_1} \mathsf{empty}'_1 & \mathsf{empty}'_1\ C_2 \xrightarrow{c_2} \mathsf{empty}'_1 \\[1em]
\mathsf{empty}_2\ C_1 \xrightarrow{c_1} \mathsf{empty}_2 & \mathsf{empty}_2\ C_2 \xrightarrow{c_2} \mathsf{empty}_2 \\
\mathsf{empty}'_2\ C_1 \xrightarrow{c_1} \mathsf{empty}'_2 & \mathsf{empty}'_2\ C_2 \xrightarrow{c'_2} \mathsf{empty}'_2
\end{array}$$

$$\begin{tikzcd}
p_i\gamma S \arrow[d, "a"] & p_i'\gamma' S \arrow[d, "a"] \\
p_jC_r\gamma S & p_j'C_r\gamma' S
\end{tikzcd}$$

Figure 7.1: Instruction L_i: $c_r := c_r + 1$; goto L_j

Proposition 7.2. Let $\gamma, \gamma' \in \{C_1, C_2\}^*$ be such that $\mathit{value}(\gamma) = \mathit{value}(\gamma') = (v_1, v_2)$ for some $v_1, v_2 \in \mathbb{N}_0$. Let $r \in \{1, 2\}$. Then

$$\mathsf{empty}_r\ \gamma S \approx \mathsf{empty}'_r\ \gamma' S \quad \text{iff} \quad v_r = 0.$$

Proof. Easy. □

Let us now define the rewrite rules that are connected with the increment instructions of R. Assume again that X ranges over the set $\{C_1, C_2, S\}$. For all $i \in \mathcal{I}nc$ such that I_i is of the type

$$L_i\texttt{:}\quad c_r := c_r + 1\texttt{; goto } L_j$$

where $1 \leq j \leq n$ and $1 \leq r \leq 2$, we add the following two rules.

$$p_iX \xrightarrow{a} p_jC_rX \qquad p_i'X \xrightarrow{a} p_j'C_rX$$

Lemma 7.1. Let $(L_i, v_1, v_2) \in \mathcal{C}onf$ be such that I_i is the 'increment' instruction and $(L_i, v_1, v_2) \hookrightarrow (L_j, v_1', v_2')$. Let $\gamma, \gamma' \in \{C_1, C_2\}^*$ be such that $\mathit{value}(\gamma) = \mathit{value}(\gamma') = (v_1, v_2)$. There is a unique continuation of the bisimulation game from the pair $p_i\gamma S$ and $p_i'\gamma' S$ such that after one round the players reach the pair $p_j\overline{\gamma}S$ and $p_j'\overline{\gamma'}S$ satisfying $\mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'}) = (v_1', v_2')$.

Proof. Obvious — see Figure 7.1. □

We proceed by giving the rules for the 'test and decrement' instructions. For all $i \in \mathcal{D}ec$ such that I_i is of the type

$$L_i\texttt{:}\quad \texttt{if } c_r = 0 \texttt{ then goto } L_j \texttt{ else } c_r := c_r - 1\texttt{; goto } L_k$$

where $1 \leq j, k \leq n$ and $1 \leq r \leq 2$, we define the rewrite rules in three parts. The intuitive meaning is that if $v_r \neq 0$ and the stacks γS and $\gamma' S$ contain on their tops the symbol C_r, we can do immediately the branching according to the rules defined later in the third part. However, if it is not the case, the first two parts of the rewrite rules enable the defender to rearrange the stack contents (while preserving the number of occurrences of C_1 and C_2) in such a way that C_r will appear as the first symbol on the stacks. Recall that X ranges over the set $\{C_1, C_2, S\}$.

$$\begin{array}{lll}
p_iX \xrightarrow{a} q_iX & p_iX \xrightarrow{a} u'_iX & \\
& & \\
p'_iX \xrightarrow{a} u'_iX & u'_iX \xrightarrow{\tau} q'_iX & u'_iX \xrightarrow{e} u'_iX \\
u'_iC_1 \xrightarrow{\tau} u'_i & u'_iC_2 \xrightarrow{\tau} u'_i & \\
u'_iX \xrightarrow{\tau} u'_iC_1X & u'_iX \xrightarrow{\tau} u'_iC_2X & \\
& & \\
q_iX \xrightarrow{c} \mathsf{equal}\ X & q'_iX \xrightarrow{c} \mathsf{equal}\ X &
\end{array}$$

Consider a bisimulation game played from $p_i\gamma S$ and $p'_i\gamma' S$. The purpose of the previously defined rules is to enable the defender to rearrange the sequence of C_1 and C_2 in γ'. Details are discussed in the proof of Lemma 7.2, here we give only a short description. If the attacker plays $p_i\gamma S \xrightarrow{a} q_i\gamma S$, the defender must answer by $p'_i\gamma' S \overset{a}{\Longrightarrow} q'_i\overline{\gamma'}S$ for some $\overline{\gamma'} \in \{C_1, C_2\}^*$. Now the attacker can check the invariant that $\mathit{value}(\gamma) = \mathit{value}(\overline{\gamma'})$ by using the rules $q_iX \xrightarrow{c} \mathsf{equal}\ X$ and $q'_iX \xrightarrow{c} \mathsf{equal}\ X$.

$$\begin{array}{lll}
q'_iX \xrightarrow{a} t'_iX & q'_iX \xrightarrow{a} u_iX & \\
& & \\
q_iX \xrightarrow{a} u_iX & u_iX \xrightarrow{\tau} t_iX & u_iX \xrightarrow{e} u_iX \\
u_iC_1 \xrightarrow{\tau} u_i & u_iC_2 \xrightarrow{\tau} u_i & \\
u_iX \xrightarrow{\tau} u_iC_1X & u_iX \xrightarrow{\tau} u_iC_2X & \\
& & \\
t_iX \xrightarrow{c} \mathsf{equal}\ X & t'_iX \xrightarrow{c} \mathsf{equal}\ X &
\end{array}$$

These rules are completely symmetric to the previous ones. In the bisimulation game starting from $q_i\gamma S$ and $q'_i\overline{\gamma'}S$, if the attacker plays $q'_i\overline{\gamma'}S \xrightarrow{a} t'_i\overline{\gamma'}S$, the defender must choose some $\overline{\gamma} \in \{C_1, C_2\}^*$ and play $q_i\gamma S \overset{a}{\Longrightarrow} t_i\overline{\gamma}S$. The attacker can again check whether $\mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'})$. The current states become $t_i\overline{\gamma}S$ and $t'_i\overline{\gamma'}S$ satisfying $\mathit{value}(\gamma) = \mathit{value}(\gamma') = \mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'})$.

The third part of the rewrite rules defined below is here to perform a branching according to whether C_r occurs in γ and γ' or not. The correctness is discussed later.

$$\begin{array}{ll}
t_iC_r \xrightarrow{a} p_k & t'_iC_r \xrightarrow{a} p'_k \\
& \\
t_iC_{3-r} \xrightarrow{b} p_jC_{3-r} & t'_iC_{3-r} \xrightarrow{b} p'_jC_{3-r} \\
t_iS \xrightarrow{b} p_jS & t'_iS \xrightarrow{b} p'_jS \\
& \\
t_iC_{3-r} \xrightarrow{d} \mathsf{empty}_r\ C_{3-r} & t'_iC_{3-r} \xrightarrow{d} \mathsf{empty}'_r\ C_{3-r}
\end{array}$$

Finally, we add one extra rule to distinguish whether the last instruction `halt` was reached. Recall that X ranges over the set $\{C_1, C_2, S\}$.

$$p_nX \xrightarrow{\mathsf{halt}} p_nX$$

Lemma 7.2. Let $(L_i, v_1, v_2) \in \mathcal{C}\mathit{onf}$ be such that I_i is the 'test and decrement' instruction

$$\begin{tikzcd}
p_i\gamma S \arrow[d, "a"] & & p'_i\gamma' S \arrow[d, Rightarrow, "a"] & \\
q_i\gamma S \arrow[r, "c"] \arrow[d, Rightarrow, "a"] & \mathsf{equal}\ \gamma S & q'_i\overline{\gamma'}S \arrow[r, "c"] \arrow[d, "a"] & \mathsf{equal}\ \overline{\gamma'}S \\
t_i\overline{\gamma}S \arrow[r, "c"] & \mathsf{equal}\ \overline{\gamma}S & t'_i\overline{\gamma'}S \arrow[r, "c"] & \mathsf{equal}\ \overline{\gamma'}S
\end{tikzcd}$$

Figure 7.2: Instruction 'test and decrement' — first two rounds

$$L_i\texttt{:}\quad \texttt{if } c_r = 0 \texttt{ then goto } L_j \texttt{ else } c_r := c_r - 1\texttt{; goto } L_k$$

and let $\gamma, \gamma' \in \{C_1, C_2\}^*$ be such that $\mathit{value}(\gamma) = \mathit{value}(\gamma') = (v_1, v_2)$. Consider a bisimulation game played from the pair

$$p_i\gamma S \quad \text{and} \quad p'_i\gamma' S.$$

a) The attacker has a strategy such that he either wins, or after three rounds the players reach the states

   1. $p_k\overline{\gamma}S$ and $p'_k\overline{\gamma'}S$ — if $v_r \neq 0$ and $(L_i, v_1, v_2)\hookrightarrow(L_k, v'_1, v'_2)$ — where $\mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'}) = (v'_1, v'_2)$, or
   2. $p_j\overline{\gamma}S$ and $p'_j\overline{\gamma'}S$ — if $v_r = 0$ and $(L_i, v_1, v_2)\hookrightarrow(L_j, v_1, v_2)$ — where $\mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'}) = (v_1, v_2)$.

b) The defender has a strategy such that he either wins, or after three rounds the players reach the states

   1. $p_k\overline{\gamma}S$ and $p'_k\overline{\gamma'}S$ — if $v_r \neq 0$ and $(L_i, v_1, v_2)\hookrightarrow(L_k, v'_1, v'_2)$ — where $\mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'}) = (v'_1, v'_2)$, or
   2. $p_j\overline{\gamma}S$ and $p'_j\overline{\gamma'}S$ — if $v_r = 0$ and $(L_i, v_1, v_2)\hookrightarrow(L_j, v_1, v_2)$ — where $\mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'}) = (v_1, v_2)$.

Proof. We begin with part a). First two rounds of the bisimulation game are depicted in Figure 7.2. The game starts from $p_i\gamma S$ and $p'_i\gamma' S$ such that $\mathit{value}(\gamma) = \mathit{value}(\gamma') = (v_1, v_2)$. We show that after two attacker's moves the players either reach a pair $t_i\overline{\gamma}S$ and $t'_i\overline{\gamma'}S$ such that $\overline{\gamma}, \overline{\gamma'} \in \{C_1, C_2\}^*$ and $\mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'}) = (v_1, v_2)$, or the attacker has an immediate winning strategy. The attacker starts by playing $p_i\gamma S \xrightarrow{a} q_i\gamma S$. The defender must respond by playing $p'_i\gamma' S \overset{a}{\Longrightarrow} q'_i\overline{\gamma'}S$ for some $\overline{\gamma'} \in \{C_1, C_2\}^*$ because of the following remark.

Remark 7.1. The defender's $\overset{a}{\Longrightarrow}$-answer must start with the transition $p'_i\gamma' S \xrightarrow{a} u'_i\gamma' S$, followed by a finite number of τ-labelled transitions using the rules that

$$\begin{tikzcd}
t_iC_r\overline{\overline{\gamma}}S \arrow[d, "a"] & & t'_iC_r\overline{\overline{\gamma'}}S \arrow[d, "a"] \\
p_k\overline{\overline{\gamma}}S & & p'_k\overline{\overline{\gamma'}}S
\end{tikzcd}$$

Figure 7.3: Case $v_r \neq 0$, i.e., $(L_i, v_1, v_2) \hookrightarrow (L_k, v'_1, v'_2)$

enable to remove an arbitrary part of the stack $\gamma' S$ and add an arbitrary sequence from the symbols C_1 and C_2. Thus the defender can reach the state $u'_i\overline{\gamma'}S$ for any sequence $\overline{\gamma'} \in \{C_1, C_2\}^*$. Also note that he must finish the sequence of τ-moves by $u'_i\overline{\gamma'}S \xrightarrow{\tau} q'_i\overline{\gamma'}S$. If not, then the attacker has an immediate winning move in the next round by playing $u'_i\overline{\gamma'}S \xrightarrow{e} u'_i\overline{\gamma'}S$ to which the defender has no answer because there is no $\overset{e}{\Longrightarrow}$-move from $q_i\gamma S$.

The bisimulation game continues from the states $q_i\gamma S$ and $q'_i\overline{\gamma'}S$. Whenever $\mathit{value}(\gamma) \neq \mathit{value}(\overline{\gamma'})$ then the attacker plays $q_i\gamma S \xrightarrow{c} \mathsf{equal}\ \gamma S$ to which the defender has only one possible answer $q'_i\overline{\gamma'}S \xrightarrow{c} \mathsf{equal}\ \overline{\gamma'}S$. Now the attacker has a winning strategy because of Proposition 7.1.

Let us so assume that $\mathit{value}(\gamma) = \mathit{value}(\overline{\gamma'})$. In the second round the attacker switches the states and performs the move $q'_i\overline{\gamma'}S \xrightarrow{a} t'_i\overline{\gamma'}S$. The game is now completely symmetric to the situation in the first round. The defender must answer with $q_i\gamma S \overset{a}{\Longrightarrow} t_i\overline{\gamma}S$ for some $\overline{\gamma} \in \{C_1, C_2\}^*$ such that $\mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'}) = (v_1, v_2)$.

In the third round played from $t_i\overline{\gamma}S$ and $t'_i\overline{\gamma'}S$ the attacker's strategy splits into two parts, according to whether $v_r \neq 0$ or $v_r = 0$.

