Open access • Book Chapter • DOI:10.1017/CBO9780511792588.004

The algorithmics of bisimilarity — Source link

Luca Aceto, Anna Ingólfsdóttir, Jiri Srba

Institutions: Aalborg University

Published on: 01 Oct 2011

Topics: Formal equivalence checking, Process calculus, Equivalence (formal languages), Preorder and Principle of compositionality

The Algorithmics of Bisimilarity

Luca Aceto

Anna Ingolfsdottir

ICE-TCS[1]*, School of Computer Science, Reykjavík University*
Kringlan 1, 103 Reykjavík, Iceland

Jiří Srba

Department of Computer Science, Aalborg University
9220 Aalborg Ø, Denmark

Contents

0.1 Introduction

A model for reactive computation, for example that of labelled transition systems (Keller, 1976), or a process algebra, such as ACP (Baeten and Weijland, 1990), CCS (Milner, 1989) or CSP (Hoare, 1985), can be used to describe both implementations of processes and specifications of their expected behaviours. Process algebras and labelled transition systems therefore naturally support the so-called *single-language approach* to process theory—that is, the approach in which a single language is used to describe both actual processes and their specifications. An important ingredient of the theory of these languages and their associated semantic models is therefore a notion of behavioural equivalence or behavioural approximation between processes. One process description, say SYS, may describe an implementation, and another, say SPEC, may describe a specification of the expected behaviour. To say that SYS and SPEC are equivalent is taken to indicate that these two processes describe essentially the same behaviour, albeit possibly at different levels of abstraction or refinement. To say that, in some formal sense, SYS is an approximation of SPEC means roughly that every aspect of the behaviour of this process is allowed by the specification SPEC, and thus that nothing unexpected can happen in the behaviour of SYS. This approach to program verification is also sometimes called *implementation verification* or *equivalence checking*.

Designers using implementation verification to validate their (models of) reactive systems need only learn one language to describe both their systems and their specifications, and can benefit from the intrinsic compositionality of their descriptions—at least when they are using a process algebra for denoting the labelled transition systems in their models and an equivalence (or preorder) that is preserved by the operations in the algebra. Moreover, specifications presented as labelled transitions or as terms in a process algebra are rather detailed and typically describe both what a correct system should do and what behaviour it *cannot* afford. An implementation that conforms to such a detailed specification offers excellent correctness guarantees.

The single-language approach to the specification and verification of reactive systems underlies much of the work on process algebras and several of the early classic textbooks on those languages, such as the above-mentioned ones, present verifications of non-trivial concurrent systems using equivalence or preorder checking. (See also the book (Baeten, 1990), which is entirely devoted to early applications of the process algebra ACP.) It became, however, clear rather early on in the development of verification techniques from the field of process algebra and in their application in the verification of case studies that tool support was necessary in the analysis of the behaviour of models of

real-life computing systems. This realization was, amongst others, a powerful incentive for the work on the development of software tools for computer-aided verification based on process-algebraic verification techniques and on the model of labelled transition systems underlying them. Examples of such tools are the Edinburgh Concurrency Workbench and its successors (Cleaveland, Parrow and Steffen, 1993; Cleaveland and Sims, 1996), CADP (Garavel, Lang, Mateescu and Serwe, 2007) and mCRL2 (Groote, Keiren, Mathijssen, Ploeger, Stappers, Tankink, Usenko, van Weerdenburg, Wesselink, Willemse and van der Wulp, 2008).

In order to develop suitable tools for computer-aided verification and to understand their limitations, researchers needed to answer some of the most basic and fundamental questions that are naturally associated with the algorithmic problems underlying implementation verification.

- What equivalences or preorders over labelled transition systems are decidable, and over what classes of labelled transition systems?
- What equivalences or preorders are efficiently decidable over finite labelled transition systems or parallel compositions of such systems? And if so, by what algorithms?
- Are there equivalences that are (efficiently) decidable over classes of infinite labelled transition systems that can be 'finitely specified'?
- What is the computational complexity of checking decidable equivalences or preorders over 'interesting' classes of labelled transition systems? Can one come up with matching lower and upper bounds?

Apart from their usefulness in the development of the algorithmics of implementation verification, the questions above are of intrinsic and fundamental scientific interest within the field of concurrency theory and related fields of computer science. Indeed, the results that have been obtained over the years in an attempt to solve the algorithmic problems associated with equivalence and preorder checking over labelled transition systems and related models have put the theory of processes on a par with the classic theory of automata and formal languages (see, for instance, the textbooks (Hopcroft and Ullman, 1979; Sipser, 2006)) in terms of depth and elegance.

In this chapter, we shall address some of the above-mentioned questions and present classic results that have been achieved in the literature on concurrency theory since the 1980s. We shall mostly focus on (variations on) bisimilarity as the chosen notion of equivalence between (states in) labelled transition systems, but, where appropriate, we shall also hint at results that have been obtained for other equivalences and preorders in the literature. Our presentation will largely follow classic lines of exposition in the field of concurrency theory. However,

in Sections 0.3.2-0.3.3, we shall present some well-known results in a new light, using the view of bisimilarity and other co-inductively defined process semantics as games. We shall also highlight some general proof techniques based on tableaux techniques and on the so-called 'Defender's forcing technique' (Jančar and Srba, 2008). We feel that the use of these techniques makes the proofs of some results more transparent than the ones originally presented in the literature. Finally, where appropriate, we have elected to present proofs that allow us to introduce general and reusable proof techniques, even though they do not yield the best possible complexity bounds for some algorithmic problems. In such cases, we always give references to the papers offering the best known complexity results.

Roadmap of the chapter. The chapter is organized as follows. We begin by surveying the classic algorithms by Kanellakis and Smolka, and Paige and Tarjan, for computing bisimilarity over finite labelled transition systems in Section 0.2. There we also briefly mention the basic ideas underlying symbolic algorithms for bisimilarity checking based on reduced ordered binary decision diagrams. Section 0.3 is devoted to results on the computational complexity of bisimilarity, and related behavioural relations, over finite processes. In that section, we consider the complexity of the relevant decision problems when measured as a function of the 'size' of the input labelled transition system, as well as the one measured with respect to the length of the description of a parallel composition of labelled transition systems. In Section 0.4, we offer a glimpse of the plethora of remarkable decidability and complexity results that have been obtained for bisimilarity over classes of infinite-state processes. We conclude this chapter with a discussion of the main uses of bisimilarity checking in verification and tools (Section 0.5).

0.2 Classic algorithms for bisimilarity over finite labelled transition systems

In this section, we consider one of the most fundamental algorithmic problems associated with equivalence checking over labelled transition systems, namely the problem of deciding bisimilarity over *finite* labelled transition systems—that is, labelled transition systems with a finite set of states and finitely many transitions. (See Definition 2.2.1 in Chapter 1 of this book.) It is well known that deciding language equivalence over this class of labelled transition systems is PSPACE-complete (Hunt, Rosenkrantz and Szymanski, 1976). (See also (Stockmeyer and Meyer, 1973).) In sharp contrast with this negative result, efficient algorithms have been developed for deciding strong bisimilarity over fi-

nite labelled transition systems. Our main aim in this section is to present a brief exposition of two classic such algorithms that are due to Kanellakis and Smolka (Kanellakis and Smolka, 1983) (see also the journal version (Kanellakis and Smolka, 1990)) and to Paige and Tarjan (Paige and Tarjan, 1987). Both these algorithms are essentially based on the characterization of strong bisimilarity as a largest fixed point discussed in Section 3.8.1 in Chapter 1 in the book, and Tarski's fixed point theorem (Theorem 3.3.9 in Chapter 1 in the book) plays an important role in establishing their correctness.

For the sake of readability, we recall here the classic notion of stratified bisimulation relations, which are also known as approximants to bisimilarity. We refer the interested reader to Chapter 1 in this book for more information.

Definition 0.2.1 The *stratified bisimulation relations* (Milner, 1980; Hennessy and Milner, 1985) $\sim_k \subseteq Pr \times Pr$ for $k \in \mathbb{N}$ are defined as follows:

- $E \sim_0 F$ for all $E, F \in Pr$
- $E \sim_{k+1} F$ iff for each $a \in Act$: if $E \stackrel{a}{\longrightarrow} E'$ then there is $F' \in Pr$ such that $F \stackrel{a}{\longrightarrow} F'$ and $E' \sim_k F'$; and if $F \stackrel{a}{\longrightarrow} F'$ then there is $E' \in Pr$ such that $E \stackrel{a}{\longrightarrow} E'$ and $E' \sim_k F'$.

Given a labelled transition system $(Pr, Act, \longrightarrow)$, we define

$$\mathsf{next}(E, a) = \{E' \in Pr \mid E \stackrel{a}{\longrightarrow} E'\}$$

for $E \in Pr$ and $a \in Act$. We also define

$$\mathsf{next}(E, *) = \bigcup_{a \in Act} \mathsf{next}(E, a) \ .$$

A labelled transition system is *image-finite* iff the set $\mathsf{next}(E, a)$ is finite for every $E \in Pr$ and $a \in Act$.

The following lemma is a standard one.

Lemma 0.2.2 ((Hennessy and Milner, 1985)) Assume that $(Pr, Act, \longrightarrow)$ is an image-finite labelled transition system and let $E, F \in Pr$. Then $E \sim F$ iff $E \sim_k F$ for all $k \in \mathbb{N}$.

The characterization of bisimilarity given in the above lemma immediately yields an algorithm skeleton for computing bisimilarity over any finite labelled transition system, namely:

Let $\sim_0$ be the universal relation over the set of states of the input labelled transition system. For each $i > 0$, compute the ith approximant of bisimilarity $\sim_i$ until $\sim_i = \sim_{i-1}$.

If the input labelled transition system is finite, the computation of each approximant of bisimilarity can obviously be carried out effectively. Moreover, a very rough analysis indicates that the non-increasing sequence of approximants stabilizes in $O(n^2)$ iterations, where n is the number of states in the labelled transition system. But how efficiently can the above algorithm skeleton be implemented? How can one best compute the ith approximant of bisimilarity, for $i \geq 1$, from the previous one? And what is the complexity of optimal algorithms for the problem of computing bisimilarity over finite labelled transition systems?

The algorithms that we shall present in what follows indicate that, in sharp contrast with well-known results in formal language theory, the problem of deciding bisimilarity over finite labelled transition systems can be solved very efficiently. In Section 0.3.2, we shall discuss lower bounds on the complexity of checking bisimilarity over finite labelled transition systems. As confirmed by further results surveyed in later sections in this chapter, bisimilarity affords remarkable complexity and decidability properties that set it apart from other notions of equivalence and preorder over various classes of labelled transition systems.

0.2.1 Preliminaries

Let $(Pr, Act, \longrightarrow)$ be a finite labelled transition system. An equivalence relation over the set of states Pr can be represented as a *partition* of Pr—that is, as a set $\{B_0, \ldots, B_k\}$, $k \geq 0$, of non-empty subsets of Pr such that

- $B_i \cap B_j = \emptyset$, for all $0 \leq i < j \leq k$, and
- $Pr = B_0 \cup B_1 \cup \cdots \cup B_k$.

The sets B_i in a partition are usually called *blocks*. In what follows, we shall typically not distinguish between partitions and their associated equivalence relations.

Let π and π' be two partitions of Pr. We say that π' is a *refinement* of π if for each block $B' \in \pi'$ there exists some block $B \in \pi$ such that $B' \subseteq B$. Observe that if π' is a refinement of π, then the equivalence relation associated with π' is included in the one associated with π.

The algorithms for computing bisimilarity due to Kanellakis and Smolka, and Paige and Tarjan, compute successive refinements of an initial partition π_{init} and converge to the largest strong bisimulation over the input finite labelled transition system. Algorithms that compute strong bisimilarity in this fashion are often called *partition-refinement algorithms* and reduce the problem of computing bisimilarity to that of solving the so-called *relational coarsest partitioning* problem (Kanellakis and Smolka, 1990; Paige and Tarjan, 1987). In

what follows, we shall present the main ideas underlying the above-mentioned algorithms. We refer the interested reader to the survey paper (Cleaveland and Sokolsky, 2001) for more information and for other algorithmic approaches to deciding bisimilarity over finite labelled transition systems.

0.2.2 The algorithm by Kanellakis and Smolka

Let $\pi = \{B_0, \ldots, B_k\}$, $k \geq 0$, be a partition of the set of states Pr. The algorithm due to Kanellakis and Smolka is based on the notion of splitter.

Definition 0.2.3 A *splitter* for a block $B_i \in \pi$ is a block $B_j \in \pi$ such that, for some action $a \in Act$, some states in B_i have a-labelled transitions whose target is a state in B_j and others do not.

Intuitively, thinking of blocks as representing approximations of equivalence classes of processes with respect to strong bisimilarity, the existence of a splitter B_j for a block B_i in the current partition indicates that we have a reason for distinguishing two groups of sets of states in B_i, namely those that afford an a-labelled transition leading to a state in B_j and those that do not. Therefore, B_i can be split by B_j with respect to action a into the two new blocks:

$$\begin{aligned} B_i^1 &= \{s \mid s \in B_i \text{ and } s \xrightarrow{a} s', \text{ for some } s' \in B_j\} \quad \text{and} \\ B_i^2 &= B_i \setminus B_i^1 \ . \end{aligned}$$

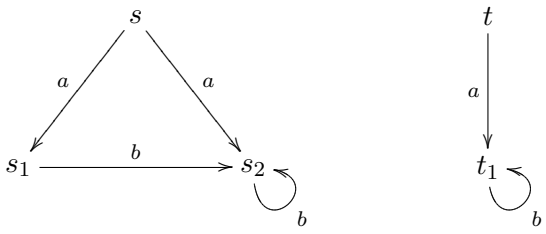
This splitting results in the new partition

$$\pi' = \{B_0, \ldots, B_{i-1}, B_i^1, B_i^2, B_{i+1}, \ldots, B_k\} \ ,$$

which is a refinement of π. The basic idea underlying the algorithm by Kanellakis and Smolka is to iterate the splitting of some block B_i by some block B_j with respect to some action a until no further refinement of the current partition is possible. The resulting partition is often called the *coarsest stable partition* and coincides with strong bisimilarity over the input labelled transition system when the initial partition π_{init} is chosen to be $\{Pr\}$. (We shall define the notion of stable partition precisely in Section 0.2.3 since it plays an important role in the algorithm proposed by Paige and Tarjan. The definitions we present in this section suffice to obtain an understanding of the algorithm by Kanellakis and Smolka.)

Before presenting the details of the algorithm by Kanellakis and Smolka, we illustrate the intuition behind the algorithm by means of a couple of examples.

Example 0.2.4 Consider the labelled transition system depicted below.

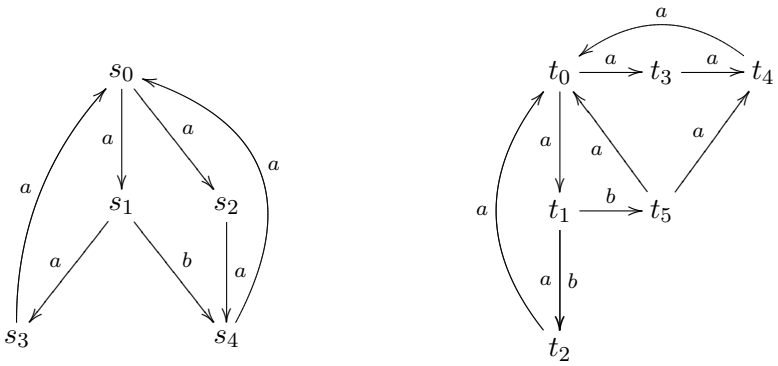


Since we are interested in computing bisimilarity, let the initial partition associated with this labelled transition system be $\{Pr\}$, where

$$Pr = \{s, s_1, s_2, t, t_1\} \ .$$

The block Pr is a splitter for itself. Indeed, some states in Pr afford a-labelled transitions while others do not. If we split Pr by Pr with respect to action a we obtain a new partition consisting of the blocks $\{s, t\}$ (the set of states that afford a-labelled transitions) and $\{s_1, s_2, t_1\}$ (the set of states that do not afford a-labelled transitions). Note that we would have obtained the same partition if we had done the splitting with respect to action b. You can now readily observe that neither of the blocks $\{s, t\}$ and $\{s_1, s_2, t_1\}$ can be split by the other with respect to any action. Indeed, states in each block are all bisimilar to one another, whereas states in different blocks are not.

Example 0.2.5 Consider now the following labelled transition system.



Let the initial partition associated with this labelled transition system be $\{Pr\}$, where

$$Pr = \{s_i, t_j \mid 0 \leq i \leq 4, 0 \leq j \leq 5\} \ .$$

The block Pr is a splitter for itself. Indeed, some states in Pr afford b-labelled transitions while others do not. If we split Pr by Pr with respect to action b we

```
function split(B, a, π)
choose some state s ∈ B
B1, B2 := ∅
for each state t ∈ B do
        if s and t can reach the same set of blocks in π via a-labelled transitions
        then B1 := B1 ∪ {t}
        else B2 := B2 ∪ {t}
if B2 is empty then return {B1}
else return {B1, B2}
```

Fig. 0.1. The function $\mathrm{split}(B, a, \pi)$

obtain a new partition consisting of the blocks

$$\{s_1, t_1\} \text{ and } \{s_i, t_j \mid 0 \leq i \leq 4, 0 \leq j \leq 5 \text{ with } i, j \neq 1\} \ .$$

Note now that the former block is a splitter for the latter one with respect to action a. Indeed only states s_0 and t_0 in that block afford a-labelled transitions that lead to a state in the block $\{s_1, t_1\}$. The resulting splitting yields the partition

$$\{\{s_0, t_0\}, \{s_1, t_1\}, \{s_i, t_j \mid 2 \leq i \leq 4, 2 \leq j \leq 5\}\} \ .$$

The above partition can be refined further. Indeed, some states in the third block have a-labelled transitions leading to states in the first block, but others do not. Therefore the first block is a splitter for the third one with respect to action a. The resulting splitting yields the partition

$$\{\{s_0, t_0\}, \{s_1, t_1\}, \{s_3, s_4, t_2, t_4, t_5\}, \{s_2, t_3\}\} \ .$$

We continue by observing that the block $\{s_3, s_4, t_4, t_2, t_5\}$ is a splitter for itself with respect to action a. For example $t_5 \xrightarrow{a} t_4$, but the only a-labelled transition from s_4 is $s_4 \xrightarrow{a} s_0$. The resulting splitting yields the partition

$$\{\{s_0, t_0\}, \{s_1, t_1\}, \{t_5\}, \{s_3, s_4, t_2, t_4\}, \{s_2, t_3\}\} \ .$$

We encourage you to continue refining the above partition until you reach the coarsest stable partition. What is the resulting partition?

The pseudo-code for the algorithm by Kanellakis and Smolka is given in Figure 0.2. The algorithm uses the function $\mathrm{split}(B, a, \pi)$ described in Figure 0.1, which given a partition π, a block B in π and an action a, splits B with respect to each block in π and action a. For example, considering the labelled transition system in Example 0.2.5, the call

$$\mathrm{split}(\{s_1, t_1\}, b, \{\{s_0, t_0\}, \{s_1, t_1\}, \{t_5\}, \{s_3, s_4, t_2, t_4\}, \{s_2, t_3\}\})$$

```
π := {Pr}
changed := true
while changed do
        changed := false
        for each block B ∈ π do
                for each action a do
                        sort the a-labelled transitions from states in B
                        if split(B, a, π) = {B1, B2} ≠ {B}
                        then refine π by replacing B with B1 and B2, and set changed
                                to true
```

Fig. 0.2. The algorithm by Kanellakis and Smolka

returns the pair $(\{s_1\}, \{t_1\})$ because the only block in the partition that can be reached from s_1 via a b-labelled transition is $\{s_3, s_4, t_2, t_4\}$, whereas t_1 can also reach the block $\{t_5\}$.

In order to decide efficiently whether the sets of blocks that can be reached from s and t are equal, the algorithm orders the transitions from each state lexicographically by their labels, and transitions with the same label are ordered by the block in the partition that contains the target state of the transition. Note that the algorithm in Figure 0.2 sorts the a-labelled transitions from states in a block B before calling split(B, a, π). This is necessary because a splitting of a block may change the ordering of transitions into that block. The lexicographic sorting of transitions can be done in $O(m+ \mid Act \mid)$ time and space, and therefore in $O(m)$ time and space because the number of actions can be taken to be at most m, using a classic algorithm from (Aho, Hopcroft and Ullman, 1974).

Theorem 0.2.6 (Kanellakis and Smolka) When applied to a finite labelled transition system with n states and m transitions, the algorithm by Kanellakis and Smolka computes the partition corresponding to strong bisimilarity in time $O(nm)$.

Proof We first argue that the algorithm by Kanellakis and Smolka is correct, that is, that when it reaches the coarsest stable partition π that partition is the one corresponding to strong bisimilarity over the input labelled transition system. To see this, let π_i, $i \geq 0$, denote the partition after the ith iteration of the outermost loop in the algorithm in Figure 0.2. Below, we shall use π_i to denote also the equivalence relation induced by the partition π_i. Recall, furthermore, that $\sim_i$ denotes the ith approximant to bisimilarity.

Observe, first of all, that

$$\sim \subseteq \sim_i \subseteq \pi_i$$

holds for each $i \geq 0$. This can be easily proved by induction on i. It follows that $\sim \subseteq \pi$.

Conversely, if π is a partition that cannot be refined further, then it is not hard to see that π is a post-fixed point of the function $F_\sim$ that was used in Chapter 1 of the book to characterize $\sim$ as a largest fixed point. Since $\sim$ is the largest post-fixed point of $F_\sim$, it follows that $\pi \subseteq \sim$. Therefore $\sim = \pi$.

As far as the complexity of the algorithm is concerned, note that the main loop of the algorithm is repeated at most $n - 1$ times. Indeed, the number of blocks in π must be between one and n, and this number increases at each iteration of the outermost loop of the algorithm that leads to a refinement of π. The calls to the function split take $O(m)$ time at each iteration of the main loop. Indeed, that function is called for each block and each action at most once, and in each call it considers each transition of every state in the block at most once. As we already remarked above, the sorting of transitions can be done in $O(m)$ time at each iteration of the main loop. Therefore we achieve the claimed $O(mn)$ complexity bound. □

Exercise 0.2.7 Fill in the details in the proof of correctness of the algorithm that we sketched above.

Remark 0.2.8 The algorithm by Kanellakis and Smolka has also $O(n + m)$ space complexity.

A natural question to ask at this point is whether the algorithm by Kanellakis and Smolka is optimal. As we shall see in the following section, the answer to this question is negative. Indeed, the time performance of a partition-refinement algorithm like the one we just discussed can be improved substantially by means of some clever use of data structures.

Remark 0.2.9 For the special case in which, for each action a and process s, the set

$$\{s' \mid s \xrightarrow{a} s'\}$$

has size at most a constant c, Kanellakis and Smolka presented in (Kanellakis and Smolka, 1983) (see also (Kanellakis and Smolka, 1990)) an $O(c^2 n \log n)$-time algorithm to decide strong bisimilarity. They also conjectured the existence of an $O(m \log n)$-time algorithm for the problem. This conjecture was confirmed by Paige and Tarjan by means of the algorithm we will present in the following section.

0.2.3 The algorithm by Paige and Tarjan

The algorithm by Paige and Tarjan is also based on the idea of partition refinement. However, by making use of more complex data structures, it significantly improves on the efficiency of the algorithm by Kanellakis and Smolka we discussed in the previous section. The low complexity of the algorithm by Paige and Tarjan is achieved by using information about previous splits to improve on the efficiency of future splits. In particular, each state can only appear in a number of splitters that is logarithmic in the number of states in the input labelled transition system.

In order to simplify the presentation of the algorithm, throughout this section we shall assume that the input finite labelled transition system is over a single action a. The extension of the algorithm to arbitrary finite labelled transition systems is only notationally more complex. (See also Exercise 0.3.12, which indicates that a single action is as general as any finite set of actions, also from the point of view of the complexity of algorithms.)

We begin our presentation of the algorithm by Paige and Tarjan by defining formally the notion of stable partition. In what follows, given a set of states S, we define:

$$\mathrm{pre}(S) \quad = \quad \{s \mid s \xrightarrow{a} s' \text{ for some } s' \in S\} \ .$$

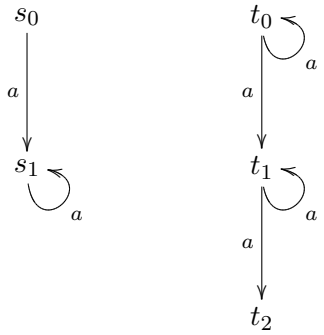
Definition 0.2.10 ((Coarsest) Stable partition)

- A set $B \subseteq Pr$ is stable with respect to a set $S \subseteq Pr$ if
  - either $B \subseteq \mathrm{pre}(S)$
  - or $B \cap \mathrm{pre}(S) = \emptyset$.
- A partition π is stable with respect to a set S if so is each block $B \in \pi$.
- A partition π is stable with respect to a partition π' if π is stable with respect to each block in π'.
- A partition π is stable if it is stable with respect to itself.

The coarsest stable refinement (of a partition π_{init}) is a stable partition that is refined by any other stable partition (that refines π_{init}).

Note that $B \subseteq Pr$ is stable with respect to a block $S \subseteq Pr$ if, and only if, S is *not* a splitter for B. (The notion of splitter was presented in Definition 0.2.3.)

Example 0.2.11 Consider the following labelled transition system.



The block $\{s_i, t_i \mid 0 \leq i \leq 1\}$ is stable with respect to the set of states $\{s_i, t_j \mid 0 \leq i \leq 1,\ 0 \leq j \leq 2\}$. On the other hand, the block $\{s_i, t_j \mid 0 \leq i \leq 1,\ 0 \leq j \leq 2\}$ is not stable with respect to itself because

$$\mathrm{pre}(\{s_i, t_j \mid 0 \leq i \leq 1,\ 0 \leq j \leq 2\}) = \{s_i, t_i \mid 0 \leq i \leq 1\} \ .$$

The coarsest stable partition over the above labelled transition system that refines $\{Pr\}$ is

$$\{\{s_0, s_1\}, \{t_0\}, \{t_1\}, \{t_2\}\} \ .$$

Note that this is the partition associated with $\sim$ over the above labelled transition system.

Exercise 0.2.12 Show that $\sim$ is the coarsest stable partition that refines $\{Pr\}$ over any finite labelled transition system with $Act = \{a\}$.

The basic idea behind the algorithm proposed by Paige and Tarjan is that, given a partition π of Pr, one can identify two different types of blocks B that act as splitters for π: simple and compound splitters.

Simple splitters are used to split blocks in π into two disjoint subsets as done in the algorithm by Kanellakis and Smolka. Below, we rephrase this notion using the definitions introduced above.

Definition 0.2.13 (Simple splitting) Let π be a partition and let B be a set of states in Pr. We write $\mathrm{split}(B, \pi)$ for the refinement of π obtained as follows.

For each block $B' \in \pi$ such that B' is not stable with respect to B, replace B' by the blocks

$$\begin{array}{rcl} B'_1 & = & B' \cap \mathrm{pre}(B) \quad \text{and} \\ B'_2 & = & B' \setminus \mathrm{pre}(B) \ . \end{array}$$

We call B a *splitter* for π when $\mathrm{split}(B, \pi) \neq \pi$. In that case, we also say that

π is refined with respect to B, and that $\mathrm{split}(B, \pi)$ is the partition that results from that refinement.

For example, consider the labelled transition system in Example 0.2.11 and the partition $\{Pr\}$. As we already remarked, the block Pr is not stable with respect to itself. Moreover,

$$\mathrm{split}(Pr, \{Pr\}) = \{\{s_0, s_1, t_0, t_1\}, \{t_2\}\} \ .$$

Therefore Pr is a splitter for $\{Pr\}$.

We observe the following useful properties of the function split and of the notion of stability.

Lemma 0.2.14

(1) Stability is preserved by refinement—that is, if π refines π' and π' is stable with respect to a set of states S, then so is π.
(2) Stability is preserved by union—that is, if π is stable with respect to sets S_1 and S_2, then π is also stable with respect to $S_1 \cup S_2$.
(3) Assume that $B \subseteq Pr$. Let π_1 and π_2 be two partitions of Pr such that π_1 refines π_2. Then $\mathrm{split}(B, \pi_1)$ refines $\mathrm{split}(B, \pi_2)$.
(4) Assume that $B, B' \subseteq Pr$. Let π be a partition of Pr. Then

$$\mathrm{split}(B, \mathrm{split}(B', \pi)) = \mathrm{split}(B', \mathrm{split}(B, \pi)) \ ,$$

that is, split is commutative.

Exercise 0.2.15 Prove the above lemma.

As a stepping stone towards introducing the algorithm by Paige and Tarjan, consider the following abstract version of the algorithm by Kanellakis and Smolka. The algorithm keeps a partition π that is initially the initial partition π_{init} and refines it until it reaches the coarsest stable partition that refines π_{init}. It repeats the following steps until π is stable:

(1) Find a set S that is a union of some of the blocks in π and is a splitter of π.
(2) Replace π by $\mathrm{split}(S, \pi)$.

This refinement algorithm works just as well if one restricts oneself to using only blocks of π as splitters. However, allowing the use of unions of blocks as splitters is one of the key ideas behind the algorithm by Paige and Tarjan.

In order to implement the algorithm efficiently, it is useful to reduce the problem to that of considering labelled transition systems without deadlocked states, i.e., to labelled transition systems in which each state has an outgoing

transition. This can be done easily by preprocessing the initial partition π_{init} by splitting each block $B \in \pi_{\text{init}}$ into

$$\begin{aligned} B_1 &= B \cap \text{pre}(Pr) \quad \text{and} \\ B_2 &= B \setminus \text{pre}(Pr) \ . \end{aligned}$$

Note that the blocks B_2 generated in this way will never be split again by the refinement algorithm. Therefore we can run the refinement algorithm starting from the partition

$$\pi'_{\text{init}} = \{B_1 \mid B \in \pi_{\text{init}}\} \ ,$$

which is a partition of the set $\text{pre}(Pr)$. The coarsest stable refinement of π'_{init} together with the blocks of the form B_2 for each $B \in \pi_{\text{init}}$ yield the coarsest stable refinement of π_{init}.

Exercise 0.2.16 Prove the claims we just made.

In order to achieve its time complexity, the Paige-Tarjan algorithm employs a very clever way of finding splitters. In addition to the current partition π, the algorithm maintains another partition X such that

- π is a refinement of X and
- π is stable with respect to X.

Initially, π is the initial partition π_{init} and $X = \{Pr\}$. The abstract version of the algorithm by Paige and Tarjan repeats the following steps until $\pi = X$:

(1) Find a block $S \in X \setminus \pi$.
(2) Find a block $B \in \pi$ such that $B \subseteq S$ and $\mid B \mid \leq \frac{|S|}{2}$.
(3) Replace S within X with the two sets B and $S \setminus B$.
(4) Replace π with $\text{split}(S \setminus B, \text{split}(B, \pi))$.

The efficiency of the above algorithm relies on the heuristic for the choice of the block B at step 2 and on the use of what is usually called *three-way splitting* to implement step 4 efficiently. We now proceed to introduce the basic ideas behind the notion of three-way splitting.

Suppose that we have a partition π that is stable with respect to a set of states S that is a union of some of the blocks in π. Assume also that π is refined first with respect to a nonempty set $B \subset S$ and then with respect to $S \setminus B$. We have that:

- Refining π with respect to B splits a block $D \in \pi$ into two blocks $D_1 = D \cap \text{pre}(B)$ and $D_2 = D \setminus \text{pre}(B)$ if, and only if, D is *not* stable with respect to B.

- Refining further $\mathrm{split}(B, \pi)$ with respect to $S \setminus B$ splits the block D_1 into two blocks $D_{11} = D_1 \cap \mathrm{pre}(S \setminus B)$ and $D_{12} = D_1 \setminus D_{11}$ if, and only if, D_1 is *not* stable with respect to $S \setminus B$.

A block S of the partition X is *simple* if it is also a block of π and is *compound* otherwise. Note that a compound block S contains at least two blocks of π. Such a block S can be partitioned into B and $S \setminus B$ in such a way that both the above-mentioned properties hold. The three-way splitting procedure outlined above gives the refinement of a partition with respect to a compound splitter.

Observe that refining $\mathrm{split}(B, \pi)$ with respect to $S \setminus B$ does *not* split the block D_2. This is because $D_2 \subseteq \mathrm{pre}(S \setminus B)$. Indeed, since π is stable with respect to S, we have that either $D \subseteq \mathrm{pre}(S)$ or $D \cap \mathrm{pre}(S) = \emptyset$. If D is *not* stable with respect to B, it holds that

$$D \not\subseteq \mathrm{pre}(B) \text{ and } D \cap \mathrm{pre}(B) \neq \emptyset \ .$$

Therefore, $D \subseteq \mathrm{pre}(S)$. Since $D_2 = D \setminus \mathrm{pre}(B)$, we can immediately infer that $D_2 \subseteq \mathrm{pre}(S \setminus B)$, as claimed. Moreover,

$$D_{12} = D_1 \setminus D_{11} = D \setminus \mathrm{pre}(S \setminus B) = D_1 \cap (\mathrm{pre}(B) \setminus \mathrm{pre}(S \setminus B)) \ .$$

The identity

$$D_{12} = D_1 \cap (\mathrm{pre}(B) \setminus \mathrm{pre}(S \setminus B))$$

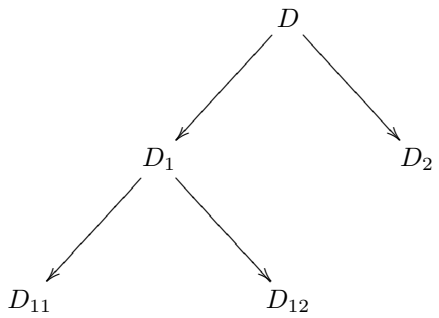
is the crucial observation underlying the implementation of the algorithm by Paige and Tarjan. We can visually depict the result of a three-way split of a block D as the binary tree depicted in Figure 0.3.

The time performance of the algorithm by Paige and Tarjan relies on the following observations. First of all note that each state in the input labelled transition system is in at most $\log n + 1$ blocks B used as refining sets. This is because each refining set is at most half the size of the previous one. Moreover, as shown by Paige and Tarjan, a refinement step with respect to a block B can be implemented in time

$$O(\mid B \mid + \sum_{s \in B} \mid \mathrm{pre}(\{s\}) \mid)$$

by means of a careful use of data structures. An $O(m \log n)$ time bound for the algorithm follows.

In order to perform three-way splitting efficiently, namely in time proportional to the cardinality of the block B chosen at step 2 of the algorithm, the Paige-Tarjan algorithm uses an integer variable $\mathrm{count}(s, S)$ for each state s and each block S. Intuitively, $\mathrm{count}(s, S)$ records the number of states in S that can be reached from s via a transition.



In the above figure, we use the following abbreviations:

$$\begin{array}{rcll} D_1 & = & D \cap \mathrm{pre}(B) & \\ D_2 & = & D \smallsetminus \mathrm{pre}(B) & \\ D_{11} & = & D \cap \mathrm{pre}(B) \cap \mathrm{pre}(S \smallsetminus B) & \text{and} \\ D_{12} & = & D \smallsetminus \mathrm{pre}(S \smallsetminus B) & . \end{array}$$

Fig. 0.3. Three-way splitting of a block

Exercise 0.2.17 Use the variables $\mathrm{count}(s, S)$ to decide in constant time to which sub-block of D a state $s \in D$ belongs.

Apart from the count variables, the Paige-Tarjan algorithm uses additional data structures to achieve its efficiency. Blocks and states are represented by records. Each block of the partition π is stored as a doubly linked list containing its elements and has an associated integer variable recording its cardinality. Each block of X points to a doubly linked list containing the blocks in π included in it. Finally, each block of π points to the block of X that includes it. The implementation of the algorithm uses also a set C that contains the compound blocks of X. Initially, $C = \{Pr\}$ and π is the initial partition π_{init}.

An efficient implementation of the Paige-Tarjan algorithm repeats the steps given in Figure 0.4, taken from (Paige and Tarjan, 1987, page 981), until the set of compound blocks becomes empty.

Theorem 0.2.18 (Paige and Tarjan) When applied to a finite labelled transition system with n states and m transitions using $\{Pr\}$ as the initial partition, the algorithm by Paige and Tarjan computes the partition corresponding to strong bisimilarity in time $O(m \log n)$.

We refer the interested readers to (Cleaveland and Sokolsky, 2001; Paige and Tarjan, 1987) for further details on the algorithm by Paige and Tarjan.

(1) Remove a compound block S from C. Find the first two blocks in the list of blocks of π that are included in S and let B be the smallest, breaking ties arbitrarily.
(2) Replace S with $S \setminus B$ and create a new simple block in X containing B as its only block of π. If S is still compound, put S back into C.
(3) Compute $\mathrm{pre}(B)$ and $\mathrm{count}(s, B)$. This is done as follows. Copy B into a temporary set B'. Compute $\mathrm{pre}(B)$ by scanning the edges with a target in B and adding the source of each such edge to $\mathrm{pre}(B)$ if it is not already contained in that set. During the count compute the count variables $\mathrm{count}(s, B)$.
(4) Replace π with $\mathrm{split}(B, \pi)$ as follows. For each block $D \in \pi$ such that $D \cap \mathrm{pre}(B) \neq \emptyset$, split D into $D_1 = D \cap \mathrm{pre}(B)$ and $D_2 = D \setminus D_1$. This is done by scanning the elements of $\mathrm{pre}(B)$. For each $s \in \mathrm{pre}(B)$, determine the block $D \in \pi$ containing it and create an associated block D' if one does not exist already. Move s from D to D'.
During the scanning, build a list of the blocks D that are split. After the scanning is done, process the list of split blocks as follows. For each such block with associated block D', mark D' as no longer being associated with D; eliminate the record for D if D is empty; if D is nonempty and the block of X containing D and D' has been made compound by the split, add this block to C.
(5) Compute $\mathrm{pre}(B) \setminus \mathrm{pre}(S \setminus B)$. This is done by scanning the transitions whose target is in B'. For each such edge $s \xrightarrow{a} s'$, add s to $\mathrm{pre}(B) \setminus \mathrm{pre}(S \setminus B)$ if s is not already in that set and $\mathrm{count}(s, B) = \mathrm{count}(s, S)$.
(6) Replace π with $\mathrm{split}(S \setminus B, \pi)$ as done in Step 4 above, but scanning $\mathrm{pre}(B) \setminus \mathrm{pre}(S \setminus B)$ in lieu of $\mathrm{pre}(B)$.
(7) Update the count variables. This is done by scanning the transitions whose target is in B'. For each such transition $s \xrightarrow{a} s'$, decrement $\mathrm{count}(s, S)$. If this count becomes zero, delete the count record and make the transition point to $\mathrm{count}(s, B)$. When the scan is done, discard B'.

Fig. 0.4. The refinement steps of the Paige-Tarjan algorithm

0.2.4 Computing bisimilarity, symbolically

The algorithms for computing bisimilarity we have presented above require that the input labelled transition system be fully constructed in advance. In practice, a labelled transition system describing a reactive system is typically specified as a suitable 'parallel composition' of some other labelled transition systems representing the behaviour of the system components. It is well known that the size of the resulting labelled transition system may grow exponentially with respect to the number of parallel components. This phenomenon is usually referred to as the *state-explosion problem* and is a major hindrance in the use of algorithms requiring the explicit construction of the input labelled transition system in the automatic verification of large reactive systems.

Amongst the approaches that have been proposed in order to deal with the state-explosion problem, we shall briefly present here the main ideas underlying a symbolic approach based on the use of *reduced ordered binary decision*

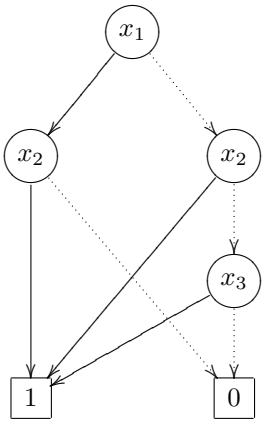


Fig. 0.5. Example of ROBDD where $x_1 < x_2 < x_3$

diagrams (ROBDDs) (Bryant, 1986; Bryant, 1992) to represent finite labelled transition systems symbolically. The use of ROBDDs often permits a succinct representation of finite objects, and has led to breakthrough results in the automation of the solution to many combinatorial problems. The use of ROBDDs in the computation of bisimilarity over finite labelled transition systems has been proposed in (Bouali and Simone, 1992).

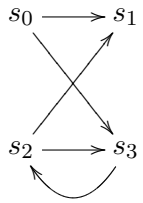
We recall here that an ROBDD represents a boolean function as a rooted directed acyclic graph. The non-terminal nodes in the graph are labelled with input variables and the terminal nodes are labelled by boolean constants. Each non-terminal node has two outgoing edges, one per possible value of the boolean variable labelling it. In each path from the root of the graph to a terminal node, the variables respect a given total variable ordering. Finally, an ROBDD contains neither redundant tests nor duplicate nodes. The former requirement means that there is no non-terminal node in the graph whose two outgoing edges lead to the same node. The latter condition ensures that the sharing of nodes in an ROBDD is maximal, in that there are no two distinct nodes in the graph that are the roots of isomorphic sub-graphs. An example of ROBDD is given in Figure 0.5 where the solid edges leaving nodes labelled with variables represent the truth assignment true, while the dotted edges correspond to the thruth assignment false. We can see that if e.g. x_1 is set to false, x_2 to false and x_3 to true, the whole boolean function evaluates to true as we reached the terminal node labelled with 1.

The crucial property of ROBDDs is their canonicity. This means that, for each boolean function with n arguments, there is exactly one ROBDD representing

it with respect to the variable ordering $x_1 < x_2 < \cdots < x_n$ and therefore two ROBDDs representing the same function with respect to that ordering are isomorphic. For example, a boolean function is a tautology if, and only if, its associated ROBDD consists of a single node labelled with 1 (true) and it is satisfiable if, and only if, its associated ROBDD does not consist of a single node labelled with 0 (false).

Encoding a labelled transition system as an ROBDD. The first step in using ROBBDs to compute bisimilarity over a finite labelled transition system is to encode the input labelled transition system. This is done as follows. The states in a finite labelled transition system with $n > 1$ states are represented by means of a function that associates with each state a distinct boolean vector of length $k = \lceil \log_2 n \rceil$. Each state s is therefore represented by a unique assignment of truth values to a vector of boolean variables $\vec{x} = (x_1, \ldots, x_k)$. The set of actions labelling transitions is encoded in exactly the same way. Each action a is therefore represented by a unique assignment of truth values to a vector of boolean variables $\vec{y} = (y_1, \ldots, y_\ell)$. The transition relation of the labelled transition system is represented by its characteristic function $\mathrm{Trans}(\vec{x}, \vec{y}, \vec{x'})$, which returns true exactly when $\vec{x}$, $\vec{y}$ and $\vec{x'}$ are the encodings of some state s, action a and state s', respectively, such that $s \stackrel{a}{\rightarrow} s'$. In the definition of the boolean formula describing the transition relation, the vector $\vec{x'} = (x'_1, \ldots, x'_k)$ consists of distinct fresh variables, which are often called the *target variables* and are used to encode the targets of transitions.

Example 0.2.19 Consider, for instance, the labelled transition system over a single action depicted below. (We have omitted the label of the transitions and we shall not consider it in the encoding for the sake of simplicity.)