1. Let $v_r \neq 0$ and hence $(L_i, v_1, v_2) \hookrightarrow (L_k, v'_1, v'_2)$. See Figure 7.3.

   – If $\overline{\gamma} = C_r\overline{\overline{\gamma}}$ for some $\overline{\overline{\gamma}}$ then the attacker plays $t_i\overline{\gamma}S \xrightarrow{a} p_k\overline{\overline{\gamma}}S$ and the defender must answer by $t'_i\overline{\gamma'}S \xrightarrow{a} p'_k\overline{\overline{\gamma'}}S$ where $\overline{\gamma'} = C_r\overline{\overline{\gamma'}}$. (If $\overline{\gamma'} = C_{3-r}\overline{\overline{\gamma'}}$ then the attacker wins immediately since $t'_i\overline{\gamma'}S$ cannot perform any $\overset{a}{\Longrightarrow}$-move.) Now the players reached the pair $p_k\overline{\overline{\gamma}}S$ and $p'_k\overline{\overline{\gamma'}}S$ as required. Obviously $\mathit{value}(\overline{\overline{\gamma}}) = \mathit{value}(\overline{\overline{\gamma'}}) = (v'_1, v'_2)$.

   – If $\overline{\gamma} = C_{3-r}\overline{\overline{\gamma}}$ for some $\overline{\overline{\gamma}}$ then the attacker plays $t_i\overline{\gamma}S \xrightarrow{d} \mathsf{empty}_r\ \overline{\gamma}S$ to which the defender has only one possible answer (if any), namely $t'_i\overline{\gamma'}S \xrightarrow{d} \mathsf{empty}'_r\ \overline{\gamma'}S$. Since $\mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'}) = (v_1, v_2)$ and $v_r \neq 0$, the attacker has a winning strategy because of Proposition 7.2.

   – The case $\overline{\gamma} = \epsilon$ is impossible since we assume that $v_r \neq 0$.

2. Let $v_r = 0$ and hence $(L_i, v_1, v_2) \hookrightarrow (L_j, v_1, v_2)$. See Figure 7.4. The assumption $v_r = 0$ implies that $\overline{\gamma}, \overline{\gamma'} \in \{C_{3-r}\}^*$. Hence the attacker can play $t_i\overline{\gamma}S \xrightarrow{b} p_j\overline{\gamma}S$ and the defender has only one answer $t'_i\overline{\gamma'}S \xrightarrow{b} p'_j\overline{\gamma'}S$. The players reached the pair $p_j\overline{\gamma}S$ and $p'_j\overline{\gamma'}S$ as required. Recall that $\mathit{value}(\overline{\gamma}) = \mathit{value}(\overline{\gamma'}) = (v_1, v_2)$.

$$\begin{tikzcd}
t_i\overline{\gamma}S \arrow[r, "d"] \arrow[d, "b"] & \mathsf{empty}_r\ \overline{\gamma}S & t'_i\overline{\gamma'}S \arrow[r, "d"] \arrow[d, "b"] & \mathsf{empty}'_r\overline{\gamma'}S \\
p_j\overline{\gamma}S & & p'_j\overline{\gamma'}S &
\end{tikzcd}$$

Figure 7.4: Case $v_r = 0$, i.e., $(L_i, v_1, v_2)\hookrightarrow(L_j, v_1, v_2)$

Let us now prove part b). First two rounds can be seen again in Figure 7.2. The initial states are $p_i\gamma S$ and $p'_i\gamma' S$ such that $\mathit{value}(\gamma) = \mathit{value}(\gamma') = (v_1, v_2)$. We claim that the defender has a strategy such that he either wins, or after two rounds the players reach the states $t_iC_r^{v_r}C_{3-r}^{v_{3-r}}S$ and $t'_iC_r^{v_r}C_{3-r}^{v_{3-r}}S$ (for definitions of $C_r^{v_r}$ and $C_{3-r}^{v_{3-r}}$ see Notation 1). In the first round the attacker has three possible moves: (i) $p_i\gamma S \xrightarrow{a} q_i\gamma S$, (ii) $p_i\gamma S \xrightarrow{a} u'_i\gamma S$ or (iii) $p'_i\gamma' S \xrightarrow{a} u'_i\gamma' S$. The moves (ii) and (iii) are good for the defender since he can immediately win by playing (ii) $p'_i\gamma' S \overset{a}{\Longrightarrow} u'_i\gamma S$ or (iii) $p_i\gamma S \overset{a}{\Longrightarrow} u'_i\gamma' S$. Obviously, two syntactically equal states are also weakly bisimilar. Hence we can assume that the attacker's first move is $p_i\gamma S \xrightarrow{a} q_i\gamma S$. The defender answers by $p'_i\gamma' S \overset{a}{\Longrightarrow} q'_iC_r^{v_r}C_{3-r}^{v_{3-r}}S$. Recall that $\mathit{value}(\gamma) = (v_1, v_2)$ and thus the attacker loses by taking (i) $q_i\gamma S \xrightarrow{c} \mathsf{equal}\ \gamma S$ or (ii) $q'_iC_r^{v_r}C_{3-r}^{v_{3-r}}S \xrightarrow{c} \mathsf{equal}\ C_r^{v_r}C_{3-r}^{v_{3-r}}S$ as his next move since the defender can respond by playing (i) $q'_iC_r^{v_r}C_{3-r}^{v_{3-r}}S \xrightarrow{c} \mathsf{equal}\ C_r^{v_r}C_{3-r}^{v_{3-r}}S$ or (ii) $q_i\gamma S \xrightarrow{c} \mathsf{equal}\ \gamma S$. The pair of states $\mathsf{equal}\ \gamma S$ and $\mathsf{equal}\ C_r^{v_r}C_{3-r}^{v_{3-r}}S$ is weakly bisimilar because of Proposition 7.1 and the defender has a winning strategy.

From the pair $q_i\gamma S$ and $q'_iC_r^{v_r}C_{3-r}^{v_{3-r}}S$ we have a symmetric situation to the previous one. So after the second round either the defender can win, or he can force the attacker to reach the states $t_iC_r^{v_r}C_{3-r}^{v_{3-r}}S$ and $t'_iC_r^{v_r}C_{3-r}^{v_{3-r}}S$. Now the game splits into two parts according to whether $v_r \neq 0$ or $v_r = 0$.

1. Let $v_r \neq 0$ and hence $(L_i, v_1, v_2)\hookrightarrow(L_k, v'_1, v'_2)$. See Figure 7.3. Then there is a unique continuation of the game reaching the states $p_kC_r^{v_r-1}C_{3-r}^{v_{3-r}}S$ and $p'_kC_r^{v_r-1}C_{3-r}^{v_{3-r}}S$. Obviously $\mathit{value}(C_r^{v_r-1}C_{3-r}^{v_{3-r}}) = (v'_1, v'_2)$.

2. Let $v_r = 0$ and hence $(L_i, v_1, v_2)\hookrightarrow(L_j, v_1, v_2)$. See Figure 7.4. Consider the game starting from $t_iC_{3-r}^{v_{3-r}}S$ and $t'_iC_{3-r}^{v_{3-r}}S$ (note that $C_r^{v_r} = C_r^0$ is the empty string here). There is either a continuation of the game such that the players reach the states $p_jC_{3-r}^{v_{3-r}}S$ and $p'_jC_{3-r}^{v_{3-r}}S$, and $\mathit{value}(C_{3-r}^{v_{3-r}}) = (v_1, v_2)$ — or the attacker performs the $\xrightarrow{d}$-move but then the defender wins because of Proposition 7.2.

□

We arrived at the point where we are ready to prove our main theorem.

Theorem 7.1. Weak bisimilarity of pushdown processes is undecidable.

Proof. Let R be a Minsky machine and let Δ be the pushdown system constructed above. We prove that R halts if and only if $p_1S \not\approx p'_1S$.

Assume that R halts, i.e., $(L_1, 0, 0) \hookrightarrow^* (L_n, v_1, v_2)$ for some $v_1, v_2 \in \mathbb{N}_0$. Then the attacker has a winning strategy starting from p_1S and p'_1S. Using repeatedly Lemma 7.1 and part a) of Lemma 7.2 we can easily see that the attacker either wins, or the players reach the states $p_n\gamma S$ and $p'_n\gamma' S$ for some $\gamma, \gamma' \in \{C_1, C_2\}^*$ such that $\mathit{value}(\gamma) = \mathit{value}(\gamma') = (v_1, v_2)$. From the pair $p_n\gamma S$ and $p'_n\gamma' S$ the attacker immediately wins by playing $p_n\gamma S \xrightarrow{\mathsf{halt}} p_n\gamma S$ to which the defender has no answer from $p'_n\gamma' S$. Hence $p_1S \not\approx p'_1S$.

On the other hand if R diverges, i.e., there is an infinite computation starting from $(L_1, 0, 0)$, the defender has a winning strategy. Using repeatedly Lemma 7.1 and part b) of Lemma 7.2 he can force the attacker to simulate the computation of R in the bisimulation game. Because the computation of R is infinite, so is the bisimulation game starting from p_1S and p'_1S. Since any infinite bisimulation game is won by the defender (Definition 2.7), we get that $p_1S \approx p'_1S$. □

Let us now study the rewrite rules defined above to see whether we can prove even a stronger undecidability result for the normed subclass of pushdown processes. As it can be observed, the pushdown processes p_1S and p'_1S are almost normed. There are only a few exceptions: computations of the pushdown automaton from p_1S and p'_1S can get stuck with nonempty stacks by reaching e.g. the states $p'_n\gamma' S$, equal_1S, equal_2S, empty_1S, or there is an infinite loop where only the increment instructions appear.

It would be easy to fix these problems by adding some extra rules but we didn't want to confuse the reader by mentioning these rules during the development of the undecidability proof. In fact, we can derive undecidability of weak bisimilarity for normed pushdown processes from the following lemma.

Lemma 7.3. Let Δ be a pushdown automaton, and $(p_1\alpha_1, \Delta)$ and $(p_2\alpha_2, \Delta)$ a pair of processes. We can construct in polynomial time a pushdown automaton Δ' and a pair of normed processes $(p_1\alpha'_1, \Delta')$ and $(p_2\alpha'_2, \Delta')$ such that

$$(p_1\alpha_1, \Delta) \approx (p_2\alpha_2, \Delta) \quad \text{if and only if} \quad (p_1\alpha'_1, \Delta') \approx (p_2\alpha'_2, \Delta').$$

Proof. Let Δ be a pushdown automaton with the set of control states Q, stack symbols Γ and actions $\mathcal{A}ct$. We define Δ' with the corresponding sets $Q' \overset{\text{def}}{=} Q \cup \{p_d\}$, $\Gamma' \overset{\text{def}}{=} \Gamma \cup \{D\}$ and $\mathcal{A}ct' \overset{\text{def}}{=} \mathcal{A}ct \cup \{f\}$ such that p_d, D and f are new symbols. In particular, D is the symbol for a new bottom of the stack. Let $\Delta' \overset{\text{def}}{=} \Delta \cup \{pX \xrightarrow{f} p_d \mid p \in Q \text{ and } X \in \Gamma'\} \cup \{p_dX \xrightarrow{\tau} p_d \mid X \in \Gamma'\}$. We define $\alpha'_1 \overset{\text{def}}{=} \alpha_1 D$ and $\alpha'_2 \overset{\text{def}}{=} \alpha_2 D$. Obviously, $(p_1\alpha'_1, \Delta')$ and $(p_2\alpha'_2, \Delta')$ are normed processes. The validity of $(p_1\alpha_1, \Delta) \approx (p_2\alpha_2, \Delta)$ iff $(p_1\alpha'_1, \Delta') \approx (p_2\alpha'_2, \Delta')$ is easy to see from the fact that $(p_d\gamma, \Delta') \approx (p_d\gamma', \Delta')$ for any $\gamma, \gamma' \in \Gamma'^*$. □

Corollary 7.1. Weak bisimilarity of normed pushdown processes is undecidable.

Proof. Immediately from Theorem 7.1 and Lemma 7.3. □

Remark 7.2. Observe that the construction in Lemma 7.3 gives immediately a polynomial time reduction from weak bisimilarity between pushdown processes and finite-state processes to the normed instances of the problems. It is also easy to see that it preserves the property of being weakly regular, i.e., $(p_1\alpha_1, \Delta)$ is weakly regular iff $(p_1\alpha'_1, \Delta')$ is weakly regular.

Remark 7.3. It is obvious that the presented reduction from 2-counter machines to weak bisimilarity of pushdown processes can be extended to work for an arbitrary number of counters and hence weak bisimilarity of normed PDA lies beyond the arithmetical hierarchy: the technique of Jančar [89] for showing high undecidability of weak bisimilarity for Petri nets can be adapted also to our case.

In the rest of this section we investigate seemingly unrelated problem of strong bisimilarity for *prefix-recognizable graphs* [40] (also called type -2 graphs). It is known that monadic second order theory is decidable for prefix-recognizable graphs [41]. Strong bisimilarity checking of prefix-recognizable graphs was stated as an open problem e.g. in [184]. Using the result for PDA presented in this section, we can easily derive undecidability this problem.