To encode the states in this labelled transition system, we use two boolean variables x_1 and x_2. Each state s_i, $i \in \{0, 1, 2, 3\}$, is represented by the binary representation of the number i. The boolean formula describing the transition

relation is

$$\begin{gathered}(\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_1'} \wedge x_2') \\ \vee \\ (\overline{x_1} \wedge \overline{x_2} \wedge x_1' \wedge x_2') \\ \vee \\ (x_1 \wedge \overline{x_2} \wedge x_1' \wedge x_2') \\ \vee \\ (x_1 \wedge \overline{x_2} \wedge \overline{x_1'} \wedge x_2') \\ \vee \\ (x_1 \wedge x_2 \wedge x_1' \wedge \overline{x_2'}) \ .\end{gathered}$$

For example, the transition from state s_2 to state s_3 is represented by the disjunct

$$x_1 \wedge \overline{x_2} \wedge x_1' \wedge x_2'$$

in the above formula. The first two conjuncts say that the source of the transition is the state whose encoding is the bit vector $(1, 0)$, namely s_2, and the last two say that the target is the state whose encoding is the bit vector $(1, 1)$, namely s_3.

The boolean formula $\mathrm{Trans}(\vec{x}, \vec{y}, \vec{x'})$ can then be represented as an ROBDD choosing a suitable variable ordering. It is important to realize here that the size of the ROBDD for a boolean function depends crucially on the chosen variable ordering. Indeed, in the worst case, the size of an ROBDD is exponential in the number of variables. (See (Bryant, 1992) for a detailed, but very accessible, discussion of ROBDDs, their properties and their applications.) A useful property of the above-mentioned ROBDD representation of a labelled transition system is that the ROBDD describing the transition relation of a labelled transition system resulting from the application of the CCS operations of parallel composition, relabelling and restriction to a collection of component reactive systems (themselves represented as ROBDDs) is guaranteed to only grow linearly in the number of parallel components, provided an appropriate variable ordering is chosen—see (Enders, Filkorn and Taubner, 1993).

Remark 0.2.20 We remark that the above-stated linear upper bound applies to the size of the resulting ROBDD, but not necessarily to the intermediate ROBBDs that are constructed as stepping stones in the computation of the final result. However, in (Enders et al., 1993), the authors claim that, in their practical experiments, the size of the intermediate ROBBDs never exceeded the size of the result ROBBD.

(1) $\mathrm{Bis}(\vec{x}, \vec{z}) := 1$
(2) `repeat`
  (a) $\mathrm{Old}(\vec{x}, \vec{z}) := \mathrm{Bis}(\vec{x}, \vec{z})$
  (b) $\mathrm{Bis}(\vec{x}, \vec{z}) := \forall \vec{x'} \forall \vec{y}.\ \mathrm{Trans}(\vec{x}, \vec{y}, \vec{x'}) \Rightarrow (\exists \vec{z'}.\ \mathrm{Trans}(\vec{z}, \vec{y}, \vec{z'}) \wedge \mathrm{Bis}(\vec{x'}, \vec{z'})) \wedge \forall \vec{z'} \forall \vec{y}.\ \mathrm{Trans}(\vec{z}, \vec{y}, \vec{z'}) \Rightarrow (\exists \vec{x'}.\ \mathrm{Trans}(\vec{x}, \vec{y}, \vec{x'}) \wedge \mathrm{Bis}(\vec{x'}, \vec{z'}))$
(3) `until` $\mathrm{Bis}(\vec{x}, \vec{z}) = \mathrm{Old}(\vec{x}, \vec{z})$

Fig. 0.6. Algorithm for the symbolic computation of bisimilarity

Symbolic computation of bisimilarity. Like the aforementioned algorithms, the symbolic algorithm for computing bisimilarity is based upon the characterization of bisimilarity as a largest fixed point given by Tarski's fixed-point theorem. In order to implement the iterative computation of the largest fixed point efficiently using ROBDDs, we need to rephrase the function $F_\sim$ in a way that is suitable for symbolic computation using ROBDD operations. The algorithm for the symbolic computation of bisimilarity is presented in Figure 0.6. In that algorithm, we assume that we are given two copies of the input labelled transition system with transition relations expressed by the boolean formulae $\mathrm{Trans}(\vec{x}, \vec{y}, \vec{x'})$ and $\mathrm{Trans}(\vec{z}, \vec{y}, \vec{z'})$. The algorithm computes the ROBDD representation of the boolean formula $\mathrm{Bis}(\vec{x}, \vec{z})$ that encodes the characteristic function of the largest bisimulation over the input labelled transition system—that is, $\mathrm{Bis}(s, s')$ returns true exactly when s and s' are bisimilar states in the labelled transition system.

The algorithm in Figure 0.6 uses existential and universal quantification over boolean variables. These can be readily implemented in terms of the standard boolean operations. By way of example, if $f(x, y)$ is a boolean function then $\exists x.\ f(x, y)$ is a shorthand for $f(0, y) \vee f(1, y)$. We refer the interested readers to (Bouali and Simone, 1992; Cleaveland and Sokolsky, 2001) for further details on the efficient implementation of the ROBDD that encodes the right-hand side of the assignment at step 2b in the algorithm in Figure 0.6, as well as for a discussion of performance issues related to the appropriate choice of the variable ordering to be used in the computation.

0.2.5 Checking weak equivalences

The problem of checking observational equivalence (weak bisimilarity) over finite labelled transition systems can be reduced to that of checking strong bisimilarity using a technique called *saturation*. Intuitively, saturation amounts to

(1) first pre-computing the weak transition relation (see Section 5.1 in Chapter 1 in this book), and then
(2) constructing a new pair of finite processes whose original transitions are replaced with the weak transitions.

The question whether two states are weakly bisimilar now amounts to checking strong bisimilarity over the saturated systems. Since the computation of the weak transition relation can be carried out in polynomial time, the problem of checking for weak bisimilarity can also be decided in polynomial time. The same holds true for the problem of checking observational congruence (Milner, 1989). (See Section 5.1 in Chapter 1 in this book for the definition of observational congruence, which is there called rooted weak bisimilarity.)

Efficient algorithms are also available for deciding a variation on the notion of weak bisimilarity called *branching bisimilarity* (Glabbeek and Weijland, 1996) over finite labelled transition systems (see Chapter 1). Notably, Groote and Vaandrager have developed in (Groote and Vaandrager, 1990) an algorithm for checking branching bisimilarity that has time complexity $O(m \log m + mn)$ and space complexity $O(m + n)$.

ADD EXACT REFERENCE to CHAPTER 1 and SECTION

0.3 The complexity of checking bisimilarity over finite processes

In the previous section, we saw that, unlike the classic notion of language equivalence, bisimilarity can be efficiently decided over finite labelled transition systems. In particular, the time complexity of the algorithm by Paige and Tarjan offers an upper bound on the time complexity of checking bisimilarity over finite labelled transition systems. This naturally raises the question of whether it is possible to provide lower bounds on the computational complexity of computing bisimilarity over such structures. Indeed, computer scientists often consider a decidable computational problem 'solved' when they possess matching lower and upper bounds on its computational complexity.

Our order of business in this section will be to survey some of the most notable results pertaining to lower bounds on the computational complexity of bisimilarity checking for several classes of finite labelled transition systems. We shall also present some rather general proof techniques that yield perspicuous proofs of the results we discuss in what follows.

Since the notion of bisimulation game plays an important role in our presentation of the complexity and decidability results discussed in the remainder of this chapter, we now proceed to introduce it briefly. (We refer the reader to Chapter 1 in the book for a more comprehensive discussion and to (Aceto, Ingolfsdottir, Larsen and Srba, 2007) for a textbook presentation.)

0.3.1 Game characterization of bisimulation-like relations

We shall use a standard game-theoretic characterization of (bi)similarity—see, for instance, (Thomas, 1993; Stirling, 1995) and Chapter 1. A *bisimulation game* on a pair of processes (P_1, Q_1) is a two-player game between *Attacker* and *Defender*. The game is played in *rounds*. In each round the players change the *current pair of states* (P, Q) (initially $P = P_1$ and $Q = Q_1$) according to the following rules:

ADD EXACT REFERENCE TO CHAPTER 1 AND SECTION

(1) Attacker chooses either P or Q, an action a and performs a move $P \stackrel{a}{\longrightarrow} P'$ or $Q \stackrel{a}{\longrightarrow} Q'$, depending on whether he chose P or Q.
(2) Defender responds by choosing the other process (either Q or P) and performs a move $Q \stackrel{a}{\longrightarrow} Q'$ or $P \stackrel{a}{\longrightarrow} P'$ under the same action a.
(3) The pair (P', Q') becomes the (new) current pair of states.

A *play* (of the bisimulation game) is a sequence of pairs of processes formed by the players according to the rules mentioned above. A play is finite iff one of the players gets stuck (cannot make a move); the player who gets stuck loses the play and the other player is the winner. If the play is infinite then Defender is the winner.

We use the following standard fact—see (Thomas, 1993; Stirling, 1995) and Chapter 1.

ADD EXACT REFERENCE TO CHAPTER 1 AND SECTION

Proposition 0.3.1 It holds that $P \sim Q$ iff Defender has a winning strategy in the bisimulation game starting with the pair (P, Q), and $P \not\sim Q$ iff Attacker has a winning strategy in the corresponding game.

The rules of the bisimulation game can be easily modified in order to capture other co-inductively defined equivalences and preorders.

- In the *simulation preorder game*, Attacker is restricted to attack only from the (left-hand-side) process P. In the *simulation equivalence game*, Attacker can first choose a side (either P or Q) but after that he is not allowed to change the side any more.
- The *completed/ready simulation game* has the same rules as the simulation game but Defender is moreover losing in any configuration which breaks the extra condition imposed by the definition (i.e. P and Q should have the same set of initial actions in case of ready simulation, and their sets of initial actions should be both empty at the same time in case of completed simulation).
- In the *2-nested simulation preorder game*, Attacker starts playing from the left-hand-side process P and at most once during the play he is allowed to switch sides. (The soundness of this game follows from the characterization

provided in (Aceto, Fokkink and Ingolfsdottir, 2001).) In the *2-nested simulation equivalence game*, Attacker can initially choose any side but the sides can be changed at most once during the play.

0.3.2 Deciding bisimilarity over finite labelled transition systems is P-complete

The algorithmic results discussed in Section 0.2 indicate that strong, weak and branching bisimilarity can be decided over finite labelled transition systems more efficiently than language equivalence (and indeed than many other equivalences). This is good news for researchers and practitioners who develop and/or use software tools that implement various forms of bisimilarity checking. However, the size of the labelled transition systems on which the above-mentioned algorithms are run in practice is often huge. It is therefore natural to ask oneself whether one can devise efficient *parallel* algorithms for bisimilarity checking. Such algorithms might exploit the parallelism that is becoming increasingly available on our computers to speed up checking whether two processes are bisimilar or not. It would be very satisfying, as well as practically useful, if algorithms for checking bisimilarity could run on a highly parallel machine with a running time that is proportional to the logarithm of the size of the state space of the input labelled transition system using a feasible number of processors. But is an algorithm meeting the above desiderata likely to exist?

A most likely negative answer to the above question has been given by Balcázar, Gabarró and Santha, who showed in (Balcazar, Gabarro and Santha, 1992) that deciding strong bisimilarity between finite labelled transition systems is P-complete—this means that it is one of the 'hardest problems' in the class P of problems solvable in deterministic polynomial time. P-complete problems are of interest because they appear to lack highly parallel solutions. So showing that an algorithmic problem is P-complete is interpreted as strong evidence that the existence of an efficient parallel solution for it is unlikely. See, for instance, the book (Greenlaw, Hoover and Ruzzo, 1995) for much more information on P-completeness.

We shall now present the main ideas behind the hardness proof of the above mentioned result, which we reiterate below in the form of a theorem.

Theorem 0.3.2 (Balcázar, Gabarró and Santha 1992) The bisimilarity checking problem between two states in a finite labelled transition system is P-complete.

The theorem can be proved by offering a reduction from a problem that is

already known to be P-complete to the problem of checking strong bisimilarity between a pair of finite-state processes P and Q. The reduction will be a so-called log-space reduction, which means that it is computable in deterministic logarithmic space.

Similarly as in the original proof by Balcázar, Gabarró and Santha, we will provide a reduction from the *monotone Circuit Value Problem* (mCVP), which is known to be P-complete (see, e.g., (Papadimitriou, 1994)). However, the ideas in the reduction presented below are different from their original proof. For the purpose of this presentation we shall use the so-called *Defender's Forcing Technique*, which simplifies many of the constructions for hardness results. For a more detailed introduction to this technique consult (Jančar and Srba, 2008).

A *monotone Boolean circuit* C is a finite directed acyclic graph in which the nodes are either of indegree zero (*input* nodes) or of indegree two and there is exactly one node of outdegree zero (the *output* node).

Each node in C that is not an input node is labelled with one of the symbols $\wedge$ and $\vee$, which stand for conjunction and disjunction, respectively.

An *input for the circuit* C is an assignment of the truth values 0 (false) or 1 (true) to all input nodes. Given an input, every node in the circuit can be uniquely assigned a truth value as follows:

- the value of an input node is given by the input assignment,
- the value of a node labelled with $\wedge$ or $\vee$ is the logical conjunction or disjunction of values of its two predecessors, respectively.

Given an input for the circuit, the *output value of the circuit* C is the value assigned to its output node.

The mCVP problem asks to decide, given a monotone Boolean circuit C and its input, whether its output value is true.

An example of a monotone Boolean circuit with an input assignment and computed values at each node is given in Figure 0.7.

Assume now a given monotone Boolean circuit C with the output node o and with a given input. We shall construct a finite labelled transition system and two of its processes P_o and Q_o such that if the output value of C is true then P_o and Q_o are strongly bisimilar, and if the output value is false then Q_o does not strongly simulate P_o. Such a construction will enable us to state a general claim about P-hardness for *all* relations between the simulation preorder and bisimilarity.

Remark 0.3.3 Indeed, we will be able to draw such a general conclusion for *any* (perhaps not even yet 'discovered') relation on processes R such that if

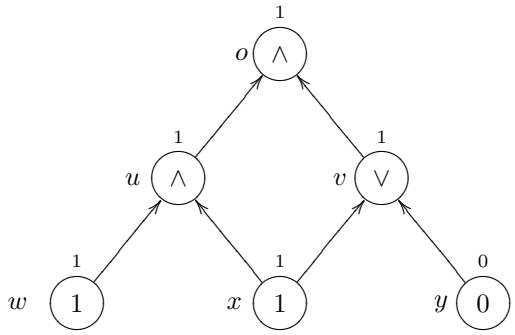


Fig. 0.7. An instance of mCVP together with assigned truth values

$P \sim Q$ then $(P, Q) \in R$, and if Q does not simulate P then $(P, Q) \notin R$, because if the output value of C is true then $P \sim Q$ and hence $(P, Q) \in R$, and if the output value of C is false then Q does not simulate P and hence $(P, Q) \notin R$. Examples of such a relation R include, among others, simulation preorder/equivalence, completed simulation preorder/equivalence, ready simulation preorder/equivalence, 2-nested simulation preorder/equivalence and, of course, also bisimulation equivalence.

The processes of the constructed labelled transition system are

- P_{end},
- P_v and Q_v for every node v in C, and additionally
- P'_v, Q^ℓ_v and Q^r_v for every node v in C labelled with $\vee$.

The transition relation contains the following transitions:

- $P_v \xrightarrow{\ell} P_{v_1}$, $P_v \xrightarrow{r} P_{v_2}$, and $Q_v \xrightarrow{\ell} Q_{v_1}$, $Q_v \xrightarrow{r} Q_{v_2}$ for every node v in C labelled with $\wedge$ and with the predecessor nodes v_1 and v_2, and
- $P_v \xrightarrow{a} P'_v$, $P_v \xrightarrow{a} Q^\ell_v$, $P_v \xrightarrow{a} Q^r_v$, and
  $P'_v \xrightarrow{\ell} P_{v_1}$, $P'_v \xrightarrow{r} P_{v_2}$, and
  $Q_v \xrightarrow{a} Q^\ell_v$, $Q_v \xrightarrow{a} Q^r_v$, and
  $Q^\ell_v \xrightarrow{\ell} Q_{v_1}$, $Q^\ell_v \xrightarrow{r} P_{v_2}$, $Q^r_v \xrightarrow{r} Q_{v_2}$, $Q^r_v \xrightarrow{\ell} P_{v_1}$
  for every node v in C labelled with $\vee$ and with the predecessor nodes v_1 and v_2, and
- $P_v \xrightarrow{0} P_{end}$ for every input node v in C assigned the value false.

The reduction is clearly computable in log-space. An example of the construction, taking as the input circuit the one in Figure 0.7, is presented in Figure 0.8.

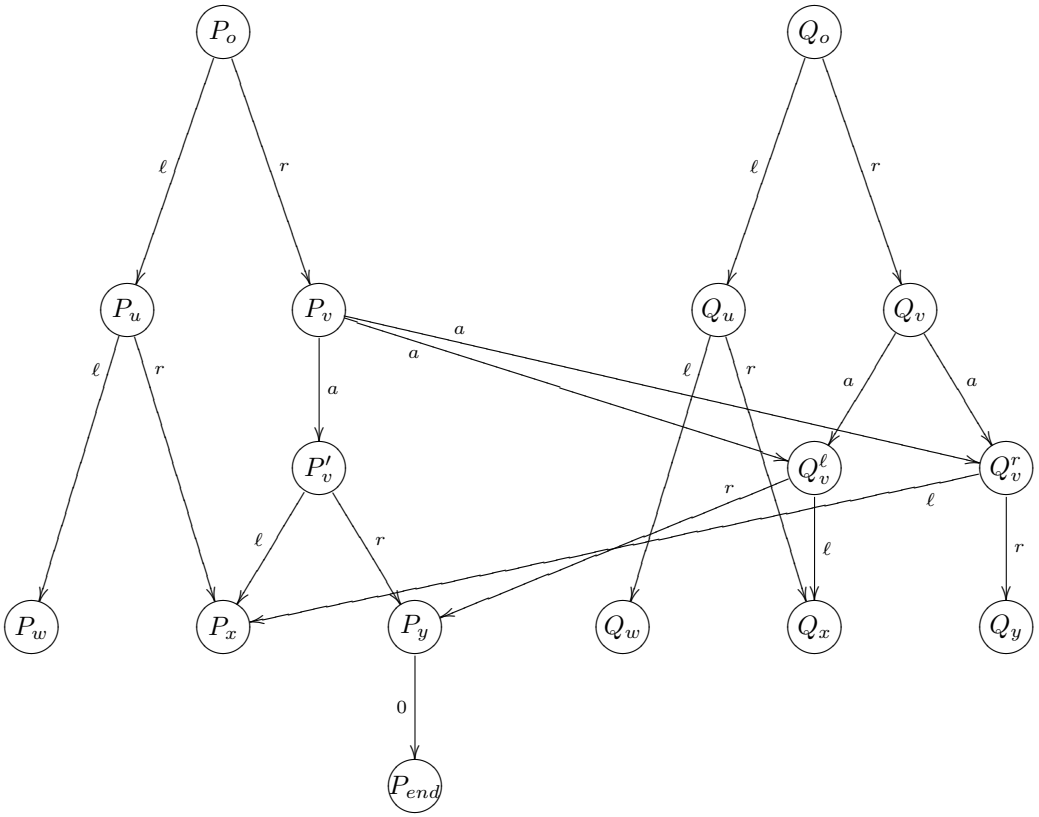


Fig. 0.8. Constructed processes P_o and Q_o according to the reduction

Exercise 0.3.4 Consider the transition system constructed in Figure 0.8. Argue that Defender has a winning strategy in the bisimulation game starting from (P_o, Q_o); in other words show that $P_o \sim Q_o$. □

We will now show that

- if the output value of C is true then P_o and Q_o are strongly bisimilar, and
- if the output value of C is false then Q_o does not strongly simulate P_o.

As a conclusion, we will get that deciding *any* relation between the simulation preorder and strong bisimilarity on finite labelled transition systems is P-hard.

Let us consider a play in the bisimulation game between Attacker and Defender starting from the pair (P_o, Q_o). The idea is that, after finitely many rounds of a play, the players will eventually reach some input node. During each play, Attacker is trying to reach an input node P_v with assigned value false, as this is a winning configuration for Attacker. (Defender has no answer

to the action 0 that Attacker can perform in the left-hand-side process.) On the other hand, Defender is trying to reach an input node with assigned value true, as Attacker loses in such a situation (no further actions are available). The processes P_o and Q_o are constructed in such a way that in the current pair (P_v, Q_v), where v is labelled by $\wedge$, it is Attacker who chooses the next node and forces reaching the next pair (P_{v_i}, Q_{v_i}) for a chosen $i \in \{1, 2\}$ by playing either the action ℓ or r. If the current pair (P_v, Q_v) corresponds to a node v which is labelled by $\vee$, it should be Defender who decides the next pair of processes. This is achieved by using a particular instance of the so-called Defender's Forcing Technique (Jančar and Srba, 2008). Initially, Attacker has five possible moves under the action a from the pair (P_v, Q_v) where v is labelled by $\vee$. Clearly, the only possible attack is by playing $P_v \xrightarrow{a} P'_v$; all the remaining four moves can be matched by Defender in such a way that the play continues from a pair of identical processes (a clear winning position for Defender). Defender can now answer by playing either $Q_v \xrightarrow{a} Q_v^\ell$ or $Q_v \xrightarrow{a} Q_v^r$ and the players continue from the pair (P'_v, Q_v^ℓ) or (P'_v, Q_v^r), depending of Defender's choice. In the first case Attacker is forced to play the action ℓ and the players reach the pair (P_{v_1}, Q_{v_1}) (playing action r leads to an immediate loss for Attacker as Defender can again reach a pair of equal processes). Similarly, in the second case Attacker must play the action r and Defender forces the pair (P_{v_2}, Q_{v_2}).

To sum up, starting from the pair (P_o, Q_o), where o is the output node of the circuit, the players choose, step by step, one particular path in the circuit leading to an input node. From the current pair (P_v, Q_v), where v is labelled by $\wedge$, it is Attacker who chooses the next pair; if the node v is labelled by $\vee$ it is Defender who chooses (in two rounds) the next pair. Attacker wins if, and only if, the players reach a pair of processes that represent an input node with the assigned value false. Hence if the output value of C is true then Defender has a winning strategy in the strong bisimulation game. Note that if, on the other hand, Attacker has a winning strategy then it can be achieved by attacking only from the left-hand-side process (in this case, when the output value of C is false, there must be an input node set to false, because the circuit is monotone). Hence if the output value of C is false then Attacker has a winning strategy in the strong simulation game.

We have so argued for the following P-hardness theorem.

Theorem 0.3.5 The equivalence checking problem between two processes in a finite-state, acyclic labelled transition system is P-hard for any relation between strong simulation preorder and strong bisimilarity.

Exercise 0.3.6 Assume that the output of some given circuit C is true. Using

the ideas of the proof presented above, generalize your solution to Exercise 0.3.4 in order to construct a bisimulation containing the pair (P_o, Q_o).

In fact, Sawa and Jančar proved in a recent paper (Sawa and Jančar, 2001) that the problem is P-hard for *all* relations between the trace preorder and bisimilarity. Their result is even more general but the encoding is more involved.

Finally, note that in the reduction presented above we used four different actions. So the question is whether equivalence checking is P-hard even in case of a singleton action alphabet. Unfortunately, the answer is positive also in this case. For bisimilarity this result is a consequence of the original reduction provided by Balcázar, Gabarró and Santha (Balcazar et al., 1992), which uses only a single action for the encoding of any alternating mCVP instance. In order to show that the problem is P-hard also for other behavioural equivalences and preorders, we shall now present a simple construction based on (Srba, 2001), which provides a general log-space reduction of bisimilarity checking on labelled transition systems with several actions to bisimilarity checking over a one-letter action set.

Assume two given processes P and Q over an LTS T with the set of actions $\{a_1, a_2, \ldots, a_\ell\}$. We shall construct a modified LTS T' which contains all the processes of T together with some additional ones defined in the following way: for every transition $P_1 \xrightarrow{a_i} P_2$ in T we add into T'

- two transitions $P_1 \rightarrow P_{(P_1,a_i,P_2)}$ and $P_{(P_1,a_i,P_2)} \rightarrow P_2$ where $P_{(P_1,a_i,P_2)}$ is a newly added state, and
- a newly added path of length i from $P_{(P_1,a_i,P_2)}$.

Finally, for every process P in T we create in T' a newly added path of length $\ell+1$ starting from P. As the LTS T' contains only one action, we omit its name from the transition relation. We shall call the newly added processes into T' *intermediate processes*. An example of the reduction is depicted in Figure 0.9.

Exercise 0.3.7 Consider the one-action LTS in Figure 0.9. Argue that Attacker has a winning strategy in the bisimulation game starting from the pair (P_1, P') where $P' = P_{(P_1,a_2,P_3)}$ and from the pair (P_1, P_2).

The reduction presented above can clearly be carried out in log-space. Moreover, we can observe the following property of the LTS T'.

Lemma 0.3.8 Let P be a process of T and P' a newly added intermediate process in T'. Then $P \not\sim P'$ in the LTS T'.

Proof We describe Attacker's winning strategy from the pair (P, P'). In the

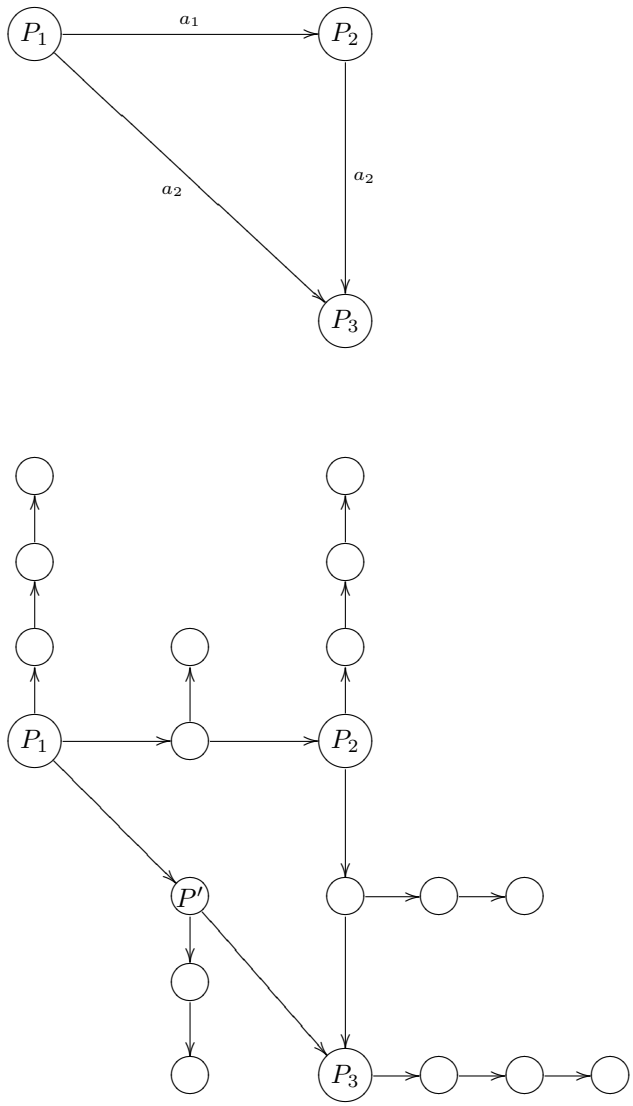


Fig. 0.9. Input LTS above and the one-action LTS below where $P' = P_{(P_1,a_2,P_3)}$

first round Attacker plays the transition from P that starts the path of length $\ell+1$ that was added to the LTS T'. Defender answers by a move from P'. Now if Defender ended again in some intermediate process, it must be in a branch starting from some $P_{(P_1,a_i,P_2)}$, but this configuration is winning for Attacker

by not changing sides and by simply making the remaining ℓ moves along the path he chose initially. Defender cannot do as many moves and loses. If, on the other hand, Defender ended in some of the original processes in T, Attacker will change sides and play the sequence of $\ell + 1$ transitions available from this original processes. Again, Defender cannot match this sequence in the other process as the sequence of moves he has at his disposal is shorter by one. Hence Defender loses also in this case. □

Due to Lemma 0.3.8, we know that during any bisimulation game Defender is forced to play in T' in such a way that every current configuration consists either of two intermediate processes or of two processes from the original LTS T.

We can now observe that for any two processes P and Q in T we have $P \sim Q$ in T if, and only if, $P \sim Q$ in T'. Indeed, each move $P_1 \stackrel{a_i}{\longrightarrow} P_2$ by the Attacker in the original LTS T is simulated by two moves $P_1 \rightarrow P_{(P_1,a_i,P_2)} \rightarrow P_2$ in the LTS T' and Defender can only answer by some corresponding two moves such that in the intermediate state the length of the available path leading to a deadlock is the same as on the Attacker's side. Hence Defender has to use the sequence of two transitions that corresponds to the same action as played by Attacker. (Should Defender choose a different move, Attacker will easily win the game.) Similarly, if Defender has a winning strategy in the original LTS T, it can be easily extended to a winning strategy in T'.

Because the proposed reduction preserves acyclicity, we obtain the following theorem.

Theorem 0.3.9 Strong bisimilarity checking over finite LTSs is log-space reducible to strong bisimilarity checking over finite LTSs over a one-letter action alphabet. Moreover, the reduction preserves acyclicity.

By combining Theorem 0.3.5 and Theorem 0.3.9 we obtain the promised P-hardness result for finite labelled transition systems over a singleton action set.

Corollary 0.3.10 The equivalence-checking problem between two processes in a finite, acyclic labelled transition system over a singleton action set is P-hard for any relation between the strong simulation preorder and strong bisimilarity.

Remark 0.3.11 Note that the above presented reduction from LTS to a one-letter action alphabet LTS works even for infinite-state systems. The question is, though, what classes of infinite-state systems are closed under this construction and whether it is efficiently implementable. Indeed, as argued in (Srba, 2001), this is true for may well-known classes of systems generated by e.g. pushdown

automata or Petri nets and the transformation is efficient and moreover preserves answers to μ-calculus model checking problems.

Exercise 0.3.12 Find a (small) modification of the reduction in the proof of Theorem 0.3.9 (see also an example in Figure 0.9), such that one can reach the following conclusion regarding the size of the constructed system over a one-letter action alphabet: if the original LTS has n states, m transitions and m' actions, then the constructed LTS has $O(n + m')$ states and $O(m)$ transitions. Since $m' \leq m$, this will imply that the constructed system has $O(n + m)$ states and $O(m)$ transitions.

It is well known that weak bisimilarity, observational congruence (Milner, 1989), branching bisimilarity and rooted branching bisimilarity (Glabbeek and Weijland, 1996) coincide with strong bisimilarity over τ-free labelled transition systems. Since the actions used in constructing the labelled transition system from a monotone Boolean circuit as in the proof of Theorem 0.3.5 can be chosen to be different from τ, the proof of Theorem 0.3.5 also yields a log-space reduction of the monotone circuit value problem to that of deciding weak bisimilarity, observational congruence, branching bisimilarity and rooted branching bisimilarity over a finite, acyclic labelled transition system. We therefore have the following result.

Theorem 0.3.13 The problem of deciding weak bisimilarity, observational congruence, branching bisimilarity and rooted branching bisimilarity between two states in a finite, acyclic labelled transition system is P-hard. This holds true even when the set of actions is a singleton.

Readers who are familiar with Chapter 1 in the book will recall that, over image-finite labelled transition systems (and hence over finite labelled transition systems), strong bisimilarity coincides with the relation

$$\sim_\omega = \cap_{i \geq 0} \sim_i \ ,$$

where $\sim_i$, $i \geq 0$, is the ith approximant to bisimilarity. (See Section 3.8.2 in Chapter 1.) In their classic paper (Hennessy and Milner, 1985), Hennessy and Milner defined an alternative non-increasing (with respect to set inclusion) sequence $\simeq_i$, $i \geq 0$, of approximants to bisimilarity as follows:

- $P \simeq_0 Q$ always holds.
- $P \simeq_{i+1} Q$ iff for each sequence of actions w:
  (1) for each P' such that $P \stackrel{w}{\rightarrow} P'$, there is some Q' such that $Q \stackrel{w}{\rightarrow} Q'$ and $P' \simeq_i Q'$;

(2) for each Q' such that $Q \stackrel{w}{\rightarrow} Q'$, there is some P' such that $P \stackrel{w}{\rightarrow} P'$ and $P' \simeq_i Q'$.

The relations $\simeq_i$, $i \geq 0$, are the so-called *nested trace semantics* (Aceto, Fokkink, Glabbeek and Ingolfsdottir, 2004). For instance, $\simeq_1$ is trace equivalence and $\simeq_2$ is possible-futures equivalence (Rounds and Brookes, 1981). Each of the $\simeq_i$, $i \geq 0$, is an equivalence relation and is preserved by the operators of CCS introduced in Chapter 1 of the book.

Hennessy and Milner showed in (Hennessy and Milner, 1985) that, over image-finite labelled transition systems, strong bisimilarity also coincides with the relation

$$\simeq_\omega = \cap_{i \geq 0} \simeq_i \ .$$

In stark contrast to the P-completeness of bisimilarity checking over finite labelled transition systems, Kanellakis and Smolka proved in (Kanellakis and Smolka, 1990) the following result to the effect that computing the relations $\simeq_i$, $i \geq 1$, is much harder than computing bisimilarity.

Theorem 0.3.14 (Kanellakis and Smolka) For each $i \geq 1$, the problem of deciding the equivalence $\simeq_i$ over finite labelled transition systems is PSPACE-complete.

Proof Recall that $\simeq_1$ is just trace equivalence. Deciding trace equivalence over finite labelled transition systems is known to be PSPACE-complete. (See Lemma 4.2 in (Kanellakis and Smolka, 1990), where this result is attributed to Chandra and Stockmeyer.) We are therefore left to prove the claim for $\simeq_i$, $i \geq 2$.

We begin by arguing that deciding the equivalence $\simeq_i$ is PSPACE-hard for each $i \geq 2$. To this end, we prove that $\simeq_1$ reduces (in polynomial time) to $\simeq_i$, for each $i \geq 2$. In order to show this claim, it suffices only to give a polynomial time reduction from $\simeq_i$ to $\simeq_{i+1}$, for each $i \geq 1$. In presenting the reduction from $\simeq_i$ to $\simeq_{i+1}$, we shall use the operators of action prefixing and choice from CCS that were introduced in Chapter 1 (Section 4.2) of this book. (Note that those operators can be viewed as operations over the class of finite labelled transition systems.)

The reduction works as follows. Let P and Q be two states in a finite labelled transition system. Define the processes P' and Q' thus:

$$\begin{aligned} P' &= a.\,(P + Q) \text{ and} \\ Q' &= a.\,P + a.\,Q \ , \end{aligned}$$

where a is an arbitrary action in our labelled transition system. We claim that

$$P \simeq_i Q \text{ iff } P' \simeq_{i+1} Q' \ .$$

This can be shown as follows.

- If $P' \simeq_{i+1} Q'$ then the definition of $\simeq_i$ yields that $P \simeq_i P + Q \simeq_i Q$, from which $P \simeq_i Q$ follows by the transitivity of $\simeq_i$.
- Assume now that $P \simeq_i Q$ holds. We prove that $P' \simeq_{i+1} Q'$.

  We limit ourselves to showing that for each sequence of actions w and process P'' such that $P' \stackrel{w}{\rightarrow} P''$, there is some Q'' such that $Q' \stackrel{w}{\rightarrow} Q''$ and $P'' \simeq_i Q''$. We proceed by considering three cases, depending on whether w is empty, $w = a$ or w is of length at least two.

  – Suppose that w is empty. Then $P' = P''$ and $Q' = Q''$. Recall that $\simeq_i$ is a congruence with respect to action prefixing and choice. Therefore, since $P \simeq_i Q$ and choice is easily seen to be idempotent with respect to $\simeq_i$,

  $$P' = a.\,(P + Q) \simeq_i a.\,P \simeq_i a.\,P + a.\,P \simeq_i a.\,P + a.\,Q \simeq_i Q' \ ,$$

  and we are done.
  – Suppose that $w = a$. Then $P'' = P + Q$ and reasoning as in the previous case we infer that, for instance, $P'' \simeq_i P$. Since $Q' \stackrel{a}{\rightarrow} P$, we are done.
  – Suppose that w is of length at least two. In this case, it is easy to see that $Q' \stackrel{w}{\rightarrow} P''$ and the claim follows since $\simeq_i$ is reflexive.

Therefore $\simeq_i$ reduces to $\simeq_{i+1}$, for each $i \geq 1$, as claimed. It follows that $\simeq_1$ reduces (in polynomial time) to $\simeq_i$, for each $i \geq 2$, by applying the reduction given above $(i-1)$ times.

Membership of each $\simeq_i$, $i \geq 2$, in PSPACE can be shown by reducing the problem of deciding $\simeq_i$ to the equivalence problem for nondeterministic finite automata (see, e.g., (Sipser, 2006)), which is a well known PSPACE-complete problem (Garey and Johnson, 1979; Stockmeyer and Meyer, 1973). To this end, observe that, since we already know that $\simeq_1$ is in PSPACE, we can reason inductively as follows. Assume that $\simeq_i$ is in PSPACE, for some $i \geq 1$. Using this assumption, we proceed to prove membership in PSPACE for $\simeq_{i+1}$. Let P and Q be two states in a finite labelled transition system. Let $\{B_1, \ldots, B_k\}$ be the partition of the set of states induced by the equivalence $\simeq_i$. (The partition can be computed using a polynomial amount of space since $\simeq_i$ is in PSPACE by our inductive assumption.)

Observe now that we can view our input finite labelled transition system as a nondeterministic finite automaton with either P or Q as start state and with any of the sets $B_1, \ldots, B_k$ as set of accept states. For each $\ell \in \{1, \ldots, k\}$, let

$L_\ell(P)$ (respectively, $L_\ell(Q)$) denote the language accepted by the nondeterministic finite automaton that results by taking P (respectively, Q) as start state and B_ℓ as set of accept states. It is easy to see that

$$P \simeq_{i+1} Q \text{ iff } L_\ell(P) = L_\ell(Q), \text{ for each } \ell \in \{1, \ldots, k\} \ .$$

This yields membership in PSPACE for $\simeq_{i+1}$, which was to be shown. □

In the light of the above theorem, bisimilarity over finite labelled transition systems can be decided in polynomial time, but is the 'limit' of a non-increasing sequence of equivalences that are all PSPACE-complete. We shall meet analogues of this result in Section 0.4.

Remark 0.3.15 In the setting of equational axiomatizations of behavioural relations, the paper (Aceto et al., 2004) presents hardness results for nested trace semantics that are akin to Theorem 0.3.14 and set them apart from bisimilarity. More precisely, in that reference, the authors show that, unlike bisimilarity, none of the relations $\simeq_i$, $i \geq 2$, affords a finite, ground-complete axiomatization over Basic CCS.

0.3.3 EXPTIME-completeness of equivalence checking on networks of finite processes

We have shown in Theorem 0.3.5 that the equivalence checking problem on finite, acyclic processes is P-hard for any relation between the simulation preorder and bisimilarity. In fact, the usually studied equivalences and preorders between the simulation preorder and bisimilarity, such as the completed, ready and 2-nested simulations preorders, are also decidable in deterministic polynomial time. (See Section 7 in Chapter 1 of this book or the encyclopaedic survey paper (Glabbeek, 2001) for the definition of those relations. An efficient algorithm for deciding the ready simulation preorder is described in (Bloom and Paige, 1995).)

The aforementioned results are rather encouraging and bode well for the use of such relations in computer-aided implementation verification. However, one should realize that the complexity of the algorithms we have presented so far is measured with respect to the size of the input labelled transition system. These systems are called *flat systems*. However, in practice the studied LTS is often given indirectly, for example as a parallel composition of communicating finite-state agents. The semantics of such a composition is still given in terms of labelled transition systems; however, the resulting LTS is often of exponential size with respect to the size of the description of the system. The formalisms that provide such a succinct description of large labelled transition systems

are called *non-flat systems*. In this section, we shall introduce one example of a non-flat system where a number of concurrent finite processes communicate via synchronization on common actions and we shall study the complexity of bisimulation-like equivalence checking on such non-flat systems. As we shall see, the resulting decision problems become computationally hard in such a setting.

Let $(Pr_i, Act_i, \rightarrow_i)$ be a finite LTS for every i, $1 \leq i \leq n$. A *network of finite processes* is a parallel composition $P_1 \mid P_2 \mid \cdots \mid P_n$ where $P_i \in Pr_i$ for each $i \in \{1, \ldots, n\}$. The semantics of such a composition is given in terms of a flat LTS $(Pr, Act, \rightarrow)$ where

- $Pr = Pr_1 \times Pr_2 \times \cdots \times Pr_n$,
- $Act = Act_1 \cup Act_2 \cup \cdots \cup Act_n$, and
- $(Q_1, Q_2, \ldots, Q_n) \xrightarrow{a} (Q'_1, Q'_2, \ldots, Q'_n)$ whenever $Q_i \xrightarrow{a}_i Q'_i$ if $a \in Act_i$, and $Q_i = Q'_i$ if $a \notin Act_i$.

The flattening of the network $P_1 \mid P_2 \mid \cdots \mid P_n$ is then the process $(P_1, P_2, \ldots, P_n)$ in the above defined LTS and the equivalence checking problem on networks of finite processes is defined simply by using the standard notion on the underlying flattened systems.

It was shown by Rabinovich (Rabinovich, 1997) that equivalence checking for networks of finite processes with hiding of actions is PSPACE-hard for any relation between trace equivalence and bisimilarity. Rabinovich conjectured that the problem is EXPTIME-hard. Indeed, EXPTIME-hardness was already known for bisimilarity (Jategaonkar and Meyer, 1996) and simulation equivalence (Harel, Kupferman and Vardi, 1997) and the conjecture was later partially confirmed by Laroussinie and Schnoebelen (Laroussinie and Schnoebelen, 2000), who proved an EXPTIME-hardness result for all the relations between the simulation preorder and bisimilarity on networks of finite processes. In their proof, Laroussinie and Schnoebelen do not use any hiding of actions. Finally, Sawa (Sawa, 2003) settled Rabinovich's conjecture by showing that equivalence checking on networks of finite processes with hiding is EXPTIME-hard for all relations between the trace preorder and bisimilarity.

We shall now argue that many equivalence checking problems with respect to bisimulation-like equivalences/preorders are EXPTIME-complete on networks of finite processes (without hiding). Our exposition of the hardness result is based on the ideas from (Laroussinie and Schnoebelen, 2000), though the construction is different in the technical details and the presentation is simplified by using the previously discussed Defender's forcing technique (Jančar and Srba, 2008).

The EXPTIME-hardness proof is done by offering a polynomial-time reduction from the acceptance problem for alternating linear bounded automata (ALBA), a well known EXPTIME-complete problem—see, e.g., (Sipser, 2006).

An ALBA is a tuple $M = (Q, Q_\exists, Q_\forall, \Sigma, \Gamma, \blacktriangleright, \blacktriangleleft, q_0, q_{acc}, q_{rej}, \delta)$ where

- $Q_\exists$ and $Q_\forall$ are finite, disjoint sets of existential, respectively universal, control states and $Q = Q_\exists \cup Q_\forall$,
- Σ is a finite input alphabet,
- $\Gamma \supseteq \Sigma$ is a finite tape alphabet,
- $\blacktriangleright, \blacktriangleleft \in \Gamma$ are the left and right end-markers,
- $q_0, q_{acc}, q_{rej} \in Q$ are the initial, accept and reject states, respectively, and
- $\delta\colon Q \setminus \{q_{acc}, q_{rej}\} \times \Gamma \to Q \times Q \times \Gamma \times \{-1, +1\}$ is a computation step function, such that whenever $\delta(p, x) = (q_1, q_2, y, d)$ then $x = \blacktriangleright$ iff $y = \blacktriangleright$ and if $x = \blacktriangleright$ then $d = +1$, and $x = \blacktriangleleft$ iff $y = \blacktriangleleft$ and if $x = \blacktriangleleft$ then $d = -1$.