Definition 7.2 (Prefix-recognizable graph).
Prefix-recognizable system Δ is a finite set of rules of the form $R_1 \xrightarrow{a} R_2$ where R_1 and R_2 are regular languages over $\mathcal{C}onst$, and $a \in \mathcal{A}ct$. The system Δ determines a *prefix-recognizable graph* (labelled transition system) $T(\Delta) \overset{\text{def}}{=} (S, \mathcal{A}ct, \longrightarrow)$ defined as follows.

$$\begin{array}{rcl} S & \overset{\text{def}}{=} & \mathcal{S}(\mathcal{C}onst) \\ \longrightarrow & \overset{\text{def}}{=} & \{(\alpha.\gamma, a, \beta.\gamma) \mid (R_1 \xrightarrow{a} R_2) \in \Delta \ \wedge \\ & & \qquad\qquad \alpha \in R_1 \ \wedge \ \beta \in R_2 \ \wedge \ \gamma \in \mathcal{S}(\mathcal{C}onst)\} \end{array}$$

States of $T(\Delta)$ are sequential process expressions over $\mathcal{C}onst$ and a rule $R_1 \xrightarrow{a} R_2$ is interpreted as a possibly infinite set of rules $\alpha \xrightarrow{a} \beta$ such that $\alpha \in R_1$ and $\beta \in R_2$.

As remarked by Stirling in [185], whenever a PDA system with control states Q and stack alphabet Γ satisfies

$$\text{if } pX \xrightarrow{\tau} \text{ for } p \in Q \text{ and } X \in \Gamma, \text{ then } pX \not\xrightarrow{a} \text{ for all } a \in \mathcal{A}ct \smallsetminus \{\tau\} \qquad (7.1)$$

then the collapsed PDA graph (i.e. the graph where the weak transitions $\overset{a}{\Longrightarrow}$ for $a \in \mathcal{A}ct$ are considered as single-step transitions and all τ transitions are omitted) is a prefix-recognizable graph.

Notice that the PDA system Δ from the proof of undecidability of weak bisimilarity can be easily modified to satisfy condition (7.1). The only problematic situation is that $u'_iX \xrightarrow{\tau}$ and also $u'_iX \xrightarrow{e} u'_iX$, and similarly $u_iX \xrightarrow{\tau}$ and also $u_iX \xrightarrow{e} u_iX$. It is e.g. enough to replace the rules $u'_iX \xrightarrow{e} u'_iX$ and $u_iX \xrightarrow{e} u_iX$ with $u'_iX \xrightarrow{\tau} pX$ and $u_iX \xrightarrow{\tau} pX$ where p is a newly added state such that $pX \xrightarrow{e} pX$. Hence also weak bisimilarity of PDA systems satisfying condition (7.1) is highly undecidable and we can conclude with the following corollary.

Corollary 7.2. Strong bisimilarity of prefix-recognizable graphs is highly undecidable.

There is even a direct (polynomial time) reduction from weak bisimilarity of PDA (without any restrictions) to strong bisimilarity of prefix-recognizable graphs. The idea is to interpret weak transition relations $\overset{a}{\Longrightarrow}$ as possibly infinite sets of strong transition relations by collapsing all the τ moves.

Theorem 7.2. Weak bisimilarity of PDA is in polynomial time reducible to strong bisimilarity of prefix-recognizable graphs (the size of the description of such a graph includes the sizes of finite automata recognizing the regular sets used in the rules).

Proof. Given a PDA system Δ with control states Q, stack alphabet Γ and the set of actions $\mathcal{A}ct$, we define four sets $pre1(pX, a)$, $post1(pX, a)$, $pre2(pX, a)$ and $post2(pX, a)$ for every $p \in Q$, $X \in \Gamma$ and $a \in \mathcal{A}ct$ as follows.

$$\begin{aligned}
pre1(pX,a) &\overset{\text{def}}{=} \{q\omega \mid q\omega \overset{\tau}{\Longrightarrow} pX\}\\
post1(pX,a) &\overset{\text{def}}{=} \{q\omega \mid pX \overset{a}{\Longrightarrow} q\omega\}\\
pre2(pX,a) &\overset{\text{def}}{=} \{q\omega \mid q\omega \overset{a}{\Longrightarrow} pX\}\\
post2(pX,a) &\overset{\text{def}}{=} \{q\omega \mid pX \overset{\tau}{\Longrightarrow} q\omega\}
\end{aligned}$$

It is a standard result that the set of reachable states of a PDA system forms a regular set [31]. It is then easy to see that $pre1(pX, a)$, $post1(pX, a)$, $pre2(pX, a)$ and $post2(pX, a)$ are also regular languages over $\mathcal{A}ct$, and the corresponding finite-state automata recognizing these languages can be constructed in polynomial time w.r.t. the size of the given PDA system (see [26, 60]). Let us now define a prefix-recognizable graph Δ' consisting of the following rules:

$$\begin{aligned}
pre1(pX,a) &\xrightarrow{a} post1(pX,a) \quad \text{for all } p \in Q,\ X \in \Gamma \text{ and } a \in \mathcal{A}ct,\\
pre2(pX,a) &\xrightarrow{a} post2(pX,a) \quad \text{for all } p \in Q,\ X \in \Gamma \text{ and } a \in \mathcal{A}ct.
\end{aligned}$$

Since the sets Q, Γ and $\mathcal{A}ct$ are finite sets, the system Δ' contains only finitely (in fact only polynomially) many rules.

Let $p_1\alpha_1$ and $p_2\alpha_2$ be a pair of states in $T(\Delta)$. It is a routine exercise to check that $(p_1\alpha_1, \Delta) \approx (p_2\alpha_2, \Delta)$ if and only if $(p_1\alpha_1, \Delta') \sim (p_2\alpha_2, \Delta')$. □

7.3 PA-Processes

In this section we further confirm the inherent complexity of weak bisimilarity by showing its undecidability for PA-processes. The proof is by reduction from Post's Correspondence Problem (PCP). For a given instance of PCP we construct a pair of PA processes that are weakly bisimilar if and only if the PCP instance has a solution. We use again the game-theoretic characterization of weak bisimilarity and combine several techniques to achieve our result.

- The first technique uses the ideas of *existential quantification* as introduced in Chapter 6. In case of weak bisimilarity it moreover provides a technique for generating arbitrarily long sequences of process constants (representing solutions of a given PCP instance).

- The second technique, used by Mayr in [124] and called the *masking technique*, deals with the following phenomenon. Assume that X is an unnormed process constant that performs an action a and becomes X again. Whenever X is added via parallel composition to any process expression γ, it is capable of masking every possible occurrence of the action a in γ.
- Finally, we adapt the technique of *deadlock elimination* from [169] into our context, in order to make the proofs more transparent.

In this section we will also frequently use the following proposition.

Proposition 7.3. Let (E, Δ) and (F, Δ) be a pair of PA-processes. If the defender has a winning strategy from E and F then he also has a winning strategy from $E \| \gamma$ and $F \| \gamma$ for any process expression $\gamma \in \mathcal{G}\big(\mathcal{C}onst(\Delta)\big)$.

Proof. Easy. $\square$

Let us now define Post's correspondence problem and then we describe the reduction. For technical convenience we use the power of deadlocks to design the reduction, however, at the end of this section we discuss a simple technique for deadlock elimination. Thus the undecidability result is valid also for PA-processes without deadlocks.

Definition 7.3 (Post's correspondence problem).
An instance of *Post's correspondence problem (PCP)* is a nonempty alphabet Σ and two lists

$$A = [u_1, \ldots, u_n] \quad \text{and} \quad B = [v_1, \ldots, v_n]$$

where $n > 0$ and $u_k, v_k \in \Sigma^+$ for all k, $1 \le k \le n$. The question is to decide whether the (A, B)-instance has a solution, i.e., whether there is an integer $m \ge 1$ and a sequence of indices $i_1, \ldots, i_m \in \{1, \ldots, n\}$ such that

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}.$$

According to the classical result due to Post, this problem is undecidable [154]. Let us consider an (A, B)-instance of PCP where

$$A = [u_1, \ldots, u_n] \quad \text{and} \quad B = [v_1, \ldots, v_n].$$

We construct a PA system Δ and a pair of processes (P_1, Δ) and (P_2, Δ) such that the (A, B)-instance has a solution if and only if $(P_1, \Delta) \approx (P_2, \Delta)$.

Let $\mathcal{SF}(\alpha)$ denote the set of all suffixes of a sequence $\alpha \in \Sigma^*$, i.e., $\mathcal{SF}(\alpha) \overset{\text{def}}{=} \{\alpha' \in \Sigma^* \mid \exists \alpha'' \in \Sigma^* \text{ such that } \alpha = \alpha''\alpha'\}$. Note that $\epsilon \in \mathcal{SF}(\alpha)$ for any α. We can now define the set of process constants $\mathcal{C}onst(\Delta)$ and actions $\mathcal{A}ct(\Delta)$ by

$$\begin{aligned} \mathcal{C}onst(\Delta) \overset{\text{def}}{=}\ & \{U^{u_k} \mid 1 \le k \le n\} \cup \{V^{v_k} \mid 1 \le k \le n\} \cup \\ & \{T^w \mid w \in \bigcup_{k=1}^{n} \mathcal{SF}(u_k) \cup \bigcup_{k=1}^{n} \mathcal{SF}(v_k)\} \cup \\ & \{X, X', X_1', Y, Y', Y_1, Z, C, C_1, C_2, W, D\} \end{aligned}$$

$$\begin{aligned} \mathcal{A}ct(\Delta) \overset{\text{def}}{=}\ & \{a \mid a \in \Sigma\} \cup \{\iota_k \mid 1 \le k \le n\} \cup \\ & \{x, y, z, c_1, c_2, \tau\}. \end{aligned}$$

Remark 7.4. In what follows D will be a distinguished process constant with no rules associated to it (deadlock). Hence in particular $\alpha.D.\beta \parallel \gamma \;\approx\; \alpha \parallel \gamma$ for any process expressions α, β and γ.

To make the rewrite rules introduced in this section more understandable, we define the system Δ in four stages. It is important to remark here that whenever we define the rules for some process constant $Q \in \mathcal{C}onst(\Delta)$, we always give all the rules for Q at the same stage. Our ultimate goal is to show that $(X\|C,\Delta) \approx (X'\|C,\Delta)$ if and only if the given (A,B)-instance of PCP has a solution. The first part of the system Δ is given by the following rules:

$$\begin{array}{lll}
U^{u_k} \xrightarrow{\iota_k} \epsilon & U^{u_k} \xrightarrow{\tau} T^{u_k} & \text{for all } k \in \{1,\dots,n\} \\
V^{v_k} \xrightarrow{\iota_k} \epsilon & V^{v_k} \xrightarrow{\tau} T^{v_k} & \text{for all } k \in \{1,\dots,n\} \\
 & & \\
T^{aw} \xrightarrow{a} T^{w} & T^{aw} \xrightarrow{\tau} T^{w} & \text{for all } a \in \Sigma \text{ and } w \in \Sigma^* \text{ such that} \\
 & & aw \in \bigcup_{k=1}^{n} \mathcal{SF}(u_k) \cup \bigcup_{k=1}^{n} \mathcal{SF}(v_k) \\
T^{\epsilon} \xrightarrow{\tau} \epsilon. & &
\end{array}$$

This means that for a given $k \in \{1,\dots,n\}$ the process constants U^{u_k} and V^{v_k} can perform e.g. the following transitions (or transition sequences): $U^{u_k} \xrightarrow{\iota_k} \epsilon$, $V^{v_k} \xrightarrow{\iota_k} \epsilon$, $U^{u_k} \overset{u_k}{\Longrightarrow} \epsilon$, $V^{v_k} \overset{v_k}{\Longrightarrow} \epsilon$, $U^{u_k} \overset{\tau}{\Longrightarrow} \epsilon$, $V^{v_k} \overset{\tau}{\Longrightarrow} \epsilon$. The intuition is that a solution $i_1,\dots,i_m \in \{1,\dots,n\}$ of the (A,B)-instance is represented by a pair of processes $U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}}$ and $V^{v_{i_1}}.V^{v_{i_2}}.\cdots.V^{v_{i_m}}$. These processes can perform the sequences of visible actions $u_{i_1}u_{i_2}\dots u_{i_m}$ and $v_{i_1}v_{i_2}\dots v_{i_m}$, respectively, or they can perform the actions corresponding to the indices, namely $\iota_{i_1}\iota_{i_2}\dots\iota_{i_m}$. Moreover, since there is no global state control, the processes can produce also a combination of the actions from Σ and $\{\iota_1,\dots,\iota_n\}$. In order to avoid this undesirable behaviour, we add (via parallel composition) a process constant C_1 or C_2 such that C_1 masks all the actions from Σ and C_2 masks all the actions testing the indices. The reason for adding Z will become clear later.