We can, without loss of generality, assume that the input alphabet is binary, that is $\Sigma = \{a, b\}$ and that the tape alphabet only contains the symbols from the input alphabet and the end-markers, that is $\Gamma = \{a, b, \blacktriangleleft, \blacktriangleright\}$. We shall write $p \xrightarrow{x \to y, d} q_1, q_2$ whenever $\delta(p, x) = (q_1, q_2, y, d)$. The intuition is that if the machine M is in a state p and is reading the tape symbol x, it can replace x with y, move the tape head one position to the right (if $d = +1$) or to the left (if $d = -1$) and enter the control state q_1 or q_2 (depending on the type of the control state p the choice between q_1 and q_2 is either existential or universal).

A configuration of M is given by the current control state, the position of the tape head and the content of the tape; we write it as a triple from $Q \times \mathbb{N} \times (\{\blacktriangleright\}.\Gamma^*.\{\blacktriangleleft\})$. The head position on the left end-marker $\blacktriangleright$ is by definition 0. A *step of computation* is a relation between configurations, denoted by $\vdash$, that is defined using the function δ in the usual way. For example, for the transition $p \xrightarrow{a \to b, +1} q_1, q_2$ we get $(p, 1, \blacktriangleright aab\blacktriangleleft) \vdash (q_1, 2, \blacktriangleright bab\blacktriangleleft)$ and $(p, 1, \blacktriangleright aab\blacktriangleleft) \vdash (q_2, 2, \blacktriangleright bab\blacktriangleleft)$.

Given a word $w \in \Sigma^*$, the initial configuration of M is $(q_0, 1, \blacktriangleright w\blacktriangleleft)$. A configuration is called *accepting* if it is of the form $(q_{acc}, i, \blacktriangleright w'\blacktriangleleft)$, and it is called *rejecting* if it is of the form $(q_{rej}, i, \blacktriangleright w'\blacktriangleleft)$.

A *computation tree* on an input $w \in \Sigma^*$ is a labelled tree such that the root is labelled by the initial configuration, the leaves are the nodes labelled by accepting or rejecting configurations and every non-leaf node labelled by a configuration c with existential control state has exactly one child labelled with some c' such that $c \vdash c'$, and every non-leaf node labelled by a configuration c with universal control state has exactly two children labelled with c_1 and c_2 such that c_1 and c_2 are the two successor configurations of c, i.e., $c \vdash c_1$ and $c \vdash c_2$. Without loss of generality, we shall assume from now on that any computation tree is finite (see e.g. (Sipser, 2006, page 198)).

A computation tree on an input w is called *accepting* if its leaves are labelled

only with accepting configurations. An ALBA M accepts a word $w \in \Sigma^*$, if there exists an accepting computation tree of M on w.

As already mentioned, the problem of deciding whether a given ALBA M accepts a given word $w \in \Sigma^*$ is EXPTIME-complete (see e.g. (Sipser, 2006)).

Let M be a given ALBA with an input $w = w_1 w_2 \dots w_n \in \{a, b\}^*$. We shall construct (in polynomial time) two networks of finite processes

$$P = C(q_0, 1) \mid T_0^{\blacktriangleright} \mid T_1^{w_1} \mid T_2^{w_2} \mid \cdots \mid T_n^{w_n} \mid T_{n+1}^{\blacktriangleleft}$$

and

$$Q = D(q_0, 1) \mid T_0^{\blacktriangleright} \mid T_1^{w_1} \mid T_2^{w_2} \mid \cdots \mid T_n^{w_n} \mid T_{n+1}^{\blacktriangleleft}$$

such that

- if M accepts w then Defender has a winning strategy in the bisimulation game from (P, Q), and
- if M does not accept w then Attacker has a winning strategy in the simulation game from (P, Q).

Having shown these claims, we will be able to conclude that equivalence checking for *any* relation between the simulation preorder and bisimilarity on networks of finite processes is EXPTIME-hard. (This kind of general reasoning was explained in Remark 0.3.3.)

The intuition behind the construction is that in the processes

$$T_0^{\blacktriangleright}, T_1^{w_1}, T_2^{w_2}, \dots, T_n^{w_n}, T_{n+1}^{\blacktriangleleft}$$

we remember the actual content of the tape. Because the left and right endmarkers cannot be rewritten, it is sufficient that the processes $T_0^{\blacktriangleright}$ and $T_{n+1}^{\blacktriangleleft}$ are one-state processes and because we assume that the rest of the tape consists only of the symbols a and b, the remaining processes $T_1^{w_1}, \dots, T_n^{w_n}$ will all contain exactly two states, remembering the current content of the particular tape cell. Additionally, the process P has the parallel component $C(p, i)$, which represents the current control state and the position of the tape head; similarly the same information is remembered in the process Q in the parallel component $D(p, i)$. In fact, the use of C or D is the only point where the processes P and Q differ. The processes representing the tape cells will communicate with the process $C(p, i)$ or $D(p, i)$ using actions of the form (x, y, i), meaning that if the tape cell i contains the symbol x, it is going to remember the symbol y after the (hand-shake) communication takes place. Defining the processes $C(p, i)$ and $D(p, i)$ is rather straightforward. The only challenge is the implementation of the choice between the new control states q_1 and q_2. If p is a universal state then it will be Attacker who chooses one of them—implementation of this choice by the Attacker is easy.

If p is an existential state, it will be Defender who chooses the next control state—this part of the construction will be implemented using the Defender's forcing technique, which was already used in the proof of Theorem 0.3.5 showing the P-hardness of the equivalence checking problem on flat processes. Finally, in the process $C(q_{rej}, i)$ Attacker will be able to perform a special action rej for which Defender will have no answer in the process $D(q_{rej}, i)$.

Formally, the reduction is computed as follows.

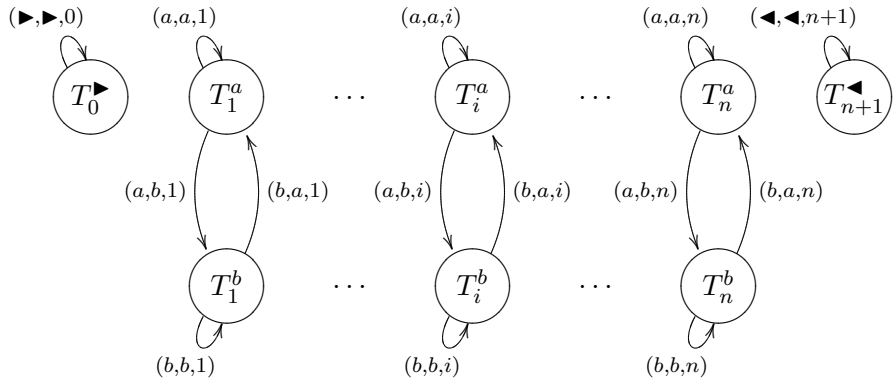
(1) Create a process $T_0^{\blacktriangleright}$ with the only transition $T_0^{\blacktriangleright} \xrightarrow{(\blacktriangleright,\blacktriangleright,0)} T_0^{\blacktriangleright}$.

(2) Create a process $T_{n+1}^{\blacktriangleleft}$ with the only transition $T_{n+1}^{\blacktriangleleft} \xrightarrow{(\blacktriangleleft,\blacktriangleleft,n+1)} T_{n+1}^{\blacktriangleleft}$.

(3) For every i, $1 \leq i \leq n$, create two processes T_i^a and T_i^b with the transitions

- $T_i^a \xrightarrow{(a,b,i)} T_i^b$,
- $T_i^a \xrightarrow{(a,a,i)} T_i^a$,
- $T_i^b \xrightarrow{(b,a,i)} T_i^a$, and
- $T_i^b \xrightarrow{(b,b,i)} T_i^b$.

The following picture shows the processes constructed in parts (1)–(3).

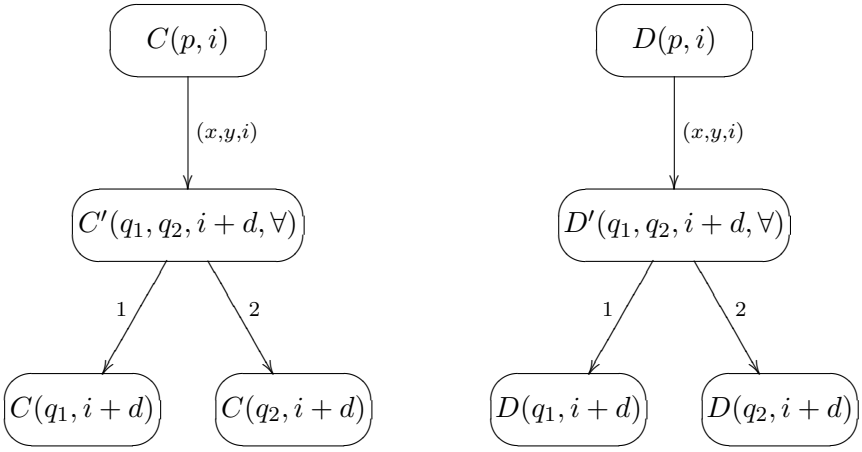


(4) For every i, $0 \leq i \leq n+1$, and every transition $p \xrightarrow{x \to y, d} q_1, q_2$ in M such that $p \in Q_\forall$ create the processes $C(p,i)$, $C'(q_1, q_2, i+d, \forall)$, $C(q_1, i+d)$, $C(q_2, i+d)$ and $D(p,i)$, $D'(q_1, q_2, i+d, \forall)$, $D(q_1, i+d)$, $D(q_2, i+d)$ together with the transitions

- $C(p,i) \xrightarrow{(x,y,i)} C'(q_1, q_2, i+d, \forall)$,
- $C'(q_1, q_2, i+d, \forall) \xrightarrow{1} C(q_1, i+d)$,
- $C'(q_1, q_2, i+d, \forall) \xrightarrow{2} C(q_2, i+d)$, and
- $D(p,i) \xrightarrow{(x,y,i)} D'(q_1, q_2, i+d, \forall)$,
- $D'(q_1, q_2, i+d, \forall) \xrightarrow{1} D(q_1, i+d)$,

- $D'(q_1, q_2, i+d, \forall) \xrightarrow{2} D(q_2, i+d)$.

The two fragments of the leading processes in P and Q, pictured below for the rule $p \xrightarrow{x \to y, d} q_1, q_2$ with $p \in Q_\forall$, demonstrate the construction.



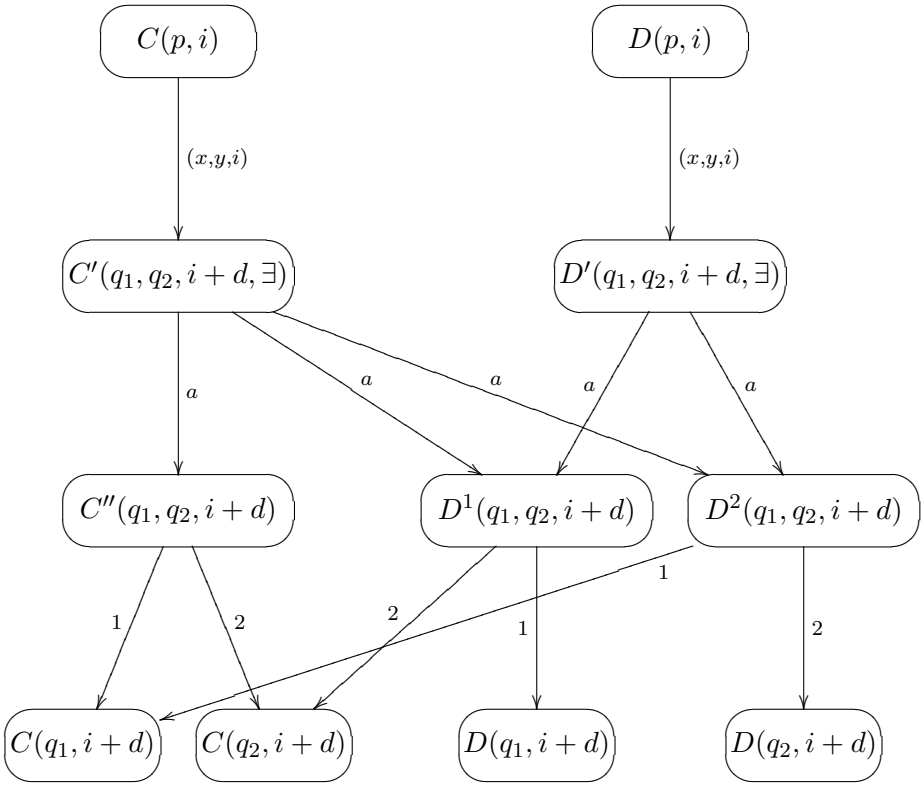
Consider a play in the bisimulation game starting from the pair of processes $C(p,i)|T$ and $D(p,i)|T$, where $T = T_0^{\blacktriangleright}|\cdots|T_i^x|\cdots|T_{n+1}^{\blacktriangleleft}$ is a parallel composition of the processes representing the tape content. Assume that there is a rule $p \xrightarrow{x \to y, d} q_1, q_2$. The only possible transition, a communication between $C(p,i)$, respectively $D(p,i)$, and T_i^x under the action (x,y,i), is uniquely determined in both of the processes $C(p,i) \mid T$ and $D(p,i) \mid T$. Hence, after one round of the bisimulation game, the players necessarily reach a pair of processes of the form $C'(q_1, q_2, i+d, \forall) \mid T'$ and $D'(q_1, q_2, i+d, \forall) \mid T'$ for some modified tape content T'. Now Attacker can play either the action 1 or 2 (without any communication) in order to select the next control state q_1 or q_2 of the ALBA M.

(5) For every i, $0 \leq i \leq n+1$, and every transition $p \xrightarrow{x \to y, d} q_1, q_2$ in M such that $p \in Q_\exists$ create the processes $C(p,i)$, $C'(q_1, q_2, i+d, \exists)$, $C''(q_1, q_2, i+d)$, $C(q_1, i+d)$, $C(q_2, i+d)$ and $D(p,i)$, $D'(q_1, q_2, i+d, \exists)$, $D^1(q_1, q_2, i+d)$, $D^2(q_1, q_2, i+d)$, $D(q_1, i+d)$, $D(q_2, i+d)$ together with the transitions

- $C(p,i) \xrightarrow{(x,y,i)} C'(q_1, q_2, i+d, \exists)$,
- $C'(q_1, q_2, i+d, \exists) \xrightarrow{a} C''(q_1, q_2, i+d)$,
- $C'(q_1, q_2, i+d, \exists) \xrightarrow{a} D^1(q_1, q_2, i+d)$,
- $C'(q_1, q_2, i+d, \exists) \xrightarrow{a} D^2(q_1, q_2, i+d)$,
- $C''(q_1, q_2, i+d) \xrightarrow{1} C(q_1, i+d)$,
- $C''(q_1, q_2, i+d) \xrightarrow{2} C(q_2, i+d)$, and
- $D(p,i) \xrightarrow{(x,y,i)} D'(q_1, q_2, i+d, \exists)$,

- $D'(q_1, q_2, i+d, \exists) \xrightarrow{a} D^1(q_1, q_2, i+d)$,
- $D'(q_1, q_2, i+d, \exists) \xrightarrow{a} D^2(q_1, q_2, i+d)$,
- $D^1(q_1, q_2, i+d) \xrightarrow{1} D(q_1, i+d)$,
- $D^1(q_1, q_2, i+d) \xrightarrow{2} C(q_2, i+d)$,
- $D^2(q_1, q_2, i+d) \xrightarrow{2} D(q_2, i+d)$,
- $D^2(q_1, q_2, i+d) \xrightarrow{1} C(q_1, i+d)$.

The added transitions for the rule $p \xrightarrow{x \to y, d} q_1, q_2$ where $p \in Q_\exists$ are depicted below.



As before, after one round of any play in the bisimulation game starting from $C(p,i) \mid T$ and $D(p,i) \mid T$ (where T is the parallel composition of processes representing the tape content) the players necessarily end in a pair consisting of processes of the form $C'(q_1, q_2, i+d, \exists) \mid T'$ and $D'(q_1, q_2, i+d, \exists) \mid T'$. Attacker is now forced to play the action a (no communication) in the left-hand-side process and enter the state $C''(q_1, q_2, i+d) \mid T'$—any other attack gives Defender the possibility to enter the same state chosen by Attacker and leads therefore to a losing line of play for Attacker. Defender answers either by entering the state

$D^1(q_1, q_2, i+d) \mid T'$ or $D^2(q_1, q_2, i+d) \mid T'$. From the pair consisting of the processes $C''(q_1, q_2, i+d) \mid T'$ and $D^1(q_1, q_2, i+d) \mid T'$, Attacker clearly has to play the action 1 (no communication) in order to avoid reaching a pair of equal processes after Defender's answer to his move. Similarly, from the pair consisting of the processes $C''(q_1, q_2, i+d) \mid T'$ and $D^2(q_1, q_2, i+d) \mid T'$, Attacker has to play the action 2. That means that the play will continue either from the pair consisting of $C(q_1, i+d) \mid T'$ and $D(q_1, i+d) \mid T'$, or from the one consisting of $C(q_2, i+d) \mid T'$ and $D(q_2, i+d) \mid T'$. Note that it was Defender who selected the next control state of the ALBA in this case.

(6) Finally, for every i, $0 \leq i \leq n+1$, add $C(q_{rej}, i) \xrightarrow{rej} C(q_{rej}, i)$.

This ensures that if during some play the players reach a rejecting configuration, Attacker wins by performing the action rej in the left-hand-side process to which Defender has no answer in the right-hand-side process.

To sum up, the players can force each other to faithfully simulate a computation of a given ALBA M on a word w. Attacker is selecting the successor configuration if the present control state is universal, Defender does the selection if the present control state is existential. Recall that we assumed that the computation tree of M on w does not have any infinite branches. We can therefore conclude that if M accepts w then Defender can force any play to reach a pair of processes representing an accepting leaf in the computation tree when it is Attacker's turn to play next. This is clearly a winning position for Defender as no continuation of the play is possible in either of the processes. On the other hand, if M does not accept w, Attacker can force any play to reach a pair of processes representing a rejecting leaf in the computation tree when it is his turn to play the action rej, and Defender loses. In this case, Attacker is moreover able to implement the winning strategy by playing exclusively in the left-hand-side process. Therefore we can state the following generic hardness result.

Theorem 0.3.16 The equivalence checking problem between two networks of finite processes is EXPTIME-hard for any relation between the strong simulation preorder and strong bisimilarity.

Notice now that because essentially all bisimulation-like equivalences and preorders studied in the literature are decidable in polynomial time on flat systems, we get an immediate EXPTIME-algorithm for these equivalence checking problems on non-flat systems, simply by flattening the networks of finite processes

(causing an exponential increase in size) and running the polynomial time algorithms on the flattened system. Hence we obtain the following corollary.

Corollary 0.3.17 The problems of checking the simulation preorder and equivalence, completed simulation preorder and equivalence, ready simulation preorder and equivalence, 2-nested simulation preorder and equivalence, and bisimilarity on networks of finite processes are EXPTIME-complete.

0.4 Decidability results for bisimilarity over infinite-state systems

We shall now consider the questions of decidability/undecidability of bisimilarity on so-called *infinite-state systems*, i.e. labelled transition systems with infinitely many reachable states. For example, these systems naturally arise when one models *infinite data domains*, such as counters, integer variables, stacks and queues, or *unbounded control structures*, like recursive procedure calls, dynamic process creation and mobility, and in *parameterized reasoning* involving an arbitrary number of identical processes.

Of course, an infinite-state system cannot be passed as an input to an algorithm as it is. We need to find a suitable *finite representation* of such a system in order to ask decidability questions. For the purpose of this chapter, we shall consider the concept of process rewrite systems as it provides a finite description of many well-studied formalisms like pushdown automata or Petri nets. After introducing process rewrite systems, we shall focus on two selected results. First, we shall argue that bisimilarity is decidable over the class of Basic Parallel Processes (BPP), which form the so-called communication-free subclass of Petri nets. The result will be proved using the *tableau technique*, which is applicable also in many other cases. Second, we will demonstrate that bisimilarity is undecidable over the whole class of Petri nets. Finally we give an overview of the current state of the art in the area.

0.4.1 Process rewrite systems

We start by introducing several well studied classes of infinite-state processes by means of process rewrite systems (PRS). Process rewrite systems are an elegant and universal approach defined, in the form presented in this chapter, by Mayr (Mayr, 2000). Mayr's definition unifies and extends some previously introduced formalisms for infinite-state systems (see e.g. the overview articles (Burkart, Caucal, Moller and Steffen, 2001; Burkart and Esparza, 1997; Moller, 1996)).

Let $\mathcal{C}onst$ be a set of *process constants*. 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

$$\begin{array}{rl} 1\text{:} & E ::= \epsilon \mid X \\ \mathcal{P}\text{:} & E ::= \epsilon \mid X \mid E \mid E \\ \mathcal{S}\text{:} & E ::= \epsilon \mid X \mid E.E \\ \mathcal{G}\text{:} & E ::= \epsilon \mid X \mid E \mid E \mid E.E \end{array}$$

where 'ϵ' is the *empty process*, X ranges over $\mathcal{C}onst$, the operator '.' stands for sequential composition and '|' stands for parallel composition. 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 $\mathcal{S} \cap \mathcal{P} = 1$.

We do not distinguish between process expressions related by a *structural congruence*, 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 both '.' and '|'.

Definition 0.4.1 Let $\alpha, \beta \in \{1, \mathcal{S}, \mathcal{P}, \mathcal{G}\}$ be classes of process expressions such that $\alpha \subseteq \beta$ and let Act be a set of *actions*. An (α, β)-PRS (Mayr, 2000) is a finite set

$$\Delta \subseteq (\alpha \setminus \{\epsilon\}) \times Act \times \beta$$

of *rewrite rules*, written $E \stackrel{a}{\longrightarrow} F$ for $(E, a, F) \in \Delta$. By $\mathcal{C}onst(\Delta)$ we denote the set of process constants that appear in Δ and by $Act(\Delta)$ the set of actions that appear in Δ.

An (α, β)-PRS determines a labelled transition system whose *states* are process expressions from the class β (modulo the structural congruence), Act is the set of *labels*, and the *transition relation* is the least relation satisfying the following SOS rules (recall that '|' is commutative).

$$\frac{(E \stackrel{a}{\longrightarrow} E') \in \Delta}{E \stackrel{a}{\longrightarrow} E'} \qquad \frac{E \stackrel{a}{\longrightarrow} E'}{E.F \stackrel{a}{\longrightarrow} E'.F} \qquad \frac{E \stackrel{a}{\longrightarrow} E'}{E \mid F \stackrel{a}{\longrightarrow} E' \mid F}$$

Many classes of infinite-state systems studied so far—e.g. basic process algebra (BPA), basic parallel processes (BPP), pushdown automata (PDA), Petri nets (PN) and process algebra (PA)—are contained in the hierarchy of process rewrite systems presented in Figure 0.10. This hierarchy is strict with respect to strong bisimilarity and we refer the reader to (Mayr, 2000) for further discussions.

We shall now take a closer look at the classes forming the PRS hierarchy.

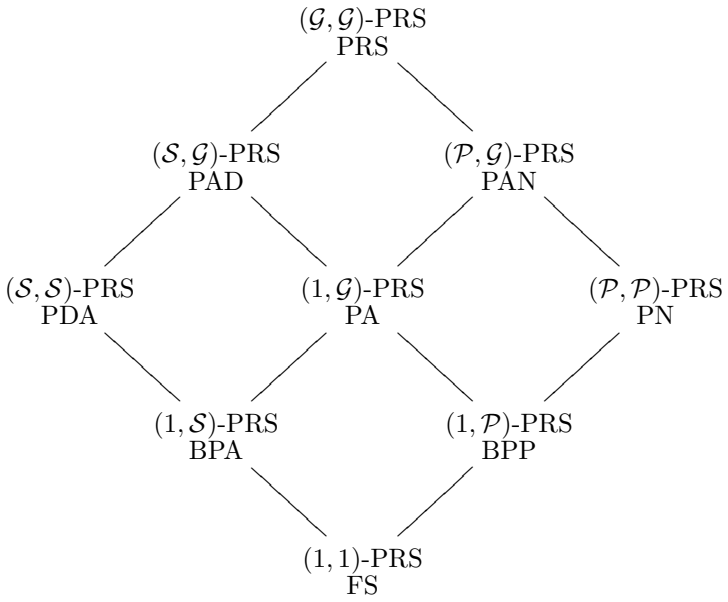


Fig. 0.10. Hierarchy of process rewrite systems

Finite-state processes. Finite-state processes (FS)—or equivalently $(1,1)$-PRS—generate a class of transition systems with finitely many reachable states. Here rules are of the form

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

where $X, Y \in \mathcal{C}onst(\Delta)$ and $a \in Act(\Delta)$.

Basic process algebra. Basic process algebra (BPA)—or equivalently $(1,\mathcal{S})$-PRS—represents the class of processes introduced by Bergstra and Klop in, for instance, (Bergstra and Klop, 1985). BPA is a model of purely sequential process behaviour such that a process constant can be replaced by a finite sequence of symbols via prefix rewriting. For example the rewrite rules

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

allow us to perform the sequence of transitions

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

where the sequential composition of the process constants behaves like a stack with the top on the left-hand side. This class also corresponds to the transition

systems associated with context-free grammars in Greibach normal form, in which only left-most derivations are allowed.

Basic parallel processes. Basic parallel processes (BPP)—or equivalently $(1, \mathcal{P})$-PRS—are a fragment of CCS (Milner, 1989) without restriction, relabelling and communication. It is a parallel analogue of BPA where any occurrence of a process constant inside a parallel composition can be replaced by a number of process constants put in parallel. For example, the rewrite rules

$$X \stackrel{a}{\longrightarrow} X \mid Y \mid Y \quad \text{and} \quad Y \stackrel{b}{\longrightarrow} \epsilon$$

allow us to derive the following sequence of transitions:

$$X \stackrel{a}{\longrightarrow} X \mid Y \mid Y \stackrel{b}{\longrightarrow} X \mid Y \stackrel{a}{\longrightarrow} X \mid Y \mid Y \mid Y \stackrel{b}{\longrightarrow} X \mid Y \mid Y \stackrel{a}{\longrightarrow} X \mid Y \mid Y \mid Y \mid Y \ .$$

The class BPP was first studied by Christensen (Christensen, 1993), 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*.

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. Caucal (Caucal, 1992) showed that an arbitrary unrestricted $(\mathcal{S}, \mathcal{S})$-PRS can be transformed into a PDA system (where rewrite rules are of the form $pX \stackrel{a}{\longrightarrow} q\gamma$ such that p and q are control states, X is a stack symbol and γ is a sequence of stack symbols) in such a way that their generated transition systems are isomorphic. Hence $(\mathcal{S}, \mathcal{S})$-PRS and PDA are equivalent formalisms.

PA-processes. PA-processes (PA for process algebra)—or equivalently $(1, \mathcal{G})$-PRS—represent the class of processes originally introduced by Bergstra and Klop in (Bergstra and Klop, 1984). This formalism combines the parallel and sequential operator but allows for neither communication nor global-state control.

Petri nets. Petri nets (PN)—or equivalently $(\mathcal{P}, \mathcal{P})$-PRS—represent the class of processes that correspond to the standard and well studied notion of labelled place/transition (P/T) nets as originally proposed by Petri (Petri, 1962). In fact, $(\mathcal{P}, \mathcal{P})$-PRS capture exactly the class of Petri nets with multiple arcs (see e.g. (Peterson, 1981)). The correspondence between $(\mathcal{P}, \mathcal{P})$-PRS and Petri nets is straightforward: process constants are names of places in the Petri net, any expression from $\mathcal{P}$ appearing during a computation is represented as a marking

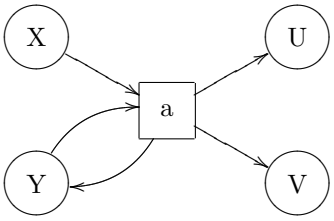


Fig. 0.11. A Petri net fragment corresponding to the rule $X \mid Y \xrightarrow{a} U \mid V \mid Y$

where the number of occurrences of a process constant gives the number of tokens in the place named after the process constant, and every rule gives rise to a labelled transition in the net as depicted on the example in Figure 0.11.

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; see (Mayr, 2000). The PAD class is the smallest common generalization of PA and PDA (Mayr, 1998) and PAN is a combination of the models PA and PN (Mayr, 1997a). The most general class PRS subsumes all the previously mentioned classes. It is worth mentioning that even the class $(\mathcal{G}, \mathcal{G})$-PRS is not Turing powerful since, e.g., the reachability problem (i.e., whether from a given process expression we can in finitely many steps reach another given process expression) remains decidable (Mayr, 2000). In fact, reachability remains decidable even in the recently introduced generalization of PRS called *weakly extended process rewrite systems* (Křetínský, Řehák and Strejček, 2005), where a finite control-unit with a monotonic behaviour provides an extra control over the process behaviours.

Further motivation for studying the classes from the PRS hierarchy can be found in (Esparza, 2002) and the classes from the PRS hierarchy also have a direct link with interprocedural control-flow analysis of programs (Esparza and Knoop, 1999).

Bisimilarity-checking problem. The problem we shall now study is whether we can decide the following:

Given two processes from some class in the PRS hierarchy, are the two processes bisimilar or not?

First, we shall argue that this problem is decidable for the BPP class and then we prove its undecidability for Petri nets.

0.4.2 Deciding bisimilarity on BPP using a tableau technique

Our aim is now to show decidability of bisimilarity on BPP. Christensen, Hirshfeld and Moller first proved this positive result in (Christensen, 1993; Christensen, Hirshfeld and Moller, 1993), using the so-called *tableau technique*. The presentation of the decidability result in this section is based on their proof.

We shall now introduce our running example. Without loss of generality we assume that in what follows any BPP system Δ uses the process constants $\mathcal{C}onst(\Delta) = \{X_1, \ldots, X_n\}$ for some n.

Example 0.4.2 Assume a BPP system Δ with the following rules.

$$\begin{array}{lcl} X_1 & \stackrel{a}{\longrightarrow} & X_1 \mid X_2 \\ X_2 & \stackrel{b}{\longrightarrow} & X_3 \\ X_3 & \stackrel{b}{\longrightarrow} & \epsilon \\ X_4 & \stackrel{a}{\longrightarrow} & X_4 \mid X_3 \mid X_3 \end{array}$$

We encourage the reader to draw initial fragments of labelled transition systems generated by the processes X_1 and X_4 and to establish that $X_1 \sim X_4$. □

We mention the standard fact that any BPP process E over Δ can be viewed as a Parikh vector $\phi(E) = (k_1, k_2, \ldots, k_n) \in \mathbb{N}^n$, where $\mathcal{C}onst(\Delta) = \{X_1, X_2, \ldots, X_n\}$ and k_i is the number of occurrences of X_i in E. Hence the Parikh vector simply counts the number of occurrences of the different process constants in a given expression. Clearly, two BPP processes E and F are structurally congruent if and only if $\phi(E) = \phi(F)$.

Example 0.4.3 The rules from Example 0.4.2 can be viewed as rules over Parikh vectors in the following way.

$$\begin{array}{lcl} (1,0,0,0) & \stackrel{a}{\longrightarrow} & (1,1,0,0) \\ (0,1,0,0) & \stackrel{b}{\longrightarrow} & (0,0,1,0) \\ (0,0,1,0) & \stackrel{b}{\longrightarrow} & (0,0,0,0) \\ (0,0,0,1) & \stackrel{a}{\longrightarrow} & (0,0,2,1) \end{array}$$

A sequence of transitions

$$X_1 \stackrel{a}{\longrightarrow} X_1 \mid X_2 \stackrel{a}{\longrightarrow} X_1 \mid X_2 \mid X_2 \stackrel{b}{\longrightarrow} X_1 \mid X_2 \mid X_3 \stackrel{b}{\longrightarrow} X_1 \mid X_2$$

then has a straightforward analogue over Parikh vectors, namely

$$(1,0,0,0) \stackrel{a}{\longrightarrow} (1,1,0,0) \stackrel{a}{\longrightarrow} (1,2,0,0) \stackrel{b}{\longrightarrow} (1,1,1,0) \stackrel{b}{\longrightarrow} (1,1,0,0)\ .$$

□

Definition 0.4.4 Let $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n$ and $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{N}^n$.

- By $\alpha + \beta$ we denote a component-wise addition defined by $\alpha + \beta = (\alpha_1 + \beta_1, \ldots, \alpha_n + \beta_n) \in \mathbb{N}^n$.
- We define a lexicographical ordering $<_\ell$ such that $\alpha <_\ell \beta$ iff there is some k, $0 \leq k \leq n$, such that $\alpha_k < \beta_k$ and $\alpha_i = \beta_i$ for all i, $1 \leq i < k$.
- We define a component-wise ordering $\leq_c$ such that $\alpha \leq_c \beta$ iff $\alpha_i \leq \beta_i$ for every i, $1 \leq i \leq n$, i.e., iff there is $\alpha' \in \mathbb{N}^n$ such that $\beta = \alpha + \alpha'$. We write $\alpha <_c \beta$ iff $\alpha \leq_c \beta$ and $\alpha \neq \beta$.

Observe that $<_\ell$ is a well-founded ordering—that is, there is no infinite sequence $\alpha_1, \alpha_2, \ldots$ such that $\alpha_1 >_\ell \alpha_2 >_\ell \ldots$—and note that $\alpha \neq \beta$ implies that either $\alpha <_\ell \beta$ or $\beta <_\ell \alpha$. The following fact is known as Dickson's Lemma.

Lemma 0.4.5 ((Dickson, 1913)) Every infinite sequence from $\mathbb{N}^n$ has an infinite nondecreasing subsequence with respect to $\leq_c$.

Before we describe the construction of a tableau proving bisimilarity of BPP processes, we introduce some simple facts about stratified bisimulation relations introduced in Definition 0.2.1 that will be used in the proof of soundness.

We note, first of all, that labelled transition systems generated by BPP processes are clearly image-finite.

An important property of labelled transition systems generated by BPP processes is that the stratified bisimulation relations $\sim_k$ are congruences. This fact will be essential in the soundness proof of the tableau construction.

Lemma 0.4.6 Let E, F and G be BPP processes over Δ. If $E \sim_k F$ then $(E \mid G) \sim_k (F \mid G)$ for any $k \in \mathbb{N}$.

Exercise 0.4.7 Prove Lemma 0.4.6.

By Lemma 0.2.2 and Lemma 0.4.6 we can conclude that bisimilarity on BPP is a congruence.

Lemma 0.4.8 Let E, F and G be BPP processes over Δ. If $E \sim F$ then $(E \mid G) \sim (F \mid G)$.

We can now proceed with the description of the tableau technique in order to demonstrate decidability of bisimilarity on BPP. Let E and F be BPP processes over the set of rules Δ. Recall that by $\phi(E)$ and $\phi(F)$ we denote the corresponding Parikh vectors over $\mathbb{N}^n$.

A *tableau* for E and F is a maximal proof tree rooted with $(\phi(E), \phi(F))$ and built according to the following rules. Let $(\alpha, \beta) \in \mathbb{N}^{2n}$ be a node in the tree.

A node (α, β) is either *terminal (leaf)* or *nonterminal*. The following nodes are terminal:

- (α, α) is a *successful leaf* for any $\alpha \in \mathbb{N}^n$,
- (α, β) is a *successful leaf* if $\mathsf{next}(\alpha, *) \cup \mathsf{next}(\beta, *) = \emptyset$,
- (α, β) is an *unsuccessful leaf* if for some $a \in Act(\Delta)$ it is the case that $\mathsf{next}(\alpha, a) \cup \mathsf{next}(\beta, a) \neq \emptyset$, and either $\mathsf{next}(\alpha, a) = \emptyset$ or $\mathsf{next}(\beta, a) = \emptyset$.

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.

$$\text{RED}_L \quad \frac{(\alpha, \beta)}{(\gamma + \omega, \beta)} \quad \begin{array}{l}\text{if there is an ancestor } (\gamma, \delta) \text{ or } (\delta, \gamma) \text{ of } (\alpha, \beta) \text{ such} \\ \text{that } \gamma <_\ell \delta \text{ and } \alpha = \delta + \omega \text{ for some } \omega \in \mathbb{N}^n\end{array}$$

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

If no reduction RED is applicable and the resulting node is not a leaf, we apply the rule EXPAND for a set of relations S_a, $a \in Act(\Delta)$, where $S_a \subseteq \mathsf{next}(\alpha, a) \times \mathsf{next}(\beta, a)$ such that

- for each $\alpha' \in \mathsf{next}(\alpha, a)$, there is some $\beta' \in \mathsf{next}(\beta, a)$ such that $(\alpha', \beta') \in S_a$ and
- for each $\beta' \in \mathsf{next}(\beta, a)$, there is some $\alpha' \in \mathsf{next}(\alpha, a)$ such that $(\alpha', \beta') \in S_a$.

Note that there are several possible relations S_a that might satisfy the condition above.

$$\text{EXPAND} \quad \frac{(\alpha, \beta)}{\{(\alpha', \beta') \mid a \in Act(\Delta) \ \wedge \ (\alpha', \beta') \in S_a\}}$$

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 such child (which is not a leaf) as long as possible. Note that reduction rules are applicable to a node iff the node is not terminal (leaf).

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

Example 0.4.9 A successful tableau for X_1 and X_4 of the BPP system Δ from

Example 0.4.2 is given below.

$$\dfrac{\dfrac{\dfrac{\dfrac{(1,0,0,0),(0,0,0,1)}{(1,1,0,0),(0,0,2,1)}\,\text{EXPAND}}{(0,1,0,1),(0,0,2,1)}\,\text{RED}_L}{\dfrac{(0,1,2,1),(0,0,4,1)}{(0,0,4,1),(0,0,4,1)}\,\text{RED}_L} \qquad \underline{(0,0,1,1),(0,0,1,1)}}\,\text{EXPAND}$$

Note that for the first application of the rule EXPAND there was only one choice how to pair the successor processes (due to the fact that X_1 and X_2 have unique a-successors). Then the rule RED_L is applicable because

$$\underbrace{(0,0,0,1)}_{\gamma} <_\ell \underbrace{(1,0,0,0)}_{\delta}\,,$$

$$\underbrace{(1,1,0,0)}_{\alpha} = \underbrace{(1,0,0,0)}_{\delta} + \underbrace{(0,1,0,0)}_{\omega} \text{ and}$$

$$(0,1,0,1) = \underbrace{(0,0,0,1)}_{\gamma} + \underbrace{(0,1,0,0)}_{\omega}\,.$$

Now no more RED_L nor RED_R are applicable, so we can apply the rule EXPAND. Again, there is a unique a-successor in both processes (left child) and a unique b-successor in both processes (right child). The right child is by definition a successful leaf. By one more application of the RED_L rule on the left child where $\underbrace{(0,0,2,1)}_{\gamma} <_\ell \underbrace{(0,1,0,1)}_{\delta}$ and where $\omega = (0,0,2,0)$, we complete the tableau as the created child is a successful leaf. □

Lemma 0.4.10 Any tableau for BPP processes E and F is finite and there are only finitely many tableaux.

Proof We first show that any tableau for E and F is finite. To this end, observe, first of all, that any tableau for E and F is finitely branching because $Act(\Delta)$ is a finite set and for a given $a \in Act(\Delta)$ any relation S_a is finite and there are finitely many such relations. Should the tableau be infinite, there must be an infinite branch, which gives an infinite sequence of vectors from $\mathbb{N}^{2n}$. 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 Dickson's Lemma 0.4.5, this sequence must contain an infinite nondecreasing subsequence

$(\alpha_1, \beta_1) \leq_c (\alpha_2, \beta_2) \leq_c \ldots$. However, the rule EXPAND cannot be applied on (α_2, β_2) since one of the RED rules is applicable. This is a contradiction.

Since there are only finitely many relations S_a for any $a \in Act(\Delta)$ available in the EXPAND rule and there are finitely many possibilities for an application of the RED rule, there are always finitely many possibilities for extending an already existing partial tableau. Suppose that there were infinitely many tableaux starting from $(\phi(E), \phi(F))$. Then there must be a tableau with an infinite branch, which contradicts that every tableau is finite. □

Lemma 0.4.11 (Completeness) Let E and F be two BPP processes over Δ. If $E \sim F$ then there is a successful tableau for E and F.

Proof We construct inductively a tableau with root $(\phi(E), \phi(F))$ 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 0.4.10. 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, without loss of generality, that (γ, δ) is an ancestor of (α, β). By induction, $\gamma \sim \delta$, which means by Lemma 0.4.8 that $(\gamma + \omega) \sim (\delta + \omega) = \alpha \sim \beta$. Hence $(\gamma + \omega) \sim \beta$. Similarly for RED_R. From the definition of bisimilarity it follows that the rule EXPAND is also forward sound, i.e., if $\alpha \sim \beta$ then we can choose for every $a \in Act(\Delta)$ a relation S_a such that $(\alpha', \beta') \in S_a$ implies that $\alpha' \sim \beta'$. □

Lemma 0.4.12 (Soundness) Let E and F be two BPP processes over Δ. If there is a successful tableau for E and F then $E \sim F$.

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

If $E \not\sim F$ then using Lemma 0.2.2 there is a minimal k such that $E \not\sim_k F$. Notice that if $\alpha \not\sim_k \beta$ such that k is minimal and we apply the rule EXPAND, 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 $(\gamma + \omega) \not\sim_k \beta$, otherwise using Lemma 0.4.6 we get that $\alpha = (\delta + \omega) \sim_k (\gamma + \omega) \sim_k \beta$, 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 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 can now conclude that it is decidable whether $E \sim F$ for given BPP processes E and F. Indeed, $E \sim F$ iff there is a successful tableau for E and F (Lemma 0.4.11 and Lemma 0.4.12). Moreover, there are only finitely many tableaux and all of them are finite. We can therefore state the following theorem.

Theorem 0.4.13 ((Christensen et al., 1993)) It is decidable whether $E \sim F$ for any two given BPP processes E and F.

Remark 0.4.14 As mentioned above, one can in principle view BPP processes as elements over the free commutative monoid $(\mathbb{N}^n, +, (0, \ldots, 0))$. Because bisimilarity is a congruence over this monoid (Lemma 0.4.8), we can apply a classical result to the effect that every congruence on a finitely generated commutative semigroup is finitely generated (Redei, 1965), in order to derive the positive decidability result. For further references on this topic see (Hirshfeld, 1994a; Burkart et al., 2001; Jančar, 2008).

Unfortunately, the complexity of the above-presented tableau-based proof for decidability of bisimilarity on BPP is unknown—no primitive recursive upper bound was given. Only recently, Jančar proved the containment of the problem in PSPACE (Jančar, 2003) using a different technique based on the so-called dd-functions. Together with the previously known PSPACE-hardness result from (Srba, 2003), PSPACE-completeness of the problem was finally shown.

Even though the tableau technique did not provide the best complexity upper bound in this case, it demonstrates a useful proof strategy that is applicable also to several other classes of systems. A successful tableau can be viewed as a compact (finite) representation of Defender's winning strategy (which is in general infinite). What is, in principle, happening during the tableau construction is that we repeatedly use the rule EXPAND in order to simulate possible attacks on the pair of processes present in the tableau. Defender's strategy corresponds to the selection of appropriate sets S_a for every available action a. Because for infinite state systems a construction of such a tableau using only the EXPAND rule might not terminate, we need a way to ensure termination. How to do this depends on the particular infinite-state system in question. In case of BPP, the guarantee of termination is based on the fact that bisimilarity is a congruence, and we can therefore ensure that any constructed tableau is finite by means of the application of the rules RED_L and RED_R. We can call this approach *simplification* of the sub-goals.