The rewrite rules for C_1, C_2 and Z are given by:

$$\begin{array}{ll}
C_1 \xrightarrow{a} C_1 & \text{for all } a \in \Sigma \\
 & \\
C_2 \xrightarrow{\iota_k} C_2 & \text{for all } k \in \{1,\dots,n\} \\
 & \\
Z \xrightarrow{z} \epsilon & Z \xrightarrow{\tau} D.
\end{array}$$

Lemma 7.4. It holds that

$$Z.U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}} \parallel C_1 \;\approx\; Z.V^{v_{j_1}}.V^{v_{j_2}}.\cdots.V^{v_{j_{m'}}} \parallel C_1$$

if and only if

$$m = m' \text{ and } i_\ell = j_\ell \text{ for all } \ell,\ 1 \le \ell \le m = m'.$$

Proof. "$\Rightarrow$": Assume that (i) $m \neq m'$, or (ii) $m = m'$ and let ℓ, $1 \le \ell \le m = m'$, be the smallest number such that $i_\ell \neq j_\ell$. It is easy to show that

$Z.U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}} \parallel C_1 \not\approx Z.V^{v_{j_1}}.V^{v_{j_2}}.\cdots.V^{v_{j_{m'}}} \parallel C_1$. In case (i), assuming w.l.o.g. that $m > m'$, the attacker can perform in the first process a sequence of actions $z\iota_{i_1}\iota_{i_2}\ldots\iota_{i_m}$ of length $m+1$ and the defender cannot answer by any corresponding sequence of the same length from the second process ($m > m'$). Hence the attacker wins. In case (ii), the attacker again performs the sequence $z\iota_{i_1}\iota_{i_2}\ldots\iota_{i_m}$ in the first process. The only appropriate sequence of the same length that the defender can perform in the second process is $z\iota_{j_1}\iota_{j_2}\ldots\iota_{j_{m'}}$ — obviously no τ rules can be used otherwise the defender loses (his sequence gets shorter). The attacker wins because $\iota_{i_\ell} \neq \iota_{j_\ell}$.
"$\Leftarrow$": We show that $Z.U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}} \parallel C_1 \approx Z.V^{v_{i_1}}.V^{v_{i_2}}.\cdots.V^{v_{i_m}} \parallel C_1$. Let $U(\ell) \overset{\text{def}}{=} U^{u_{i_\ell}}.U^{u_{i_{\ell+1}}}.\cdots.U^{u_{i_m}}$ and $V(\ell) \overset{\text{def}}{=} V^{v_{i_\ell}}.V^{v_{i_{\ell+1}}}.\cdots.V^{v_{i_m}}$ for all ℓ, $1 \le \ell \le m$. By definition $U(m+1) \overset{\text{def}}{=} \epsilon$ and $V(m+1) \overset{\text{def}}{=} \epsilon$. Let us consider the following relation R.

$$\begin{array}{lrcll}
\{ (& Z.U(1)\|C_1 & , & Z.V(1)\|C_1 &) \} \cup \\
\{ (& D.U(1)\|C_1 & , & D.V(1)\|C_1 &) \} \cup \\
\{ (& U(\ell)\|C_1 & , & V(\ell)\|C_1 &) \mid 1 \le \ell \le m+1 \} \cup \\
\{ (& T^w.U(\ell)\|C_1 & , & V(\ell)\|C_1 &) \mid 2 \le \ell \le m+1 \ \wedge\ w \in \mathcal{SF}(u_{\ell-1})\} \cup \\
\{ (& U(\ell)\|C_1 & , & T^w.V(\ell)\|C_1 &) \mid 2 \le \ell \le m+1 \ \wedge\ w \in \mathcal{SF}(v_{\ell-1})\}
\end{array}$$

It is a routine exercise to check that R is a weak bisimulation. Moreover, it satisfies that $(Z.U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}} \parallel C_1,\ Z.V^{v_{i_1}}.V^{v_{i_2}}.\cdots.V^{v_{i_m}} \parallel C_1) \in R$. $\square$

Lemma 7.5. It holds that

$$Z.U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}} \parallel C_2 \approx Z.V^{v_{j_1}}.V^{v_{j_2}}.\cdots.V^{v_{j_{m'}}} \parallel C_2$$

if and only if

$$u_{i_1}u_{i_2}\ldots u_{i_m} = v_{j_1}v_{j_2}\ldots v_{j_{m'}}.$$

Proof. "$\Rightarrow$": Let $\sigma \overset{\text{def}}{=} U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}}$ and $\omega \overset{\text{def}}{=} V^{v_{j_1}}.V^{v_{j_2}}.\cdots.V^{v_{j_{m'}}}$, and let $u \overset{\text{def}}{=} u_{i_1}u_{i_2}\ldots u_{i_m}$ and $v \overset{\text{def}}{=} v_{j_1}v_{j_2}\ldots v_{j_{m'}}$. Hence $\sigma \overset{u}{\Longrightarrow} \epsilon$ and $\omega \overset{v}{\Longrightarrow} \epsilon$, and u and v are the longest sequences (and unique ones among the sequences of the length $|u|$ resp. $|v|$) of visible actions from Σ that σ and ω can perform. From the assumption that $u \neq v$ it is easy to see that $Z.\sigma\|C_2 \not\approx Z.\omega\|C_2$.
"$\Leftarrow$": We show that $Z.U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}} \parallel C_2 \approx Z.V^{v_{j_1}}.V^{v_{j_2}}.\cdots.V^{v_{j_{m'}}} \parallel C_2$ assuming that $u_{i_1}u_{i_2}\ldots u_{i_m} = v_{j_1}v_{j_2}\ldots v_{j_{m'}}$. Let $\alpha \in \mathcal{SF}(u_{i_1}u_{i_2}\ldots u_{i_m}) = \mathcal{SF}(v_{j_1}v_{j_2}\ldots v_{j_{m'}})$. We define two sets $U(\alpha)$ and $V(\alpha)$. The intuition is that $U(\alpha)$ contains all the states reachable from $U^{u_{i_1}}.\cdots.U^{u_{i_m}}$ such that α is the longest sequence of visible actions from Σ that these states can perform, and similarly for $V(\alpha)$.

Let us fix the following notation: $U^{u_{m+1}}.\cdots.U^{u_m} \overset{\text{def}}{=} \epsilon$, $V^{v_{m'+1}}.\cdots.V^{v_{m'}} \overset{\text{def}}{=} \epsilon$ (here 'ϵ' stands for the empty process), and $u_{m+1}\ldots u_m \overset{\text{def}}{=} \epsilon$, $v_{m'+1}\ldots v_{m'} \overset{\text{def}}{=} \epsilon$ (here 'ϵ' means the empty sequence of actions).

$$\begin{aligned}
U(\alpha) \overset{\text{def}}{=} \; & \{U^{u_{i_\ell}}.U^{u_{i_{\ell+1}}}.\cdots.U^{u_{i_m}} \mid 1 \le \ell \le m \;\wedge\; u_{i_\ell}u_{i_{\ell+1}}\dots u_{i_m} = \alpha\} \cup \\
& \{T^w.U^{u_{i_\ell}}.U^{u_{i_{\ell+1}}}.\cdots.U^{u_{i_m}} \mid 2 \le \ell \le m+1 \;\wedge\; w \in \mathcal{SF}(u_{i_{\ell-1}}) \;\wedge \\
& \qquad\qquad\qquad\qquad wu_{i_\ell}u_{i_{\ell+1}}\dots u_{i_m} = \alpha\}
\end{aligned}$$

$$\begin{aligned}
V(\alpha) \overset{\text{def}}{=} \; & \{V^{v_{j_\ell}}.V^{v_{j_{\ell+1}}}.\cdots.V^{v_{j_{m'}}} \mid 1 \le \ell \le m' \;\wedge\; v_{j_\ell}v_{j_{\ell+1}}\dots v_{j_{m'}} = \alpha\} \cup \\
& \{T^w.V^{v_{j_\ell}}.V^{v_{j_{\ell+1}}}.\cdots.V^{v_{j_{m'}}} \mid 2 \le \ell \le m'+1 \;\wedge\; w \in \mathcal{SF}(v_{j_{\ell-1}}) \;\wedge \\
& \qquad\qquad\qquad\qquad wv_{j_\ell}v_{j_{\ell+1}}\dots v_{j_{m'}} = \alpha\}.
\end{aligned}$$

We remind the reader of the fact that $1 \le |U(\alpha)|, |V(\alpha)| \le 3$ for all α. For example if $m \ge 2$ then $U(u_{i_m}) = \{U^{u_{i_m}},\; T^{\epsilon}.U^{u_{i_m}},\; T^{u_{i_m}}\}$. Moreover, if $E \in U(\alpha)$ and $F \in V(\alpha)$ then $E \overset{\alpha}{\Longrightarrow} \epsilon$ and $F \overset{\alpha}{\Longrightarrow} \epsilon$, and α is the longest sequence of actions from Σ satisfying this property. Let us consider the following relation R where $U(1) \overset{\text{def}}{=} U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}}$, $V(1) \overset{\text{def}}{=} V^{v_{j_1}}.V^{v_{j_2}}.\cdots.V^{v_{j_{m'}}}$, and $\beta \overset{\text{def}}{=} u_{i_1}u_{i_2}\dots u_{i_m} = v_{j_1}v_{j_2}\dots v_{j_{m'}}$.

$$\begin{array}{lrcll}
\{\,(& Z.U(1)\|C_2 & , & Z.V(1)\|C_2 &)\,\} \cup \\
\{\,(& D.U(1)\|C_2 & , & D.V(1)\|C_2 &)\,\} \cup \\
\{\,(& E\|C_2 & , & F\|C_2 &) \mid E \in U(\alpha) \wedge F \in V(\alpha) \wedge \alpha \in \mathcal{SF}(\beta)\} \cup \\
\{\,(& C_2 & , & C_2 &)\,\}
\end{array}$$

As in the previous lemma, it is easy to check that R is a weak bisimulation. Moreover $(Z.U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}} \parallel C_2,\; Z.V^{v_{j_1}}.V^{v_{j_2}}.\cdots.V^{v_{j_{m'}}} \parallel C_2) \in R$. $\square$

We continue with the definition of Δ by adding rules which enable the defender to generate a solution of the (A, B)-instance (if it exists).

$$\begin{array}{lll}
X \xrightarrow{x} Y & X' \xrightarrow{x} X'_1 & \\
X \xrightarrow{x} X'_1 & & \\
& X'_1 \xrightarrow{\tau} X'_1.V^{v_k} & \text{for all } k \in \{1,\dots,n\} \\
& X'_1 \xrightarrow{\tau} Y'.V^{v_k} & \text{for all } k \in \{1,\dots,n\} \\
& & \\
Y \xrightarrow{y} Y_1 & Y' \xrightarrow{y} Z & \\
& Y' \xrightarrow{y} Y_1.D & \\
Y_1 \xrightarrow{\tau} Y_1.U^{u_k} & & \text{for all } k \in \{1,\dots,n\} \\
Y_1 \xrightarrow{\tau} Z.U^{u_k} & & \text{for all } k \in \{1,\dots,n\}
\end{array}$$

See Figure 7.5 for fragments of transition systems generated by (X, Δ) and (X', Δ). The following lemma explains the purpose of the rules defined above.

Lemma 7.6. Consider a bisimulation game from (X, Δ) and (X', Δ). The defender has a strategy such that after two rounds the players reach a pair of states $Z.\sigma$ and $Z.\omega$ where $\sigma = U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}}$ and $\omega = V^{v_{j_1}}.V^{v_{j_2}}.\cdots.V^{v_{j_{m'}}}$ $(m, m' \ge 1)$, and where σ and ω were chosen by the defender; or the defender wins by reaching a pair of weakly bisimilar states.

Proof. In the first round of the bisimulation game played from (X, Δ) and (X', Δ) the attacker has only one possible move: $X \xrightarrow{x} Y$. If the attacker

$$\begin{tikzcd}
& X \arrow[ddd, "x"] \arrow[dr, "x"] & \\
& & X_1' \arrow[d, dotted, no head] \\
& & {} \\
& Y \arrow[d, "y"] & \\
& Y_1 \arrow[d, Rightarrow, "\tau"] & \\
& Z.\sigma \arrow[dl, "\tau"'] \arrow[dr, "z"] & \\
D.\sigma \ (\approx \epsilon) & & \sigma \arrow[d, dotted, no head] \\
& & {}
\end{tikzcd}
\qquad
\begin{tikzcd}
& X' \arrow[d, "x"] & \\
& X_1' \arrow[d, Rightarrow, "\tau"] & \\
& Y'.\omega \arrow[dd, "y"] \arrow[dr, "y"] & \\
& & Y_1.D.\omega \ (\approx Y_1) \arrow[d, dotted, no head] \\
& Z.\omega \arrow[dl, "\tau"'] \arrow[dr, "z"] & {} \\
D.\omega \ (\approx \epsilon) & & \omega \arrow[d, dotted, no head] \\
& & {}
\end{tikzcd}$$

Figure 7.5: Fragments of transition systems generated by (X, Δ) and (X', Δ)

plays any other move ($X \xrightarrow{x} X_1'$ or $X' \xrightarrow{x} X_1'$) then the defender can make the resulting processes syntactically equal and he wins. The defender's answer to the move $X \xrightarrow{x} Y$ is by $X' \overset{x}{\Longrightarrow} Y'.\omega$ for some $\omega = V^{v_{j_1}}.V^{v_{j_2}}.\cdots.V^{v_{j_{m'}}}$ such that $m' \geq 1$.

In the next round played from Y and $Y'.\omega$ the attacker is forced to continue by $Y'.\omega \xrightarrow{y} Z.\omega$. Similarly as in the first round: if the attacker chooses any other move, the defender can make the resulting processes weakly bisimilar (here we use the fact that $Y_1 \approx Y_1.D.\omega$). The defender can now choose some $\sigma = U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}}$ such that $m \geq 1$ and plays $Y \overset{y}{\Longrightarrow} Z.\sigma$. Hence the defender either won or he chose nonempty σ and ω and forced the attacker in two rounds to reach the pair $Z.\sigma$ and $Z.\omega$. □

We finish the definition of Δ by adding the rules:

$$\begin{array}{lll}
C \xrightarrow{c_1} C_1 & C \xrightarrow{c_2} C_2 & C \xrightarrow{z} C \| W \\
 & & \\
W \xrightarrow{\tau} W.U^{u_k} & W \xrightarrow{\tau} W.V^{v_k} & \text{for all } k \in \{1, \ldots, n\} \\
W \xrightarrow{\tau} \epsilon. & &
\end{array}$$

The intuition is that while playing a bisimulation game from $X \| C$ and $X' \| C$, the defender can generate a solution of the (A, B)-instance by forcing the attacker to reach the states $Z.\sigma \| C$ and $Z.\omega \| C$ (see Lemma 7.6) such that $\sigma = U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}}$ and $\omega = V^{v_{i_1}}.V^{v_{i_2}}.\cdots.V^{v_{i_m}}$ where $i_1, \ldots, i_m$ is a solution of the (A, B)-instance (if it exists). The attacker waits with using the rule $C \xrightarrow{c_1} C_1$ or $C \xrightarrow{c_2} C_2$ until the pair $Z.\sigma \| C$ and $Z.\omega \| C$ is reached and then he can check that the sequence $i_1, \ldots, i_m$ is indeed a solution: from $Z.\sigma \| C_1$ and $Z.\omega \| C_1$ he checks whether the defender generated the same indices in both σ and ω, and from $Z.\sigma \| C_2$ and $Z.\omega \| C_2$ he checks whether $u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$. The purpose of the rules for the process constant W is explained later.

Lemma 7.7. If $(X \| C, \Delta) \approx (X' \| C, \Delta)$ then the (A, B)-instance has a solution.

Proof. Assume that the (A, B)-instance has no solution, i.e., for every sequence of indices $i_1, \ldots, i_m \in \{1, \ldots, n\}$ where $m \geq 1$ it is the case that $u_{i_1}u_{i_2} \ldots u_{i_m} \neq v_{i_1}v_{i_2} \ldots v_{i_m}$. We show that the attacker has a winning strategy from the pair $X\|C$ and $X'\|C$. In the first round the attacker plays $X\|C \xrightarrow{x} Y\|C$. The defender can only answer by $X'\|C \xrightarrow{x} X'_1\|C$ followed by a finite number of τ actions, thus reaching a state $X'_1.\omega\|C$ or $Y'.\omega\|C$ for some ω. In the first case the attacker switches the processes and uses e.g. the rule $X'_1 \xrightarrow{\tau} Y'.V^{v_1}$. Since $Y\|C \overset{\tau}{\not\longrightarrow}$, the defender can only stay at the state $Y\|C$. In the second case the state is already of the form $Y'.\omega\|C$.

The game now continues from the pair of states $Y\|C$ and $Y'.\omega\|C$ for some ω. The attacker chooses the move $Y'.\omega\|C \xrightarrow{y} Z.\omega\|C$. The defender has to answer by $Y\|C \xrightarrow{y} Y_1\|C$ followed by a finite number of τ actions. This means that he can reach a state $Y_1.\sigma\|C$, or $Z.\sigma\|C$, or $D.\sigma\|C$ for some σ. The attacker wants to force the defender to reach the second possibility. We show later that if the defender reaches $D.\sigma\|C$ then he loses. Moreover, if the defender reaches $Y_1.\sigma\|C$ then the attacker can use e.g. the rule $Y_1 \xrightarrow{\tau} Z.U^{u_1}$ and the defender can only respond by staying in $Z.\omega\|C$, or by the move $Z.\omega\|C \xrightarrow{\tau} D.\omega\|C$. As we want the game to continue from $Z.\sigma\|C$ and $Z.\omega\|C$, it is enough to show that the attacker has a winning strategy from $D.\sigma\|C$ and $Z.\omega\|C$, and from $Z.\sigma\|C$ and $D.\omega\|C$. We show how the attacker wins from $D.\sigma\|C$ and $Z.\omega\|C$ (the situation from $Z.\sigma\|C$ and $D.\omega\|C$ is completely symmetric). The attacker plays in the second state: $Z.\omega\|C \xrightarrow{c_1} Z.\omega\|C_1$. The defender can only respond by $D.\sigma\|C \xrightarrow{c_1} D.\sigma\|C_1$. Now, $Z.\omega\|C_1 \xrightarrow{z} \omega\|C_1$ but $D.\sigma\|C_1 \overset{z}{\not\Longrightarrow}$. Hence the attacker wins.

To sum up, either the attacker wins or the game continues from the pair $Z.\sigma\|C$ and $Z.\omega\|C$ for some $\sigma = U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}}$ and $\omega = V^{v_{j_1}}.V^{v_{j_2}}.\cdots.V^{v_{j_{m'}}}$ where $m, m' \geq 1$. There are two cases.

- If $m = m'$ and $i_\ell = j_\ell$ for all ℓ, $1 \leq \ell \leq m = m'$, then using our assumption that the (A, B)-instance has no solution and by the fact that $m, m' \geq 1$ we get that $u_{i_1}u_{i_2} \ldots u_{i_m} \neq v_{i_1}v_{i_2} \ldots v_{i_m}$. The attacker plays $Z.\sigma\|C \xrightarrow{c_2} Z.\sigma\|C_2$ and the defender has to answer by $Z.\omega\|C \xrightarrow{c_2} Z.\omega\|C_2$ or $Z.\omega\|C \overset{c_2}{\Longrightarrow} D.\omega\|C_2$. From the pair $Z.\sigma\|C_2$ and $Z.\omega\|C_2$ the attacker has a winning strategy because of Lemma 7.5 and the attacker's strategy from the pair $Z.\sigma\|C_2$ and $D.\omega\|C_2$ is obvious: $Z.\sigma\|C_2 \xrightarrow{z} \sigma\|C_2$ but $D.\omega\|C_2 \overset{z}{\not\Longrightarrow}$.

- If it is not the case that $m = m'$ and $i_\ell = j_\ell$ for all ℓ, $1 \leq \ell \leq m = m'$, the attacker plays $Z.\sigma\|C \xrightarrow{c_1} Z.\sigma\|C_1$ and the defender must respond by $Z.\omega\|C \xrightarrow{c_1} Z.\omega\|C_1$ or $Z.\omega\|C \overset{c_1}{\Longrightarrow} D.\omega\|C_1$. By Lemma 7.4 the attacker has a winning strategy from $Z.\sigma\|C_1$ and $Z.\omega\|C_1$. The argument for the attacker's winning strategy from $Z.\sigma\|C_1$ and $D.\omega\|C_1$ is as in the previous case.

□

Lemma 7.8. If the (A, B)-instance has a solution then $(X\|C, \Delta) \approx (X'\|C, \Delta)$.

Proof. Let $i_1, \ldots, i_m \in \{1, \ldots n\}$ where $m \geq 1$ be a solution of the (A, B)-instance. We show that the defender has a winning strategy from the pair $X \| C$ and $X' \| C$.

As it was already proved in Lemma 7.6, in the bisimulation game played from X and X' the defender can force the attacker to reach the pair $Z.\sigma$ and $Z.\omega$, or the defender has a winning strategy. In particular, the defender can make sure that the players reach the pair $Z.\sigma$ and $Z.\omega$ where σ and ω correspond to the solution of the (A, B)-instance, i.e., $\sigma = U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}}$ and $\omega = V^{v_{i_1}}.V^{v_{i_2}}.\cdots.V^{v_{i_m}}$.

The situation in this lemma, however, requires that the players start playing from $X \| C$ and $X' \| C$. We have to extend the defender's strategy by defining his responses to the attacks from the process constant C, or more generally from any context γ reachable from C (see the last part of the definition of Δ). To any attacker's move $X \| \gamma \longrightarrow X \| \gamma'$ or $X' \| \gamma \longrightarrow X' \| \gamma'$ the defender answers simply by imitating the same move in the other process. The bisimulation game then continues from the pair $X \| \gamma'$ and $X' \| \gamma'$. Since any infinite game is a winning one for the defender, the attacker must eventually use some rules for X or X'. In this case the defender uses the strategy from Lemma 7.6. The attacker is forced to play $X \| \gamma \xrightarrow{x} Y \| \gamma$ and the defender answers by $X' \| \gamma \overset{x}{\Longrightarrow} Y'.\omega \| \gamma$ where $\omega = V^{v_{i_1}}.V^{v_{i_2}}.\cdots.V^{v_{i_m}}$. From the states $Y \| \gamma$ and $Y'.\omega \| \gamma$, again the defender imitates any attacks from the context γ. Thus the attacker must eventually play $Y'.\omega \| \gamma \xrightarrow{y} Z.\omega \| \gamma$ and the defender answers by $Y \| \gamma \overset{y}{\Longrightarrow} Z.\sigma \| \gamma$ where $\sigma = U^{u_{i_1}}.U^{u_{i_2}}.\cdots.U^{u_{i_m}}$.

By inspecting the rules for C we can see that the context γ always contains either the process constant (i) C, (ii) C_1, or (iii) C_2. Hence γ can be written as (i) $C \| \gamma'$, (ii) $C_1 \| \gamma'$, or (iii) $C_2 \| \gamma'$ for some context γ'. In case (ii) the bisimulation game continues from the pair $Z.\sigma \| C_1 \| \gamma'$ and $Z.\omega \| C_1 \| \gamma'$, and the defender has a winning strategy because of Lemma 7.4 and Proposition 7.3. In case (iii) the bisimulation game continues from the pair $Z.\sigma \| C_2 \| \gamma'$ and $Z.\omega \| C_2 \| \gamma'$, and the defender has a winning strategy because of Lemma 7.5 and Proposition 7.3. It remains to demonstrate that the defender has a winning strategy also in case (i). Hence assume that the game continues from $Z.\sigma \| C \| \gamma'$ and $Z.\omega \| C \| \gamma'$. By Proposition 7.3 it is enough to show that $Z.\sigma \| C \approx Z.\omega \| C$. We will analyze the attacker's moves from $Z.\sigma \| C$. The arguments for the moves from $Z.\omega \| C$ are completely symmetric. The attacker has the following moves available.

$$\begin{array}{ll}
\text{(i)} & Z.\sigma \| C \xrightarrow{c_1} Z.\sigma \| C_1 \\
\text{(ii)} & Z.\sigma \| C \xrightarrow{c_2} Z.\sigma \| C_2 \\
\text{(iii)} & Z.\sigma \| C \xrightarrow{z} Z.\sigma \| C \| W \\
\text{(iv)} & Z.\sigma \| C \xrightarrow{\tau} D.\sigma \| C \\
\text{(v)} & Z.\sigma \| C \xrightarrow{z} \sigma \| C
\end{array}$$

In case (i) the defender answers by $Z.\omega \| C \xrightarrow{c_1} Z.\omega \| C_1$ and wins because of Lemma 7.4. In case (ii) the defender answers by $Z.\omega \| C \xrightarrow{c_2} Z.\omega \| C_2$ and wins because of Lemma 7.5. In case (iii) the defender answers by $Z.\omega \| C \xrightarrow{z} Z.\omega \| C \| W$. By Proposition 7.3 this case is already covered by the discussion of the defender's strategy from $Z.\sigma \| C$ and $Z.\omega \| C$. In case (iv) the defender answers by

$Z.\omega\|C \xrightarrow{\tau} D.\omega\|C$ and he wins since $D.\sigma\|C \approx C \approx D.\omega\|C$. Case (v) is the only case where we need the rules for the process constant W. The defender answers by the following sequence:

$$Z.\omega\|C \xrightarrow{\tau} D.\omega\|C \xrightarrow{z} D.\omega\|C\|W \xRightarrow{\tau} D.\omega\|C\|\sigma.$$

This can be written in one step as $Z.\omega\|C \xRightarrow{z} D.\omega\|C\|\sigma$. Now the game continues from the pair $\sigma\|C$ and $D.\omega\|C\|\sigma$, however, $\sigma\|C \approx D.\omega\|C\|\sigma$. This implies that the defender has a winning strategy also in this case. $\square$

Theorem 7.3. Weak bisimilarity of PA-processes with deadlocks is undecidable.

Proof. Immediately from Lemmas 7.7 and 7.8. $\square$

In the rest of this section we show that the presence of the deadlock D in Δ is not an essential requirement. We build upon the technique of deadlock elimination described (for the case of BPA) in [169].

Lemma 7.9. There is a (polynomial time) reduction from weak bisimilarity of PA with deadlocks to weak bisimilarity of PA without deadlocks.

Proof. Let Δ be a PA system. By $\mathcal{D}(\Delta)$ we denote the set of all process constants which have no rewrite rule in Δ, i.e., $\mathcal{D}(\Delta) = \{X \in \mathcal{C}onst(\Delta) \mid X \not\longrightarrow\}$. Let us consider a PA system Δ' such that $\mathcal{C}onst(\Delta') \overset{\text{def}}{=} \mathcal{C}onst(\Delta) \setminus \mathcal{D}(\Delta) \cup \{D\}$ and $\mathcal{A}ct(\Delta') \overset{\text{def}}{=} \mathcal{A}ct(\Delta) \cup \{d\}$ where D is a new process constant and d is a new action. Let $\Delta' \overset{\text{def}}{=} \{X \xrightarrow{a} \overline{E} \mid (X \xrightarrow{a} E) \in \Delta\} \cup \{D \xrightarrow{d} D\}$ such that $\overline{\epsilon} \overset{\text{def}}{=} \epsilon$, $\overline{X} \overset{\text{def}}{=} X$ if $X \notin \mathcal{D}(\Delta)$, $\overline{X} \overset{\text{def}}{=} D$ if $X \in \mathcal{D}(\Delta)$, $\overline{E.F} \overset{\text{def}}{=} \overline{E}.\overline{F}$, and $\overline{E\|F} \overset{\text{def}}{=} \overline{E}\|\overline{F}$, where X is a process constant and E, F are process expressions. Obviously $\mathcal{D}(\Delta') = \emptyset$ and it is easy to verify that $(E, \Delta) \approx (F, \Delta)$ if and only if $(\overline{E}\|D, \Delta') \approx (\overline{F}\|D, \Delta')$ for any process expressions E and F. $\square$

Hence the undecidability result is valid also for PA-processes without deadlocks.

Corollary 7.3. Weak bisimilarity of PA-processes (without deadlocks) is undecidable.

7.4 Concluding Remarks

We proved that weak bisimilarity of pushdown processes and PA-processes is undecidable. These results confirm that decidability issues for weak bisimilarity are more complex than those for strong bisimilarity, even though not many examples of infinite-state systems which give similar conclusions have been found so far.

In case of PDA we saw that the undecidability result holds also for the normed subclass. Using arguments from [89] we can moreover conclude that the problem is beyond the arithmetical hierarchy of undecidable problems. An interesting corollary of this result is that strong bisimilarity of prefix-recognizable graphs is also highly undecidable (we described a polynomial time reduction from weak bisimilarity of PDA to strong bisimilarity of prefix-recognizable graphs).