In fact, the assumptions in the proof of soundness and completeness of the tableau technique are quite general, which allows us to extend the technique to cover several other variants/extensions of BPP processes like e.g. lossy BPP, BPP with interrupt and discrete timed-arc BPP nets (Srba, 2002).

The tableau technique has been frequently used in the literature. Here we limit ourselves to mentioning a few references relevant to the classes in the PRS hierarchy. Hüttel and Stirling designed tableau rules in order to show that bisimilarity is decidable on normed BPA (Hüttel and Stirling, 1998). (Normed means that all process constants in the BPA system can be reduced to the empty process ϵ in finitely many transitions.) In that reference, the authors use a similar congruence property in order to simplify the sub-goals in the tableau. A tableau technique proved useful also for the decidability of *weak* bisimilarity on a subclass of BPP processes (Stirling, 2001).

Considerably more involved applications of the tableau technique can be found in the proofs of decidability of bisimilarity for normed PA-processes (Hirshfeld and Jerrum, 1999), normed pushdown systems (Stirling, 1998) as well as for general pushdown system (Stirling, 2000).

Other uses of the tableau technique include model checking problems for various temporal logics (Bradfield, Esparza and Mader, 1996; Cleaveland, 1990; Lichtenstein and Pnueli, 1985; Stirling and Walker, 1991) and some other problems, such as those addressed in (Mayr, 1997b; Stirling, 2003).

As already mentioned for the class BPP, tableau techniques do not always provide the most efficient algorithms from the computational complexity point of view. There have been other techniques developed for proving decidability of bisimilarity, most notably the techniques based on *finite bisimulation bases*. In some instances these techniques provide polynomial time algorithms for bisimilarity on infinite-state systems. For a regularly updated list of references consult (Srba, 2004). Readers interested in an intuitive explanation of such techniques are directed to some recent overview articles (Kučera and Jančar, 2006; Jančar, 2008).

0.4.3 Undecidability of bisimilarity on Petri nets

In this section we shall demonstrate a negative result for bisimilarity checking on infinite-state labelled transition systems generated by Petri nets. The presentation of the result is to a large extend based on (Jančar and Srba, 2008), slightly simplifying the original proof by Jančar (Jančar, 1995). The proof is done by reduction from the halting problem for 2-counter Minsky machines.

A *(Minsky) 2-counter machine (MCM) M*, with *nonnegative* counters c_1 and c_2, is a sequence of (labelled) instructions:

$$1 : instr_1;\ 2 : instr_2;\ \ldots\ n : instr_n$$

where $instr_n = \text{HALT}$ and each $instr_i$, for $i \in \{1, 2, \ldots, n-1\}$, is of the following two types (assuming $j, k \in \{1, 2, \ldots, n\}$, $r \in \{1, 2\}$):

Type (1) $c_r := c_r + 1$; goto j
Type (2) if $c_r = 0$ then goto j else ($c_r := c_r - 1$; goto k)

The instructions of type (1) are called *increment instructions*, the instructions of type (2) are *zero-test (and decrement) instructions*.

The *computation* of M on the input $(i_1, i_2) \in \mathbb{N} \times \mathbb{N}$ is the sequence of configurations (i, n_1, n_2), starting with $(1, i_1, i_2)$, where $i \in \{1, 2, \ldots, n\}$ is the label of the instruction to be performed, and $n_1, n_2 \in \mathbb{N}$ are the (current) counter values; the sequence is determined by the instructions in the obvious way. The computation is either finite, i.e. halting by reaching the instruction n : HALT, or infinite.

We define *inf-MCM* as the problem to decide whether the computation of a given 2-counter MCM on $(0, 0)$ is infinite, and we recall the following well-known fact.

Proposition 0.4.15 ((Minsky, 1967)) Problem inf-MCM is undecidable.

Given a 2-counter machine M with n instructions, we construct a $(\mathcal{P}, \mathcal{P})$-PRS system (a Petri net) with $\mathcal{C}onst = \{q_1, q_2, \ldots, q_n, q'_1, q'_2, \ldots, q'_n, C_1, C_2\}$ in which $q_1 \sim q'_1$ iff the computation of M on $(0, 0)$ is infinite. We use the pair of process constants q_i, q'_i for (the label of) the ith instruction, $i \in \{1, 2, \ldots, n\}$, and the constants C_1, C_2 for unary representations of the values of the counters c_1, c_2. We define the following rules corresponding to (all) instructions of M (where action a can be read as 'addition', i.e., increment, d as 'decrement', z as 'zero', and h as 'halt').

'$i : c_r := c_r + 1$; goto j':

$$q_i \xrightarrow{a} q_j \mid C_r \qquad\qquad q'_i \xrightarrow{a} q'_j \mid C_r$$

'i : if $c_r = 0$ then goto j else ($c_r := c_r - 1$; goto k)':

$$\begin{array}{ll} q_i \mid C_r \xrightarrow{d} q_k & q'_i \mid C_r \xrightarrow{d} q'_k \\ q_i \xrightarrow{z} q_j & q'_i \xrightarrow{z} q'_j \\ q_i \mid C_r \xrightarrow{z} q'_j \mid C_r & q'_i \mid C_r \xrightarrow{z} q_j \mid C_r \end{array}$$

'n : HALT':

$$q_n \xrightarrow{h} \epsilon$$

Let (i, n_1, n_2) be a configuration of M. It will be represented by the pair of processes

$$q_i \mid C_1^{n_1} \mid C_2^{n_2} \quad \text{and} \quad q_i' \mid C_1^{n_1} \mid C_2^{n_2}$$

where $C_r^{n_r}$ for $r \in \{1, 2\}$ is a parallel composition of n_r process constants C_r. If the current instruction is increment, it is clear that, no matter how Attacker and Defender play, in one round they reach a pair of processes that represent the successor configuration (one occurrence of C_r is added on both sides). If the current instruction is zero-test and decrement and Attacker plays the action d, Defender can only mimic the move under d in the other process and both players remove one occurrence of C_r on both sides, again reaching a configuration representing the successor configuration of M. The same happens if the counter c_r is empty and Attacker plays using the rule $q_i \xrightarrow{z} q_j$ or $q_i' \xrightarrow{z} q_j'$. Defender's only choice is to play using the rule $q_i' \xrightarrow{z} q_j'$ or $q_i \xrightarrow{z} q_j$ in the other process and the players faithfully simulate the computation of the machine M. The only situation where Attacker can 'cheat' is when the counter c_r is not empty and Attacker plays the action z according to one of the four available rules (two on the left-hand side and two on the right-hand side). Nevertheless, Defender can in this case answer on the other side under the action z is such a way that the players reach a pair of syntactically equal processes—a clear win for Defender.

The rules for the zero-test and decrement instruction provide another example of the use of *Defender's Forcing Technique* (Jančar and Srba, 2008) as it was already explained in the proofs of Theorem 0.3.5 and Theorem 0.3.16.

To sum up we note that if M halts on $(0, 0)$ then Attacker can force the correct simulation of M in both components of the starting pair q_1 and q_1' and finally win by playing $q_n \xrightarrow{h} \epsilon$ (because there is no such rule for q_n'). If the computation of M on $(0, 0)$ is infinite then Defender forces Attacker to perform the correct simulation of the infinite computation and Defender wins the game.

This implies the following undecidability result.

Theorem 0.4.16 ((Jančar, 1995)) Bisimilarity on Petri nets is undecidable.

Remark 0.4.17 We remind the reader of the fact that the simulation of a 2-counter Minsky machine by a Petri net, as outlined above, provides only a so-called *weak simulation*. This means that the Petri net can mimic any behaviour of the Minsky machine, but also displays some additional 'cheating' behaviours. Indeed, the Petri net might decide to use the action z (for zero) even if the counter is not empty. This means, for instance, that the reachability problem for Petri nets is still decidable (as mentioned before even for the whole class of

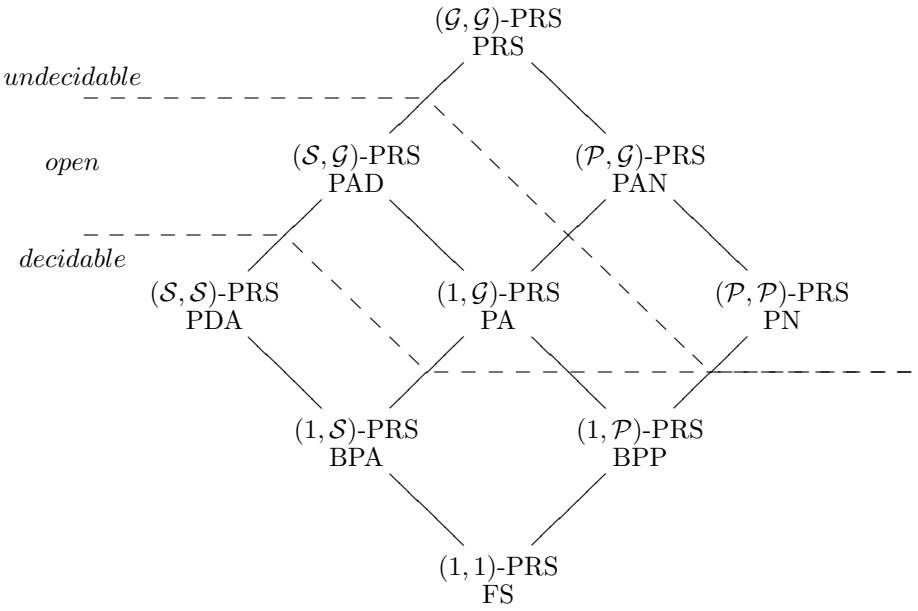


Fig. 0.12. (Un)decidability results for strong bisimilarity

PRS, a superclass of Petri nets), but bisimilarity has the extra power to filter out such incorrect behaviours and hence bisimilarity checking is undecidable.

0.4.4 Overview of results

An overview of the state of the art for strong bisimilarity checking over the classes of processes from the PRS hierarchy is provided in Figure 0.12. As shown in Theorem 0.4.16, bisimilarity is undecidable for PN and the two classes above it. It is on the other hand decidable for PDA (Senizergues, 1998; Stirling, 2000) and hence also for BPA (direct proofs are also available in (Christensen, Hüttel and Stirling, 1995; Burkart, Caucal and Steffen, 1995)), and for BPP (Theorem 0.4.13). It is remarkable that most other behavioral equivalences apart from bisimilarity are undecidable already for BPA and BPP. Bar-Hillel, Perles, and Shamir (Bar-Hillel, Perles and Shamir, 1961) showed that the equivalence problem for languages generated by context-free grammars is undecidable. In fact, all the equivalences in van Glabbeek's spectrum (Glabbeek, 2001) (apart from bisimilarity) are undecidable for BPA (Huynh and Tian, 1995; Groote and Hüttel, 1994). For the case of BPP, Hirshfeld (Hirshfeld, 1994b) showed that once again language equivalence is undecidable and as before none of the equivalences from van Glabbeek's spectrum coarser than bisimilarity are decid-

able (Hüttel, 1994). Finally, we note that decidability of bisimilarity remains an open problem for PAD and PA, though for normed PA processes a positive decidability result already exists (Hirshfeld and Jerrum, 1999).

For the case of weak bisimilarity, we know that it is not only undecidable but even highly undecidable (complete for the class Σ_1^1 on the first level of the analytical hierarchy) for PDA, PA and PN (Jančar and Srba, 2008). The problem still remains open for BPA and BPP.

For further references the reader may consult (Srba, 2004), where an on-line and updated list of results is available (including also results for strong/weak regularity checking problems). Some recent overview articles (Burkart et al., 2001; Moller, Smolka and Srba, 2004; Kučera and Jančar, 2006; Jančar, 2008) provide further introduction to the area of equivalence checking over infinite-state systems.

0.5 The use of bisimilarity checking in verification and tools

As we stated in Section 0.1, a model of computation like that of labelled transition systems can be used to describe both implementations of processes and specifications of their expected behaviours. A notion of behavioural equivalence or preorder over labelled transition systems can therefore be used as the formal yardstick to establish the correctness of a proposed implementation with respect to a specification, when both are represented as objects in that model. Furthermore, from a foundational viewpoint, in the single-language approach to process theory, the informal notion of 'process' is formally characterized as an equivalence class of labelled transition systems with respect to some chosen behavioural relation.

In the light of the importance of behavioural relations in the single-language approach to process theory and of the lack of consensus on what constitutes a 'reasonable' notion of observable behaviour for reactive systems, it is perhaps not overly surprising that the literature on concurrency theory offers a large variety of behavioural semantics over labelled transition systems. (See the encyclopaedic survey paper (Glabbeek, 2001) and the conference article (Glabbeek, 1993) for the definition of a large number of those relations. Chapter 1 in this book provides a small representative sample.) However, as the results we surveyed in the previous sections indicate, various notions of bisimilarity stand out amongst the plethora of available behavioural semantics because of their pleasing mathematical properties and the remarkable algorithmic, complexity and decidability results that are available for them. Those algorithmic results make notions of bisimilarity the prime candidates for implementation in software tools for computer-aided verification based on the single-language approach to pro-

cess theory, such as CADP (Garavel et al., 2007), the Edinburgh Concurrency Workbench and its further incarnations (Cleaveland et al., 1993; Cleaveland and Sims, 1996), and mCRL2 (Groote et al., 2008) to name but a few. After about two decades from the first development of such tools and of their use in computer-aided verification, it is natural to try and assess the actual applications of bisimilarity checking in the practice of verification. This is our aim in the present section. Let us state at the outset that we cannot possibly hope to give a comprehensive account of the available software tools and of their applications here. What we shall try to do instead is to highlight some of the most common uses of bisimilarity checking in the practice of (computer-aided) verification and sometimes how, and whether, they are used in applications of the representative tools we have mentioned above. Apart from the references we present below, our presentation is also based on personal communications with Rance Cleaveland, Hubert Garavel and Jan Friso Groote.

0.5.1 Some uses of bisimilarity checking

Variations on the notion of bisimilarity play several roles in verification. In the remainder of this section, in order to focus our discussion, we shall only consider those that, to our mind, are the most common and/or remarkable ones.

Bisimilarity as a correctness criterion. As mentioned above, bisimilarity can be used as the notion of behavioural relation for establishing formally that implementations are correct with respect to specifications expressed as labelled transition systems. In the light of the results we discussed in previous sections, the choice of bisimilarity as a formal yardstick for proving correctness is algorithmically attractive.

However, notions of bisimilarity are amongst the finest semantics in van Glabbeek's lattice of behavioural semantics over labelled transition systems. This begs the questions whether bisimulation-like behavioural equivalences are coarse enough in practice and whether such an approach to verification is widely applicable. Answers to those questions vary from researcher to researcher and we are not aware of systematic studies on the applicability of bisimilarity in verification. However, the experts we consulted tended to agree that weak and branching bisimilarity (Glabbeek and Weijland, 1996; Milner, 1989) are, in general, adequate in applications and are far more efficient than other alternatives, both in manual and in computer-aided correctness proofs. By way of example, Rance Cleaveland, one of the chief architects behind the development and the applications of the Concurrency Workbench of the New Century (Cleaveland and Sims, 1996), wrote to us (Cleaveland, 2009):

I personally have not found bisimulation to be unnecessarily restrictive when comparing a specification LTS to an implementation LTS, but this is because in almost all the cases I have worked with, the specification is deterministic.

Hubert Garavel, the chief architect of CADP (Garavel et al., 2007), supported this view and wrote (Garavel, 2009):

In practice, you can do a lot by using strong bisimulation, branching bisimulation, and safety equivalence together (with their preorders). I do not remember a case where a more exotic equivalence would have been mandatory.

(Safety equivalence was introduced in (Bouajjani, Fernandez, Graf, Rodriguez and Sifakis, 1991) in order to provide a topological characterization of safety properties in branching-time semantics. Two processes are related by the safety preorder iff whenever the left-hand side process performs a sequence of τ actions followed by a visible action a then the right-hand side can also perform a sequence of τ actions followed by a in such a way that the resulting processes are again related by the safety preorder. It turns out that, mirroring the situation in linear-time semantics (Abadi and Lamport, 1991; Manna and Pnueli, 1989), safety properties are exactly the closed sets in the topology induced by this preorder.) There are a few notorious examples of implementations and specifications that are not bisimulation equivalent, such as Bloom's two-writer register (Bloom, 1988), which can be proved correct in trace semantics. However, such examples are rare (Groote, 2009).

There are also some interesting variations on using bisimilarity as a direct verification technique. The INRIA-Sophia Antipolis team of Robert de Simone and others proposed the notion of 'observation criteria' (Boudol, Roy, de Simone and Vergamini, 1989). The basic idea behind this notion is

(1) to define contexts (that is, open systems with which the implementation labelled transition system is expected to interact) describing 'usage scenarios',
(2) to put the system being verified in these contexts and
(3) then use bisimilarity to compare the composite system to a labelled transition system representing an expected result.

One example of successful application of this approach is provided by (Ray and Cleaveland, 2004), where Ray and Cleaveland used this idea to good effect in verifying a medical-device model. (In that paper, the verification method is called 'unit verification'.) A closely related analysis technique is presented in (Vaandrager, 1990), together with some applications. All these approaches may be seen as special instances of the notion of context-dependent bisimilarity

developed in (Larsen, 1986) and are based on the idea to combat the state-explosion problem by focusing on fragments of system behaviour; the context is responsible for identifying the relevant fragments. An early example of a compositional protocol verification using context-dependent bisimilarity may be found in (Larsen and Milner, 1992).

According to Jan Friso Groote (Groote, 2009), when using the verification tool mCRL2 (Groote et al., 2008), a very common approach is currently to minimize a state space with respect to weak or branching bisimilarity and to inspect the resulting labelled transition system using a visualization tool (in their setting, the tool `ltsgraph` from the mCRL2 tool set). (If the graphs are not too large, reducing them further using weak trace equivalence often simplifies the picture even more and is satisfactory when the interesting properties are trace properties.) Bisimilarity is the key technology for this approach, which is also supported and adopted by CADP (Garavel et al., 2007) when the state space of the (minimized) model is small enough that one can draw it and verify it visually.

The debate on which notion of behavioural relation is the most 'reasonable' (in some sense) and the most suitable in verification (see also the discussion in Chapter 1) has been going on at least since the publication of Milner's first book on the Calculus of Communicating Systems (Milner, 1980). (See, for instance, the postings related to this issue that appeared in a very interesting debate that raged on the Concurrency mailing list in the late 1980s and early 1990s (*Concurrency mailing list archive*, 1988–1990).) We feel that this book chapter is not the most appropriate setting for a thorough discussion of the relative merits of linear and branching time semantics in verification, as this would lead us astray from the main gist of this contribution. We refer our readers to, for example, the articles (Glabbeek, 1994; Nain and Vardi, 2007) for further interesting discussions of this issue and more pointers to the literature. In the former reference, van Glabbeek presents arguments for performing verifications using behavioural relations that, like various forms of bisimilarity, preserve the 'internal structure' of processes. The advantage of this approach is that verifications using such equivalences apply to any coarser behavioural relation based on testing scenarios. In the latter reference, Nain and Vardi argue instead that branching-time-based behavioural equivalences are not suitable for use in verification because they distinguish processes that cannot be told apart by any reasonable notion of observation. (See also the article (Bloom, Istrail and Meyer, 1995).)

ADD EXACT REFERENCE TO CHAPTER 1

We hope that we have provided enough information to our readers so that they can draw draw their own conclusions on this long standing debate.

Bisimilarity as a tool for checking other behavioural relations. Algorithms for computing bisimilarity over finite labelled transition systems can be used to compute other behavioural relations. See, for instance, the paper (Cleaveland and Hennessy, 1993) for an early example of this application of bisimilarity. In that reference, Cleaveland and Hennessy showed how to compute the testing semantics from (De Nicola and Hennessy, 1984) by using algorithms for bisimilarity checking. The resulting algorithms have been implemented in the Edinburgh Concurrency Workbench and its further incarnations (Cleaveland et al., 1993; Cleaveland and Sims, 1996).

Moreover, minimization with respect to bisimilarity can be used to improve on the efficiency of computing other behavioural relations. Indeed, since bisimilarity is a finer relation than the others in van Glabbeek's lattice, one can minimize systems before comparing them with respect to the chosen behavioural semantics.

In the specific case of the classic simulation preorder (see Chapter 1 in this book or (Glabbeek, 2001) for a definition of this relation), there has been some interesting work on combining the computation of the simulation preorder with bisimulation minimization. The main idea behind this line of work is to intertwine the computation of simulation with minimization, so that one can terminate early whenever the constraints of the simulation preorder are violated—see, for instance, the paper (Tan and Cleaveland, 2001), which reports experimental data indicating that this approach yields a substantial improvement upon the best known algorithm for computing the simulation preorder, even when the systems are minimized with respect to bisimulation equivalence before applying the simulation algorithm. It is interesting to note that, even though the simulation preorder looks deceptively similar to bisimilarity, recasting the simulation problem as a coarsest partition problem, which is the basic idea underlying the algorithms we discussed in Section 0.2, does not appear to be at all easy. See, for instance, the paper (Glabbeek and Ploeger, 2008) for a detailed discussion of this issue in relation to earlier proposals.

Bisimulation minimization as a pre-processing step. Minimizing a finite labelled transition system modulo bisimilarity can be used to reduce the size of a system description before applying model checking (Clarke, Grumberg and Peled, 2000). The use of bisimilarity in this context is particularly attractive because, as essentially shown in (Hennessy and Milner, 1985), bisimilarity provides an abstraction technique that preserves the truth and falsehood of any formula expressed in the μ-calculus (Kozen, 1983), and hence all CTL* (Dam, 1994), CTL (Clarke and Emerson, 1981; McMillan, 1993), and LTL (Pnueli, 1977)

properties. Moreover, as we saw in Section 0.2.4, bisimilarity can be computed symbolically and automatically over finite labelled transition systems.