The undecidability result of weak bisimilarity for PA-processes contrasts to the situation of strong bisimilaririty for normed PA, which is known to be decidable in 2-NEXPTIME [73]. The problems of strong bisimilarity for unnormed PA and of weak bisimilarity for normed PA remain still open. Another question to be considered is, whether the problem of weak bisimilarity for PA is highly undecidable. In particular, we do not know whether it lies inside the arithmetical hierarchy, or whether it is beyond the hierarchy.

Another interesting problems left open are decidability of strong/weak regularity for PDA and PA.

7.5 Bibliographical Remarks

The content of this chapter is based on two papers. Results from Section 7.2 appeared in the paper "Undecidability of Weak Bisimilarity for Pushdown Processes" [181], except for Corollary 7.2 which was not published yet (we thank Colin Stirling for pointing out this fact). Section 7.3 is based on the paper "Undecidability of Weak Bisimilarity for PA-Processes" [180].

Chapter 8

Conclusion: State-of-the-Art

This chapter provides a comprehensive summary of equivalence checking results for infinite-state processes from the PRS-hierarchy.

8.1 Motivation

The growing interest in verification of infinite-state systems during the last decade led to the situation where many new results and novel approaches were invented. The first attempt to map the fundamental techniques and results for the equivalence checking problems was done by Moller in his overview paper "Infinite Results" [137], followed by the paper "More Infinite Results" [37] by Burkart and Esparza focusing on the model checking problems.

A large survey of equivalence and model checking techniques "Verification on Infinite Structures" appeared in the handbook of process algebra [34] due to Burkart, Caucal, Moller and Steffen. Yet another overview paper "Equivalence-Checking with Infinite-State Systems: Techniques and Results" [110] by Kučera and Jančar contains some recent techniques for simulation and bisimulation checking.

Although these comprehensive survey papers provide a valuable overview of proof techniques, the state-of-the-art advances so rapidly that many papers contain outdated information even before they are published.

The main objective of this chapter is to conclude the thesis by offering an updated overview of known decidability and complexity results for equivalence checking in the classes of process rewrite systems.

The most recent version of this chapter is available from the web-page `http://www.brics.dk/∼srba/roadmap` and we will promptly incorporate any improvements in the presented results and display them on the mentioned web-page.

We hope that the overview we provide will stimulate further research on fundamental equivalence checking problems for infinite-state systems and it will eventually lead towards a definitive closing of all the gaps in the mosaic of infinite results.

8.2 Studied Problems

In the overview tables presented in the following section we consider all the classes from the PRS-hierarchy as introduced in Chapter 2.

We include decidability and complexity results of the following problems:

- Strong/Weak Bisimilarity ($\sim$/$\approx$).
- Strong/Weak Bisimilarity with Finite-State Systems ($\sim$ FS/$\approx$ FS).
- Strong/Weak Regularity ($\sim$ reg/$\approx$ reg).

We deal with both unnormed and normed systems and for each decision problem we provide the best complexity bounds achieved so far. Results presented in this thesis have a reference pointing to the published papers where the proofs were first described.

The following remark[1] will be relevant for many results in the overview tables and that is why we include it into this section.

Remark 1. In the hierarchy of process rewrite systems there is a very close and obvious relationship between strong regularity checking of normed processes and the boundedness problem: a normed process is strongly regular if and only if it has only finitely many (on the syntactical level) reachable states.

Example 8.1. This examle illustrates that the property mentioned in Remark 1 for strong bisimilarity is not valid for weak bisimilarity. Consider the following BPA system Δ with $\mathcal{C}onst(\Delta) \overset{\text{def}}{=} \{X, Y\}$ and $\mathcal{A}ct(\Delta) \overset{\text{def}}{=} \{a\}$.

$$
\begin{array}{ll}
X \xrightarrow{a} X.X \qquad & X \xrightarrow{\tau} \epsilon \\
Y \xrightarrow{a} Y \qquad & Y \xrightarrow{\tau} \epsilon
\end{array}
$$

Obviously, the process (X, Δ) in normed and it has infinitely many reachable states of the form X^i for $i \in \mathbb{N}_0$. On the other hand, it is still weakly regular since $(X, \Delta) \approx (Y, \Delta)$. □

8.3 Summary of Known Results

Each box in the tables below contains the information whether the considered problem is decidable or not, and in the positive case we present the best known upper bound in the upper part of the box and lower bound in the lower part.

[1] In order to enable quick orientation throughout this chapter, we will drop the chapter number when refering to remarks. We also use e.g. R.1 instead of the full reference Remark 1.

8.3.1 BPA (Basic Process Algebra)

	BPA	normed BPA
$\sim$	$\in$ 2-EXPTIME [35] PSPACE-hard [178]	$\in$ P [74] P-hard [17]
$\approx$	? EXPTIME-hard [128]	? EXPTIME-hard [128]
$\sim$ FS	$\in$ P [113] P-hard [17]	$\in$ P [74] P-hard [17]
$\approx$ FS	$\in$ P [113] P-hard [17]	$\in$ P [113] P-hard [17]
$\sim$ reg	$\in$ 2-EXPTIME [36, 35] PSPACE-hard [178]	$\in$ NL [106] NL-hard [178]
$\approx$ reg	? EXPTIME-hard [128]	? NP-hard [174, 189]

8.3.2 BPP (Basic Parallel Processes)

	BPP	normed BPP
$\sim$	decidable [43], R.2 PSPACE-hard [177]	$\in$ P [75] P-hard [17]
$\approx$	?, R.2 PSPACE-hard [174]	?, R.2 PSPACE-hard [174]
$\sim$ FS	$\in$ PSPACE [94] P-hard [17]	$\in$ P [75] P-hard [17]
$\approx$ FS	$\in$ PSPACE [94] P-hard [17]	$\in$ P [113] P-hard [17]
$\sim$ reg	decidable [93] PSPACE-hard [177]	$\in$ NL [106] NL-hard [177]
$\approx$ reg	? PSPACE-hard [174]	? PSPACE-hard [174]

Remark 2. Jančar announced [92] that strong bisimilarity of BPP is in PSPACE and he also conjectured that the method might be used to show decidability of weak bisimilarity for BPP.

8.3.3 PDA (Pushdown Processes)

	PDA	normed PDA
$\sim$	decidable [163], R.3 EXPTIME-hard [112]	decidable [184] EXPTIME-hard [112]
$\approx$	undecidable [181]	undecidable [181]
$\sim$ FS	$\in$ PSPACE [112] PSPACE-hard [125]	$\in$ PSPACE [112] PSPACE-hard [125], R.4
$\approx$ FS	$\in$ PSPACE [112] PSPACE-hard [125]	$\in$ PSPACE [112] PSPACE-hard [125], R.5
$\sim$ reg	? EXPTIME-hard [112, 178],R.6	$\in$ P [60], R.7 NL-hard [178]
$\approx$ reg	? EXPTIME-hard [112, 178],R.6	? EXPTIME-hard [112, 178],R.5,6

Remark 3. Additional useful references concerning deterministic PDA are [164], [165] and [186].

Remark 4. The reduction from [125] (Theorem 8) uses unnormed processes but can be modified to work also for the normed case. An important observation is that the stack size of the PDA from Theorem 8 is bounded by the number of variables in the instance of quantified satisfiability from which the reduction is done.

Remark 5. Lemma 3 in [181] gives a polynomial time reduction from weak bisimilarity between two pushdown processes (and between a pushdown process and a finite-state process) to the normed instance of the problem. The reduction moreover preserves the property of being weakly regular. See also Remark 7.2 in Chapter 7.

Remark 6. In [112] a polynomial time reduction from the acceptance problem of alternating linear-bounded automata to strong bisimilarity of normed PDA is provided. Even though there are infinitely many reachable configurations in the constructed PDAs, one can observe that only a fixed part from the top of the stack is relevant for the construction. Hence it is possible to ensure that the PDAs are strongly regular and Theorem 2 from [178] can be applied.

Remark 7. Strong regularity of normed PDA is equivalent to the boundedness problem (Remark 1). Boundedness (even for unnormed PDA) is decidable in polynomial time using the fact that the set of all reachable configurations of a pushdown process is a regular language L [31] and a finite automaton A recognizing L can be constructed in polynomial time (see e.g. [60]). The check whether A generates a finite language can also be done in polynomial time.

8.3.4 PA (Process Algebra)

	PA	normed PA
$\sim$	? PSPACE-hard [177]	2-NEXPTIME [73] P-hard [17]
$\approx$	undecidable [180]	? EXPTIME-hard [128]
$\sim$ FS	decidable [86] P-hard [17]	2-NEXPTIME [73] P-hard [17]
$\approx$ FS	decidable [94] P-hard [17]	decidable [94] P-hard [17]
$\sim$ reg	? PSPACE-hard [177]	$\in$ NL [107] NL-hard [177]
$\approx$ reg	? EXPTIME-hard [128]	? PSPACE-hard [174]

8.3.5 PN (Petri Nets)

	PN	normed PN
$\sim$	undecidable [90]	undecidable [90], R.8
$\approx$	undecidable [90]	undecidable [90], R.8
$\sim$ FS	decidable [96] EXPSPACE-hard [116], R.9	decidable [96] P-hard [17]
$\approx$ FS	undecidable [93]	? EXPSPACE-hard [116], R.9
$\sim$ reg	decidable [93] EXPSPACE-hard [116], R.10	$\in$ EXPSPACE [157], R.1 EXPSPACE-hard [116], R.10
$\approx$ reg	undecidable [93]	? EXPSPACE-hard [116], R.10

Remark 8. The technique for proving undecidability of strong bisimilarity for Petri nets from [90] can be slightly modified to ensure that the constructed nets are normed. Essentially, it is enough to add extra transitions which enable to remove all tokens from places. Moreover, whenever such an extra transition is fired the two nets are forced to become bisimilar.

Remark 9. The problem whether a given place p of a PN can ever become marked is EXPSPACE-hard (follows from Lipton's construction [116], for a more accessible proof see e.g. [58]). We can now easily see that this problem is reducible in polynomial time to strong nonbisimilarity between PN and FS.

All transitions in a given Petri net N are assigned the same label a and we add one more place q (initially marked) and an extra transition labelled by a which takes a token from the place q and returns it back. Moreover we add another transition labelled by b which can be fired whenever there is a token in the place p. Let P be a finite-state process defined by $P \xrightarrow{a} P$. The following property is immediate: the place p can become marked iff N is not strongly bisimilar to P.

This reduction works also for normed PN and weak bisimilarity: we take our modified net N and for each its place we add one extra transition labelled by τ such that the transition takes a token from the place and removes it. To the finite-state process P we add the rewrite rule $P \xrightarrow{\tau} \epsilon$. The net N is now normed and moreover it is weakly bisimilar to P iff the place p can never become marked.

Remark 10. Regularity of normed PN is equivalent to the boundedness problem (Remark 1). Boundedness of PN is decidable in EXPSPACE, more precisely in space $2^{cn \log n}$ for some constant c [157]. Moreover, the boundedness problem of normed PN is EXPSPACE-hard because it can be easily seen to be polynomially equivalent to the boundedness problem of general (unnormed) PN and this problem is EXPSPACE-hard [116] (see also [58]).

8.3.6 PAD

	PAD	normed PAD
$\sim$	? EXPTIME-hard [112]	? EXPTIME-hard [112]
$\approx$	undecidable [181]	undecidable [181]
$\sim$ FS	decidable [86] PSPACE-hard [125]	decidable [86] PSPACE-hard [125], R.4
$\approx$ FS	decidable [94] PSPACE-hard [125]	decidable [94] PSPACE-hard [125], R.5
$\sim$ reg	? EXPTIME-hard [112, 178],R.6	decidable [129], R.1 NL-hard [178]
$\approx$ reg	? EXPTIME-hard [112, 178],R.6	? EXPTIME-hard [112, 178],R.5,6

8.3.7 PAN

	PAN	normed PAN
$\sim$	undecidable [90]	undecidable [90], R.8
$\approx$	undecidable [90]	undecidable [90], R.8
$\sim$ FS	? EXPSPACE-hard [116], R.9	decidable [129], R.1 P-hard [17]
$\approx$ FS	undecidable [93]	? EXPSPACE-hard [116], R.9
$\sim$ reg	? EXPSPACE-hard [116], R.10	decidable [129], R.1 EXPSPACE-hard [116], R.10
$\approx$ reg	undecidable [93]	? EXPSPACE-hard [116], R.10

8.3.8 PRS (Process Rewrite Systems)

	PRS	normed PRS
$\sim$	undecidable [90]	undecidable [90], R.8
$\approx$	undecidable [90]	undecidable [90], R.8
$\sim$ FS	? EXPSPACE-hard [116], R.9	decidable [129], R.1 PSPACE-hard [125], R.4
$\approx$ FS	undecidable [93]	? EXPSPACE-hard [116], R.9
$\sim$ reg	? EXPSPACE-hard [116], R.10	decidable [129], R.1 EXPSPACE-hard [116], R.10
$\approx$ reg	undecidable [93]	? EXPSPACE-hard [116], R.10

8.4 Bibliographical Remarks

The content of this chapter is based on the overview paper “Roadmap of Infinite Results” [176]. We also designed a project which aims to update the information about the mentioned results according to the current development in equivalence checking of infinite-state systems. The updated version of the document is available from `http://www.brics.dk/~srba/roadmap`.

Possible inclusion into a mailing list enables to inform interested researchers about any new results within the area.

Bibliography

[1] P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[2] P. A. Abdulla and K. Cerans. Simulation is decidable for one-counter nets. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *LNCS*, pages 253–268, 1998.

[3] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53(2-3):225–241, 1987.

[4] L. Aceto and M. Hennessy. Termination, deadlock, and divergence. *Journal of the ACM*, 39(1):147–187, 1992.

[5] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. *Proceedings of the 5th Symposium on Logic in Computer Science (LICS 90)*, pages 414–425, 1990.

[6] R. Alur, C. Courcoubetis, T. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R. Grossman, A.Nerode, A. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer-Verlag, 1993.

[7] R. Alur and D. Dill. Automata for modelling real-time systems. In *Proceedings of the 17th International Colloquium on Algorithms, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335, 1990.

[8] R. Alur and D. Dill. Automata for Modelling Real-Time Systems. *Theoretical Computer Science*, 126(2):183–236, 1994.

[9] J. Baeten and J. Bergstra. Mode transfer in process algebra. Technical report CSR 00-01, Vakgroep Informatica, Technische Universiteit Eindhoven, 2000.

[10] J. Baeten, J. Bergstra, and J. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *Proceedings of the Conference on Parallel Architectures and Languages Europe (PARLE'87). Volume II: Parallel Languages*, volume 259 of *LNCS*, pages 94–113. Springer-Verlag, 1987.

[11] J. Baeten, J. Bergstra, and J. Klop. Ready-trace semantics for concrete process algebra with the priority pperator. *Computer Journal*, 30(6):498–506, 1987.

[12] J. Baeten, J. Bergstra, and J. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM*, 40(3):653–682, 1993.

[13] J. Baeten, J. Bergstra, and J. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the Association for Computing Machinery*, 40:653–682, 1993.

[14] J. Baeten and R. van Glabbeek. Another look at abstraction in process algebra. In *Proceedings of the 14th International Colloquium on Automata, Languages and Programming (ICALP'87)*, volume 267 of *LNCS*, pages 84–94. Springer-Verlag, 1987.

[15] J. Baeten and W. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[16] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, IX(2):127–168, 1986.

[17] J. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4(6A):638–648, 1992.

[18] M. A. Bednarczyk. *Categories of Asynchronous Systems*. PhD thesis, University of Sussex, 1988.

[19] M. A. Bednarczyk. Hereditary history preserving bisimulations or what is the power of the future perfect in program logics. Technical report — http://www.ipipan.gda.pl/~marek, Polish Academy of Sc., Gdansk, 1991.

[20] B. Berard, A. Labroue, and P. Schnoebelen. Verifying performance equivalence for timed basic parallel processes. In *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'00)*, volume 1784 of *LNCS*, pages 35–47. Springer-Verlag, 2000.

[21] J. Bergstra and J. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[22] J. A. Bergstra. A mode transfer operator in process algebra. Technical report P8808b, University of Amsterdam, The Netherlands, 1989.

[23] B. Bloom, S. Istrail, and A. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.

[24] G. V. Bochmann. Finite state description of communication protocols. *Computer Networks and ISDN Systems*, 2:361–372, 1985.

[25] T. Bolognesi, F. Lucidi, and S. Trigila. From timed Petri nets to timed LOTOS. In *Proceedings of the IFIP WG 6.1 Tenth International Symposium on Protocol Specification, Testing and Verification (Ottawa 1990)*, pages 1–14. North-Holland, Amsterdam, 1990.

[26] 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 *LNCS*, pages 135–150. Springer-Verlag, 1997.

[27] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of *LNCS*, pages 323–333. Springer-Verlag, 1999.

[28] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[29] S. Brookes, C. Hoare, and A. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[30] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[31] J. Büchi. Regular canonical systems. *Arch. Math. Logik u. Grundlagenforschung*, 6:91–111, 1964.

[32] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[33] O. Burkart. Queues as processes. In *Proceedings of MFCS'98 Workshop on Concurrency — Algorithms and Tools*, volume 18 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2000.

[34] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.

[35] O. Burkart, D. Caucal, and B. Steffen. An elementary 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 *LNCS*, pages 423–433. Springer-Verlag, 1995.

[36] 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 *LNCS*, pages 247–262. Springer-Verlag, 1996.

[37] O. Burkart and J. Esparza. More infinite results. *Bulletin of the European Association for Theoretical Computer Science*, 62:138–159, June 1997. Columns: Concurrency.

[38] D. Caucal. Graphes canoniques de graphes algebriques. Rapport de Recherche 872, INRIA, 1988.

[39] D. Caucal. Graphes canoniques de graphes algébriques. *Theoretical Informatics and Applications*, 24(4):339–352, 1990.

[40] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, 1992.

[41] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Proceedings of the 23th International Colloquium on Automata, Languages and Programming (ICALP'96)*, volume 1099, pages 194–205. Springer-Verlag, 1996.

[42] S. Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, The University of Edinburgh, 1993.

[43] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation is decidable for basic parallel processes. In *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *LNCS*, pages 143–157. Springer-Verlag, 1993.

[44] S. Christensen, Y. Hirshfeld, and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science (LICS'93)*, pages 386–396. IEEE Computer Society Press, 1993.

[45] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In *Proceedings of the 3rd International Conference on Concurrency Theory (CONCUR'92)*, volume 630 of *LNCS*, pages 138–147. Springer-Verlag, 1992.

[46] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121:143–148, 1995.

[47] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag, 1981.

[48] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[49] T. Cobben and A. Engels. Disrupt and interrupt in MSC: Possibilities and problems. In *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC*, number 104 in Informatik-Berichte, pages 75–83. Humboldt-Universität zu Berlin, 1998.

[50] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[51] P. Darondeau. An enlarged definition and complete axiomatisation of observational congruence of finite processes. In *Proceedings of the 5th International Symposium on Programming*, volume 137 of *LNCS*, pages 47–62. Springer-Verlag, 1982.

[52] R. de Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24(2):211–237, 1987.

[53] R. de Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(1-2):83–133, 1984.

[54] L. Dickson. Finiteness of the odd perfect and primitive abundant numbers with distinct factors. *American Journal of Mathematics*, 35:413–422, 1913.

[55] B. Diertens. New features in PSF I: Interrupts, disrupts, and priorities. Technical report P9417, University of Amsterdam, The Netherlands, 1994.

[56] F. Drewes, A. Habel, and H. Kreowski. Hyperedge replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapter 2, pages 95–162. World Scientific, 1997.

[57] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. In *Proceedings of the 10th International Conference on Fundamentals of Computation Theory (FCT'95)*, volume 965 of *LNCS*, pages 221–232. Springer-Verlag, 1995.

[58] J. Esparza. Decidability and complexity of Petri net problems – an introduction. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 374–428. Springer-Verlag, 1998.

[59] J. Esparza. Grammars as processes. In W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, editors, *Formal and Natural Computing*, volume 2300 of *LNCS*, pages 277–297. Springer-Verlag, 2002.

[60] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 232–247. Springer-Verlag, 2000.

[61] J. Esparza and J.Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proceedings of the 2nd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'99)*, volume 1578 of *LNCS*, pages 14–30. Springer-Verlag, 1999.

[62] P. Godefroid. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032 of *LNCS*. Springer-Verlag, 1996.

[63] J. Groote. A short proof of the decidability of bisimulation for normed BPA processes. *Information Processing Letters*, 42(3):167–171, 1992.

[64] J. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):353–371, 1994.

[65] J. Groote and F. Vaandrager. Structural operational semantics and bisimulation as a congruence (extended abstract). In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP'89)*, volume 372 of *LNCS*, pages 423–438. Springer-Verlag, 1989.

[66] A. Habel. *Hyperedge Replacement: Grammars and Languages.* PhD thesis, University of Bremen, 1992.

[67] H. Hanisch. Analysis of place/transition nets with timed-arcs and its application to batch process control. In *Proceedings of the 14th International Conference on Application and Theory of Petri Nets (ICATPN'93)*, volume 691 of *LNCS*, pages 282–299, 1993.

[68] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the Association for Computing Machinery*, 32(1):137–161, 1985.

[69] M. Hennessy and G. Plotkin. A term model for CCS. In *Proceedings of the 9th International Symposium on Mathematical Foundations of Computer Science (MFCS'80)*, volume 88 of *LNCS*, pages 261–274. Springer-Verlag, 1980.

[70] Y. Hirshfeld. Petri nets and the equivalence problem. In *Proceedings of the 7th Workshop on Computer Science Logic (CSL'93)*, volume 832 of *LNCS*, pages 165–174. Springer-Verlag, 1993.

[71] Y. Hirshfeld. Congruences in commutative semigroups. Technical report ECS-LFCS-94-291, Department of Computer Science, University of Edinburg, 1994.

[72] Y. Hirshfeld. Bisimulation trees and the decidability of weak bisimulations. In *Proceedings of the 1st International Workshop on Verification of Infinite State Systems (INFINITY'96)*, volume 5 of *Electronic Notes in Theoretical Computer Science*. Springer-Verlag, 1996.

[73] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proceedings of 26th International Colloquium on Automata, Languages and Programming (ICALP'99)*, volume 1644 of *LNCS*, pages 412–421. Springer-Verlag, 1999.

[74] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158(1–2):143–159, 1996.

[75] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed basic parallel processes. *Mathematical Structures in Computer Science*, 6(3):251–259, 1996.

[76] Y. Hirshfeld and F. Moller. A fast algorithm for deciding bisimilarity of normed context-free processes. In *Proceedings of the 5th International Conference on Concurrency Theory (CONCUR'94)*, volume 836 of *LNCS*, pages 48–63. Springer-Verlag, 1994.

[77] Y. Hirshfeld and F. Moller. Decidability results in automata and process theory. In *Logics for Concurrency: Automata vs. Structure*, volume 1043 of *LNCS*, pages 102–148. F. Moller and G. Birtwistle, 1996.

[78] C. Hoare. Communicating sequential processes. In *On the construction of programs – an advanced course*, pages 229–254. Cambridge University Press, 1980.

[79] C. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[80] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[81] H. Hüttel. *Decidability, Behavioural Equivalences and Infinite Transition Graphs.* PhD thesis, The University of Edinburgh, 1991.

[82] H. Hüttel. Undecidable equivalences for basic parallel processes. In *Proceedings of the 2nd International Symposium on Theoretical Aspects of Computer Software (TACS'94)*, volume 789 of *LNCS*, pages 454–464. Springer-Verlag, 1994.

[83] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science (LICS'91)*, pages 376–386. IEEE Computer Society Press, 1991.

[84] D. Huynh and L. Tian. Deciding bisimilarity of normed context-free processes is in Σ_2^p. *Theoretical Computer Science*, 123(2):183–197, 1994.

[85] D. Huynh and L. Tian. On deciding readiness and failure equivalences for processes in Σ_2^P. *Information and Computation*, 117(2):193–205, 1995.

[86] P. Jančar and A. Kučera. Bisimilarity of processes with finite-state systems. In *Proceedings of the 2nd International Workshop on Verification of Infinite State Systems (INFINITY'97)*, volume 9 of *Electronic Notes in Theoretical Computer Science*, 1997.

[87] P. Jančar. Decidability of a temporal logic problem for Petri nets. *Theoretical Computer Science*, 74(1):71–93, 1990.

[88] P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. In *Proceedings of the 11th Symposium on Theoretical Aspects of Computer Science (STACS'94)*, volume 775 of *LNCS*, pages 581–592. Springer-Verlag, 1994.

[89] P. Jančar. High undecidability of weak bisimilarity for Petri nets. In *Proceedings of Colloquium on Trees in Algebra and Programming (CAAP'95)*, volume 915 of *LNCS*, pages 349–363. Springer-Verlag, 1995.

[90] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995.

[91] P. Jančar. Bisimulation equivalence is decidable for one-counter processes. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *LNCS*, pages 549–559. Springer-Verlag, 1997.

[92] P. Jančar. New results for bisimilarity on basic parallel processes. In *Proceedings of the 4th International Workshop on Verification of Infinite-State Systems (INFINITY'02)*, page 153. Technical report, Faculty of Informatics, Masaryk University Brno, FIMU-RS-2002-04, 2002. Short presentation.

[93] P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimulation. In *Proceedings of 23rd International Colloquium on Automata, Languages, and Programming (ICALP'96)*, volume 1099 of *LNCS*, pages 478–489. Springer-Verlag, 1996.

[94] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theoretical Computer Science*, 258(1–2):409–433, 2001.

[95] P. Jančar, A. Kučera, and F. Moller. Simulation and bisimulation over one-counter processes. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science (STACS'00)*, volume 1770 of *LNCS*, pages 334–345. Springer-Verlag, 2000.

[96] P. Jančar and F. Moller. Checking regular properties of Petri nets. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *LNCS*, pages 348–362. Springer-Verlag, 1995.

[97] P. Jančar and F. Moller. Techniques for decidability and undecidability of bisimilarity – an invited tutorial. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 30–45. Springer-Verlag, 1999.

[98] L. Jategaonkar and A. R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154(1):107–143, 1996.

[99] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

[100] M. Jurdzinski and M. Nielsen. Hereditary history preserving bisimilarity is undecidable. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS'00)*, volume 1770 of *LNCS*. Springer-Verlag, 2000.

[101] M. Jurdzinski, M. Nielsen, and J. Srba. Undecidability of domino games and hhp-bisimilarity. *Information and Computation*, 2002. To appear.

[102] P. Kanellakis and S. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[103] J. Kennaway. *Formal Semantics of Nondeterminism and Parallelism.* PhD thesis, University of Oxford, 1981.

[104] J. Knoop. *Optimal Interprocedural Program Optimization: A New Framework and its Application*, volume 1428 of *LNCS*. Springer-Verlag, 1998.

[105] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[106] A. Kučera. Regularity is decidable for normed BPA and normed BPP processes in polynomial time. In *Proceedings of the 23th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'96)*, volume 1175 of *LNCS*, pages 377–384. Springer-Verlag, 1996.

[107] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Proceedings of the 16th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'96)*, volume 1180 of *LNCS*, pages 111–122. Springer-Verlag, 1996.

[108] A. Kučera. Efficient verification algorithms for one-counter processes. In *Proceedings of the 27th International Colloquium on Automata, Languages, and Programming (ICALP'00)*, volume 1853 of *LNCS*, pages 317–328. Springer-Verlag, 2000.

[109] A. Kučera and J. Esparza. A logical viewpoint on process-algebraic quotients. In *Proceedings of the 8th Annual Conference of the European Association for Computer Science Logic (CSL'99)*, volume 1683 of *LNCS*, pages 499–514. Springer-Verlag, 1999.

[110] A. Kučera and P. Jančar. Equivalence-checking with infinite-state systems: Techniques and results. In *Proceedings of the 29th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'02)*, volume 2540 of *LNCS*, pages 41–73. Springer-Verlag, 2002.

[111] A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*. Springer-Verlag, 1999.

[112] 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 *LNCS*, pages 433–445. Springer-Verlag, 2002.

[113] A. Kučera and R. Mayr. Weak bisimilarity between finite-state systems and BPA or normed BPP is decidable in polynomial time. *Theoretical Computer Science*, 270:667–700, 2002.

[114] K. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Proceedings of the 10th International Conference on Fundamentals of Computation Theory (FCT'95)*, number 965 in LNCS, pages 62–88, 1995.

[115] S. Lasota. Decidability of strong bisimilarity for timed bpp. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 562–578. Springer-Verlag, 2002.

[116] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, 1976.

[117] M. Lohrey, P. D'Argenio, and H. Hermanns. Axiomatising divergence. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *LNCS*, pages 585–596, 2002.

[118] M. Lowry. Software construction and analysis tools for future space missions. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280, pages 1–19. Springer-Verlag, 2002.

[119] K. Marriott. Frameworks for abstract interpretation. *Acta Informatica*, 30(2):103–129, 1993.

[120] R. Mayr. Weak bisimulation and model checking for basic parallel processes. In *Proceedings of the 16th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'96)*, volume 1180 of *LNCS*, pages 88–99. Springer-Verlag, 1996.

[121] R. Mayr. Combining Petri nets and PA-processes. In *Proceedings of the 3rd International Symposium on Theoretical Aspects of Computer Software (TACS'97)*, volume 1281 of *LNCS*, pages 547–561. Springer-Verlag, 1997.

[122] R. Mayr. Process rewrite systems. In *Proceedings of the 4th International Workshop on Expressiveness in Concurrency (EXPRESS'97)*, volume 7 of *Electronic Notes in Theoretical Computer Science*, 1997.

[123] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems.* PhD thesis, TU-München, 1998.

[124] R. Mayr. On the complexity of bisimulation problems for basic parallel processes. In *Proceedings of 27st International Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *LNCS*, pages 329–341. Springer-Verlag, 2000.

[125] R. Mayr. On the complexity of bisimulation problems for pushdown automata. In *Proceedings of IFIP International Conference on Theoretical Computer Science (IFIP TCS'00)*, volume 1872 of *LNCS*. Springer-Verlag, 2000.

[126] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.

[127] R. Mayr. Undecidable problems in unreliable computations. In *Proceedings of Latin American Theoretical Informatics (LATIN'00)*, volume 1776 of *LNCS*. Springer-Verlag, 2000.

[128] R. Mayr. Weak bisimilarity and regularity of BPA is EXPTIME-hard. Technical Report No. 181, Department of Computer Science, Freiburg University, Germany, 2002.

[129] R. Mayr and M. Rusinowitch. Reachability is decidable for ground AC rewrite systems. In *Proceedings of 3rd International Workshop on Verification of Infinite State Systems (INFINITY'98)*, volume TUM-I9825 of *Technical Report, Technishe Universität München*, pages 53–64, 1998.

[130] A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

[131] A. Mazurkiewicz. Trace theory. In *Petri Nets, Applications and Relationship to other Models of Concurrency*, number 255 in LNCS, pages 279–324. Springer-Verlag, 1987.

[132] K. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[133] R. Milner. A calculus of communicating systems. *LNCS*, 92, 1980.

[134] R. Milner. A modal characterization of observable machine-behaviour. In *Proceedings of the 6th Colloquium on Trees in Algebra and Programming (CAAP'81)*, volume 112 of *LNCS*, pages 25–34. Springer-Verlag, 1981.

[135] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[136] M. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.

[137] F. Moller. Infinite results. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *LNCS*, pages 195–216. Springer-Verlag, 1996.

[138] F. Moller and S. Smolka. On the computational complexity of bisimulation. *ACM Computing Surveys*, 27(2):287–289, 1995.

[139] S. Muchnick. *Advanced Compiler Design and Implementation.* Morgan Kaufmann Publishers, 1997.

[140] M. Nielsen, V. Sassone, and J. Srba. Properties of distributed timed-arc Petri nets. In *Proceedings of the 21st International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01)*, volume 2245 of *LNCS*, pages 280–291. Springer-Verlag, 2001.

[141] M. Nielsen, V. Sassone, and J. Srba. Towards a notion of distributed time for Petri nets. In *Proceedings of the 22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *LNCS*, pages 23–31. Springer-Verlag, 2001.

[142] M. Nielsen and G. Winskel. Petri nets and bisimulation. *Theoretical Computer Science*, 153(1–2):211–244, 1996.

[143] E. Olderog and C. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23(1):9–66, 1986.

[144] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[145] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, New York, 1994.

[146] D. Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI Conference*, volume 104 of *LNCS*, pages 167–183. Springer-Verlag, 1981.

[147] D. Peled. All from one, one for all: On model checking using representatives. In *Proceedings of the 5th International Computer Aided Verification Conference (CAV'93)*, volume 697 of *LNCS*, pages 409–423, 1993.

[148] J. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice-Hall, 1981.

[149] C. Petri. *Kommunikation mit Automaten.* PhD thesis, Darmstadt, 1962.

[150] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50(3):241–284, 1987.

[151] G. Plotkin. A structural approach to operational semantics. Technical Report Daimi FN-19, Department of Computer Science, University of Aarhus, 1981.

[152] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *Proceedings of the 12th International Colloquium on Automata, Languages and Programming (ICALP'85)*, volume 194 of *LNCS*, pages 15–32. Springer-Verlag, 1985.

[153] L. Pomello. Some equivalence notions for concurrent systems. an overview. In *Advances in Petri Nets 1985*, volume 222 of *LNCS*, pages 381–400. Springer-Verlag, 1986.

[154] E. Post. A variant of a recursively unsolvable problem. *Bulletion of the American Mathematical Society*, 52:264–268, 1946.

[155] V. Pratt. A decidable mu-calculus. In *Proceedings of the 22nd IEEE Symposium on Foundations of Computer Science (FOCS'81)*, pages 421–427. IEEE Computer Society Press, 1981.

[156] A. Rabinovich and B. Trakhtenbrot. Behaviour structures and nets of processes. *Fundamenta Informaticae*, 11:357–404, 1988.

[157] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

[158] W. Rounds and S. Brookes. Possible futures, acceptances, refusals, and communicating processes. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science (FOCS'81)*, pages 140–149. IEEE Computer Society Press, 1981.

[159] V. V. Ruiz, D. de Frutos Escrig, and O. M. Alonso. Decidability of properties of timed-arc Petri nets. In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets (ICATPN'00)*, volume 1825 of *LNCS*, pages 187–206. Springer-Verlag, 2000.

[160] J. Rutten. Universal coalgebra: A theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[161] V. Saraswat. *Concurrent Constraint Programming.* MIT Press, Cambridge, MA, 1993.

[162] G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *LNCS*, pages 671–681. Springer-Verlag, 1997.

[163] 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, 1998.

[164] G. Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theoretical Computer Science*, 251(1–2):1–166, 2001.

[165] G. Sénizergues. L(A)=L(B)? a simplified decidability proof. *Theoretical Computer Science*, 281(1–2):555–608, 2002.

[166] M. Shields. Concurrent machines. *Computer Journal*, 28:449–465, 1985.

[167] J. Srba. Complexity of weak bisimilarity and regularity for BPA and BPP. In *Proceedings of the 7th International Workshop on Expressiveness in Concurrency (EXPRESS'00)*, volume 39 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2000. To appear.

[168] J. Srba. Complexity of weak bisimilarity and regularity for BPA and BPP. Technical report RS-00-16, BRICS Research Series, 2000.

[169] J. Srba. Basic process algebra with deadlocking states. *Theoretical Computer Science*, 266(1–2):605–630, 2001.

[170] J. Srba. Note on the tableau technique for commutative transition systems. Technical Report RS-01-50, BRICS Research Series, 2001.

[171] J. Srba. On the power of labels in transition systems. In *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *LNCS*, pages 277–291. Springer-Verlag, 2001.

[172] J. Srba. On the power of labels in transition systems. Technical Report RS-01-19, BRICS Research Series, 2001.

[173] J. Srba. Applications of the existential quantification technique. In *Proceedings of the 4th International Workshop on Verification of Infinite-State Systems (INFINITY'02)*, pages 151–152. Technical report, Faculty of Informatics, Masaryk University Brno, FIMU-RS-2002-04, 2002. Short presentation.

[174] J. Srba. Complexity of weak bisimilarity and regularity for BPA and BPP. *Mathematical Structures in Computer Science*, 2002. To appear.

[175] J. Srba. Note on the tableau technique for commutative transition systems. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'02)*, volume 2303 of *LNCS*, pages 387–401. Springer-Verlag, 2002.

[176] J. Srba. Roadmap of infinite results. *Bulletin of the European Association for Theoretical Computer Science*, 78:163–175, Oct. 2002. Columns: Concurrency.

[177] 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 *LNCS*, pages 535–546. Springer-Verlag, 2002.

[178] J. Srba. Strong bisimilarity and regularity of basic process algebra is PSPACE-hard. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *LNCS*, pages 716–727. Springer-Verlag, 2002.

[179] J. Srba. Strong bisimilarity of simple process algebras: Complexity lower bounds. Technical Report RS-02-16, BRICS Research Series, 2002.

[180] J. Srba. Undecidability of weak bisimilarity for PA-processes. In *Proceedings of the 6th International Conference on Developments in Laguage Theory (DLT'02)*, LNCS. Springer-Verlag, 2002. To appear.

[181] J. Srba. Undecidability of weak bisimilarity for pushdown processes. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 579–593. Springer-Verlag, 2002.

[182] C. Stirling. Modal logics for communicating systems. *Theoretical Computer Science*, 49(2-3):311–347, 1987.

[183] C. Stirling. Local model checking games. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *LNCS*, pages 1–11. Springer-Verlag, 1995.

[184] C. Stirling. Decidability of bisimulation equivalence for normed pushdown processes. *Theoretical Computer Science*, 195(2):113–131, 1998.

[185] C. Stirling. Decidability of bisimulation equivalence for pushdown processes. 2000. Submitted for publication.

[186] C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, 255(1–2):1–31, 2001.

[187] C. Stirling. Decidability of weak bisimilarity for a subset of basic parallel processes. In *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'01)*, volume 2030 of *LNCS*, pages 379–393. Springer-Verlag, 2001.

[188] C. Stirling. Deciding DPDA equivalence is primitive recursive. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *LNCS*, pages 821–832. Springer-Verlag, 2002.

[189] J. Stříbrná. Hardness results for weak bisimilarity of simple process algebras. In *Proceedings of the MFCS'98 Workshop on Concurrency*, volume 18 of *Electronic Notes in Theoretical Computer Science*. Springer-Verlag, 1998.

[190] W. Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science (extended abstract). In *Proceedings of the 4th International Joint Conference CAAP/FASE, Theory and Practice of Software Development (TAPSOFT'93)*, volume 668 of *LNCS*, pages 559–568. Springer-Verlag, 1993.

[191] D. Turi and J. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Mathematical Structures in Computer Science*, 8(5):481–540, 1998.

[192] A. Valmari. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (ICATPN'90): Advances in Petri Nets*, volume 483 of *LNCS*, pages 491–515. Springer-Verlag, 1991.

[193] R. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, CWI/Vrije Universiteit, 1990.

[194] R. van Glabbeek. The linear time—branching time spectrum. In *Proceedings of the 1st Internatinal Conference on Theories of Concurrency: Unification and Extension (CONCUR'90)*, volume 458 of *LNCS*, pages 278–297. Springer-Verlag, 1990.

[195] R. van Glabbeek. The linear time – branching time spectrum II (the semantics of sequential systems with silent moves). In *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *LNCS*, pages 66–81. Springer-Verlag, 1993.

[196] R. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions (extended abstract). In *Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science (MFCS'89)*, volume 379 of *LNCS*, pages 237–248. Springer-Verlag, 1989.

[197] R. van Glabbeek and W. Weijland. Branching time and abstraction in bisimulation semantics. *Information Processing Letters*, 89:613–618, 1989.

[198] R. van Glabbeek and W. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[199] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Annual IEEE Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society Press, 1986.

[200] W. Vogler. Deciding history preserving bisimilarity. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming (ICALP'91)*, volume 510 of *LNCS*, pages 493–505. Springer-Verlag, 1991.

[201] W. Vogler. *Modular construction and partial order semantics of Petri nets*, volume 625 of *LNCS*. Springer-Verlag, 1992.

[202] D. Walker. Bisimulation and divergence. *Information and Computation*, 85(2):202–241, 1990.

[203] Winskel and M. Nielsen. Models for concurrency. In *Handbook of Logic in Computer Science*, volume 4, *Semantic Modelling*, pages 1–148. Oxford University Press, 1995.

[204] W. Yi. Real–time behaviour of asynchronous agents. In *Proceedings of the 1st Internatinal Conference on Theories of Concurrency: Unification and Extension (CONCUR'90)*, number 458 in LNCS. Springer–Verlag, 1990.