Despite the aforementioned mathematically pleasing properties of minimization with respect to bisimilarity, which make it a promising pre-processing step before applying model checking, and the increasing use of model checking formulae in the modal μ-calculus with data, state-space reduction is not used in verifications using mCRL2 (Groote, 2009). Instead, mCRL2 reduces the model checking problem to that of finding a solution to a *Parameterized Boolean Equation System* (Groote and Willemse, 2005). As Jan Friso Groote wrote to us (Groote, 2009):

> Bisimulation reduction is not of much (practical) relevance here.

This conclusion is supported by the work by Fisler and Vardi reported in (Fisler and Vardi, 1998; Fisler and Vardi, 2002). In (Fisler and Vardi, 1998), the authors provided experimental evidence showing that the use of bisimulation minimization as a pre-processing step to model checking does reduce the resource requirements of model checking. However, the work presented in that reference also indicates that the cost of bisimulation minimization often exceeds that of model checking significantly. In (Fisler and Vardi, 2002), Fisler and Vardi provide experimental and analytical comparisons between symbolic model checking for invariance properties and three novel ROBBD-based algorithms that combine bisimulation minimization and model checking based on the original algorithms given in (Bouajjani, Fernandez and Halbwachs, 1991; Lee and Yannakakis, 1992; Paige and Tarjan, 1987). Their results indicate that the integrated algorithms are less efficient than model checking and strongly suggest that performing bisimulation minimization does not lead to improvements over the direct use of model checking.

Bisimilarity in compositional verification. In the context of compositional verification approaches, bisimilarity can be used to minimize components before each composition. This is the so-called minimize-then-compose heuristic, which can also be used as a means to combat the state-explosion problem in the generation of the labelled transition system for a composite system from those for its components.

Such an approach is used substantially in verifications using the tools CADP and FDR (Formal Systems (Europe) Ltd., 2000), but is never used in applications of mCRL2 (Groote, 2009). According to Jan Friso Groote (Groote, 2009), the benefits offered by bisimilarity do not differ very much from those provided by other equivalences in van Glabbeek's spectrum in this respect, but its algorithmic efficiency makes it more applicable than other behavioural semantics.

Bisimilarity as a tool to handle infinite-state systems. Minimization of a system with respect to bisimilarity can sometimes be used to collapse infinite labelled transition systems to finite ones. A system that is bisimilar to a finite one is often called a *regular* process. Regularity checking has been studied substantially over the classes of infinite-state systems from the PRS hierarchy—see, e.g., (Srba, 2004) for an overview of results in that area.

The existence of a finite quotient of a hybrid or real-time system with respect to bisimilarity is the key theoretical step behind tool support for the verification of such systems (Aceto et al., 2007; Alur and Dill, 1994; Alur, Henzinger, Lafferriere and Pappas, 2000). In the case of hybrid and timed automata, bisimilarity, while not explicitly present in the implementation of the algorithms embodied in tools such as HyTech (Henzinger, Ho and Wong-Toi, 1997), Kronos (Bozga, Daws, Maler, Olivero, Tripakis and Yovine, 1998) and Uppaal (Behrmann, David, Larsen, Håkansson, Pettersson, Yi and Hendriks, 2006), serves as a mathematical enabler of model checking and as theoretical background for providing correctness and termination arguments for the implemented algorithms.

The same phenomenon holds true for probabilistic and stochastic systems; see, for instance, (Garavel and Hermanns, 2002; Hillston, 1996; Larsen and Skou, 1991). In the setting of stochastic processes, notions of Markovian bisimilarity (Hillston, 1996) play a key role since they are consistent with the notion of *lumping* over continuous-time Markov chains (Buchholz, 1994). This means that, whenever two processes are Markovian bisimilar, they are guaranteed to possess the same performance characteristics.

0.5.2 Concluding remarks

Overall, apart from its very satisfying mathematical properties and its strong connections with logics and games, bisimilarity is computationally superior to the other behavioural semantics that have been considered in the literature on concurrency theory. Its algorithmic, complexity and decidability properties that have been surveyed in this chapter are, to our mind, truly remarkable. As far as its suitability for and actual uses in verification are concerned, we hope that we have been able to provide our readers with enough information to draw their own conclusions. We feel confident that the debate on the merits of bisimilarity and related notions will continue for some time amongst concurrency theorists and members of the computer-aided-verification community. Perhaps, this is a sign of the fact that, despite divergences of opinions, the notion of bisimilarity is considered important, or at least interesting, by researchers in the aforementioned communities and that it has become part of our cultural heritage. Why would it be worth discussing otherwise?

Acknowledgements. We thank Rance Cleaveland, Hubert Garavel and Jan Friso Groote for their contributions to the development of Section 0.5. Without their expert and thought-provoking opinions that section would have been much less informative and rather uninteresting, if not plain boring. We are also indebted to Arnar Birgisson for his detailed reading of, and his comments on, parts of the chapter. Luca Aceto and Anna Ingolfsdottir were partly supported by the projects 'The Equational Logic of Parallel Processes' (nr. 060013021) and 'New Developments in Operational Semantics' (nr. 080039021) of The Icelandic Research Fund. Jiří Srba acknowledges a partial support from the Ministry of Education of the Czech Republic, grant no. MSM0021622419.

Bibliography

Abadi, M. and Lamport, L. (1991). The existence of refinement mappings, *Theoretical Computer Science* **82**(2): 253–284.

Aceto, L., Fokkink, W., Glabbeek, R. v. and Ingolfsdottir, A. (2004). Nested semantics over finite trees are equationally hard, *Information and Computation* **191**(2): 203–232.

Aceto, L., Fokkink, W. and Ingolfsdottir, A. (2001). 2-nested simulation is not finitely equationally axiomatizable, *Proceedings of the 18th International Symposium on Theoretical Aspects of Computer Science, STACS 2001 (Dresden)*, Vol. 2010 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 39–50.

Aceto, L., Ingolfsdottir, A., Larsen, K. G. and Srba, J. (2007). *Reactive Systems: Modelling, Specification and Verification*, Cambridge University Press.

Aho, A., Hopcroft, J. and Ullman, J. (1974). *The design and analysis of computer algorithms*, Addison-Wesley.

Alur, R. and Dill, D. L. (1994). A theory of timed automata, *Theoretical Computer Science* **126**(2): 183–235. Fundamental Study.

Alur, R., Henzinger, T. A., Lafferriere, G. and Pappas, G. J. (2000). Discrete abstractions of hybrid systems, *Proceedings of the IEEE*, Vol. 88, pp. 971–984.

Baeten, J. (ed.) (1990). *Applications of Process Algebra*, Cambridge Tracts in Theoretical Computer Science 17, Cambridge University Press.

Baeten, J. and Weijland, P. (1990). *Process Algebra*, Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press.

Balcazar, J., Gabarro, J. and Santha, M. (1992). Deciding bisimilarity is P-complete, *Formal Aspects of Computing* **4**: 638–648.

Bar-Hillel, Y., Perles, M. and Shamir, E. (1961). On formal properties of simple phrase structure grammars, *Zeitschrift für Phonetik, Sprachwissenschaft, und Kommunikationsforschung* **14**: 143–177.

Behrmann, G., David, A., Larsen, K. G., Håkansson, J., Pettersson, P., Yi, W. and Hendriks, M. (2006). UPPAAL 4.0, *Third International Conference on the Quantitative Evaluaiton of Systems (QEST 2006), 11–14 September 2006, Riverside, California, USA*, IEEE Computer Society, pp. 125–126.

Bergstra, J. and Klop, J. (1985). Algebra of communicating processes with abstraction, *Theoretical Computer Science* **37**: 77–121.

Bergstra, J. and Klop, J. W. (1984). Process algebra for synchronous communication, *Information and Control* **60**(1/3): 109–137.

Bergstra, J., Ponse, A. and Smolka, S. A. (eds) (2001). *Handbook of Process Algebra*, Elsevier.

Bloom, B. (1988). Constructing two-writer registers, *IEEE Transactions on Computers* **37**(12): 1506–1514.

Bloom, B., Istrail, S. and Meyer, A. (1995). Bisimulation can't be traced, *Journal of the ACM* **42**(1): 232–268.

Bloom, B. and Paige, R. (1995). Transformational design and implementation of a new efficient solution to the ready simulation problem, *Science of Computer Programming* **24**(3): 189–220.

Bouajjani, A., Fernandez, J.-C., Graf, S., Rodriguez, C. and Sifakis, J. (1991). Safety for branching time semantics, *in* J. Leach Albert, B. Monien and M. Rodríguez (eds), *Proceedings* 18^{th} *ICALP,* Madrid, Vol. 510 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 76–92.

Bouajjani, A., Fernandez, J.-C. and Halbwachs, N. (1991). Minimal model generation, *in* E. Clarke and R. Kurshan (eds), *Proceedings of the 2nd International Conference on Computer-Aided Verification,* New Brunswick, NJ, USA June 1990, Vol. 531 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 197–203.

Bouali, A. and Simone, R. d. (1992). Symbolic bisimulation minimisation, *Proceedings of CAV'92*, Vol. 663 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 96–108.

Boudol, G., Roy, V., de Simone, R. and Vergamini, D. (1989). Process calculi, from theory to practice: Verification tools, *in* J. Sifakis (ed.), *Automatic Verification Methods for Finite State Systems*, Vol. 407 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 1–10.

Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S. and Yovine, S. (1998). Kronos: A model-checking tool for real-time systems, *in* A. J. Hu and M. Y. Vardi (eds), *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, Vol. 1427 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 546–550.

Bradfield, J., Esparza, J. and Mader, A. (1996). An effective tableau system for the linear time μ-calculus, *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming (ICALP'96)*, Vol. 1099 of *LNCS*, Springer-Verlag, pp. 98–109.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation, *IEEE Transactions on Computers* **C-35**(6): 677–691.

Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams, *ACM Computing Surveys* **24**(3): 293–318.

Buchholz, P. (1994). Exact and ordinary lumpability in finite Markov chains, *Journal of Applied Probability* **31**(1): 59–75.

Burkart, O., Caucal, D., Moller, F. and Steffen, B. (2001). Verification on infinite structures, *in* J. Bergstra, A. Ponse and S. Smolka (eds), *Handbook of Process Algebra*, Elsevier Science, chapter 9, pp. 545–623.

Burkart, O., Caucal, D. and Steffen, B. (1995). An elementary decision procedure for arbitrary context-free processes, *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, Vol. 969 of *LNCS*,

Springer-Verlag, pp. 423–433.

Burkart, O. and Esparza, J. (1997). More infinite results, *Bulletin of the European Association for Theoretical Computer Science* **62**: 138–159. Columns: Concurrency.

Caucal, D. (1992). On the regular structure of prefix rewriting, *Theoretical Computer Science* **106**(1): 61–86.

Christensen, S. (1993). *Decidability and Decomposition in Process Algebras*, PhD thesis, The University of Edinburgh.

Christensen, S., Hirshfeld, Y. and Moller, F. (1993). Bisimulation is decidable for basic parallel processes, *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, Vol. 715 of *LNCS*, Springer-Verlag, pp. 143–157.

Christensen, S., Hüttel, H. and Stirling, C. (1995). A polynomial algorithm for deciding bisimilarity of normed context-free processes, *Information and Computation* **12**(2): 143–148.

Clarke, E. and Emerson, E. (1981). Design and synthesis of synchronization skeletons using branching-time temporal logic, *in* D. Kozen (ed.), *Proceedings of the Workshop on Logic of Programs,* Yorktown Heights, Vol. 131 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 52–71.

Clarke, E. M., Grumberg, O. and Peled, D. A. (2000). *Model Checking*, MIT Press.

Cleaveland, R. (1990). Tableau-based model checking in the propositional mu-calculus, *Acta Informatica* **27**(8): 725–747.

Cleaveland, R. (2009). Personal communication.

Cleaveland, R. and Hennessy, M. (1993). Testing equivalence as a bisimulation equivalence, *Formal Aspects of Computing* **5**(1): 1–20.

Cleaveland, R., Parrow, J. and Steffen, B. (1993). The concurrency workbench: A semantics-based verification tool for finite state systems, *ACM Transactions on Programming Languages and Systems* **15**(1): 36–72.

Cleaveland, R. and Sims, S. (1996). The NCSU Concurrency Workbench, *in* R. Alur and T. Henzinger (eds), *Proceedings of the 8th International Conference Computer Aided Verification,* New Brunswick, NJ, U.S.A., July/August 1996, Vol. 1102 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 394–397.

Cleaveland, R. and Sokolsky, O. (2001). Equivalence and preorder checking for finite-state systems, *in* Bergstra, Ponse and Smolka (2001), chapter 6, pp. 391–424.

Concurrency mailing list archive (1988–1990). Available on line at http://homepages.cwi.nl/∼bertl/concurrency/.

Dam, M. (1994). CTL* and ECTL* as fragments of the modal μ-calculus, *Theoretical Computer Science* **126**: 77–96.

De Nicola, R. and Hennessy, M. (1984). Testing equivalences for processes, *Theoretical Computer Science* **34**: 83–133.

Dickson, L. (1913). Finiteness of the odd perfect and primitive abundant numbers with distinct factors, *American Journal of Mathematics* **35**: 413–422.

Enders, R., Filkorn, T. and Taubner, D. (1993). Generating BDDs for symbolic model checking in CCS, *Distributed Computing* **6**(3): 155–164.

Esparza, J. (2002). Grammars as processes, *in* W. Brauer, H. Ehrig, J. Karhumäki and A. Salomaa (eds), *Formal and Natural Computing*, Vol. 2300 of *LNCS*, Springer-Verlag, pp. 277–297.

Esparza, J. and Knoop, J. (1999). An automata-theoretic approach to interprocedural

data-flow analysis, *Proceedings of the 2nd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'99)*, Vol. 1578 of *LNCS*, Springer-Verlag, pp. 14–30.

Fisler, K. and Vardi, M. Y. (1998). Bisimulation minimization in an automata-theoretic verification framework, *in* G. Gopalakrishnan and P. J. Windley (eds), *Formal Methods in Computer-Aided Design, Second International Conference, FMCAD '98, Palo Alto, California, USA, November 4–6, 1998, Proceedings*, Vol. 1522 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 115–132.

Fisler, K. and Vardi, M. Y. (2002). Bisimulation minimization and symbolic model checking, *Formal Methods in System Design* **21**(1): 39–78.

Formal Systems (Europe) Ltd. (2000). *Failures-Divergence Refinement—FRD2 User Manual*.

Garavel, H. (2009). Personal communication.

Garavel, H. and Hermanns, H. (2002). On combining functional verification and performance evaluation using CADP, *in* L.-H. Eriksson and P. A. Lindsay (eds), *FME 2002: Formal Methods - Getting IT Right, International Symposium of Formal Methods Europe, Copenhagen, Denmark, July 22–24, 2002, Proceedings*, Vol. 2391 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 410–429.

Garavel, H., Lang, F., Mateescu, R. and Serwe, W. (2007). Cadp 2006: A toolbox for the construction and analysis of distributed processes, *in* W. Damm and H. Hermanns (eds), *Proceedings of the 19th International Conference on Computer Aided Verification CAV'2007 (Berlin, Germany)*, Vol. 4590 of *Lecture Notes in Computer Science*, Springer-Verlag.

Garey, M. R. and Johnson, D. S. (1979). *Computers and intractability*, W. H. Freeman and Co., San Francisco, Calif. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

Glabbeek, R. v. (1993). The linear time – branching time spectrum II: the semantics of sequential processes with silent moves, *in* E. Best (ed.), *Proceedings CONCUR 93,* Hildesheim, Germany, Vol. 715 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 66–81.

Glabbeek, R. v. (1994). What is branching time semantics and why to use it?, *Bulletin of the EATCS* **53**: 191–198.

Glabbeek, R. v. (2001). The linear time–branching time spectrum. I. The semantics of concrete, sequential processes, *in* Bergstra et al. (2001), pp. 3–99.

Glabbeek, R. v. and Ploeger, B. (2008). Correcting a space-efficient simulation algorithm, *in* A. Gupta and S. Malik (eds), *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7–14, 2008, Proceedings*, Vol. 5123 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 517–529.

Glabbeek, R. v. and Weijland, W. (1996). Branching time and abstraction in bisimulation semantics, *Journal of the ACM* **43**(3): 555–600.

Greenlaw, R., Hoover, H. J. and Ruzzo, W. R. (1995). *Limits to Parallel Computation: P-Completeness Theory*, Oxford University Press.

Groote, J. F. (2009). Personal communication.

Groote, J. F. and Hüttel, H. (1994). Undecidable equivalences for basic process algebra, *Information and Computation* **115**(2): 353–371.

Groote, J. F., Keiren, J., Mathijssen, A., Ploeger, B., Stappers, F., Tankink, C., Usenko,

Y., van Weerdenburg, M., Wesselink, W., Willemse, T. and van der Wulp, J. (2008). The mCRL2 toolset, *Proceedings International Workshop on Advanced Software Development Tools and Techniques (WASDeTT 2008)*.

Groote, J. F. and Vaandrager, F. (1990). An efficient algorithm for branching bisimulation and stuttering equivalence, *in* M. Paterson (ed.), *Proceedings* 17^{th} *ICALP,* Warwick, Vol. 443 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 626–638.

Groote, J. F. and Willemse, T. A. C. (2005). Parameterised boolean equation systems, *Theoretical Computer Science* **343**(3): 332–369.

Harel, D., Kupferman, O. and Vardi, M. (1997). On the complexity of verifying concurrent transition systems, *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'07)*, Vol. 1243 of *LNCS*, Springer-Verlag, pp. 258–272.

Hennessy, M. and Milner, R. (1985). Algebraic laws for nondeterminism and concurrency, *Journal of the ACM* **32**(1): 137–161.

Henzinger, T. A., Ho, P.-H. and Wong-Toi, H. (1997). HYTECH: A model checker for hybrid systems, *in* O. Grumberg (ed.), *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, Vol. 1254 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 460–463.

Hillston, J. (1996). *A Compositional Approach to Performance Modelling*, Cambridge University Press.

Hirshfeld, Y. (1994a). Congruences in commutative semigroups, *Technical report ECS-LFCS-94-291*, Department of Computer Science, University of Edinburg.

Hirshfeld, Y. (1994b). Petri nets and the equivalence problem, *Proceedings of the 7th Workshop on Computer Science Logic (CSL'93)*, Vol. 832 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 165–174.

Hirshfeld, Y. and Jerrum, M. (1999). Bisimulation equivalence is decidable for normed process algebra, *Proceedings of 26th International Colloquium on Automata, Languages and Programming (ICALP'99)*, Vol. 1644 of *LNCS*, Springer-Verlag, pp. 412–421.

Hoare, C. (1985). *Communicating Sequential Processes*, Prentice-Hall International, Englewood Cliffs.

Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley.

Hunt, H. B., Rosenkrantz, D. J. and Szymanski, T. G. (1976). On the equivalence, containment, and covering problems for the regular and context-free languages, *Journal of Computer and System Sciences* **12**: 222–268.

Hüttel, H. (1994). Undecidable equivalences for basic parallel processes, *Proceedings of the 2nd International Symposium on Theoretical Aspects of Computer Software (TACS'94)*, Vol. 789 of *LNCS*, Springer-Verlag, pp. 454–464.

Hüttel, H. and Stirling, C. (1998). Actions speak louder than words: Proving bisimilarity for context-free processes, *Journal of Logic and Computation* **8**(4): 485–509.

Huynh, D. and Tian, L. (1995). On deciding readiness and failure equivalences for processes in Σ_2^P, *Information and Computation* **117**(2): 193–205.

Jančar, P. (1995). Undecidability of bisimilarity for Petri nets and some related problems, *Theoretical Computer Science* **148**(2): 281–301.

Jančar, P. (2003). Strong bisimilarity on basic parallel processes is PSPACE-complete,

Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03), IEEE Computer Society Press, pp. 218–227.

Jančar, P. (2008). Selected ideas used for decidability and undecidability of bisimilarity, *Proceedings of the 12th International Conference on Developments in Language Theory (DLT'08)*, Vol. 5257 of *LNCS*, Springer-Verlag, pp. 56–71.

Jančar, P. and Srba, J. (2008). Undecidability of bisimilarity by defender's forcing, *Journal of the ACM* **55**(1): 1–26.

Jategaonkar, L. and Meyer, A. (1996). Deciding true concurrency equivalences on safe, finite nets, *Theoretical Computer Science* **154**(1): 107–143.

Kanellakis, P. C. and Smolka, S. A. (1983). CCS expressions, finite state processes, and three problems of equivalence, *Proceedings of the 2nd Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'83)*, ACM, pp. 228–240.

Kanellakis, P. C. and Smolka, S. A. (1990). CCS expressions, finite state processes, and three problems of equivalence, *Information and Computation* **86**(1): 43–68.

Keller, R. (1976). Formal verification of parallel programs, *Communications of the ACM* **19**(7): 371–384.

Kozen, D. (1983). Results on the propositional mu-calculus, *Theoretical Computer Science* **27**: 333–354.

Křetínský, M., Řehák, V. and Strejček, J. (2005). Reachability of Hennessy–Milner properties for weakly extended PRS, *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, Vol. 3821 of *LNCS*, Springer-Verlag, pp. 213–224.

Kučera, A. and Jančar, P. (2006). Equivalence-checking on infinite-state systems: Techniques and results, *Theory and Practice of Logic Programming* **6(3)**: 227–264.

Laroussinie, F. and Schnoebelen, P. (2000). The state explosion problem from trace to bisimulation equivalence, *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'00)*, Vol. 1784 of *LNCS*, Springer-Verlag, pp. 192–207.

Larsen, K. G. (1986). *Context-dependent bisimulation between processes*, PhD thesis, Department of Computer Science, University of Edinburgh.

Larsen, K. G. and Milner, R. (1992). A compositional protocol verification using relativized bisimulation, *Information and Computation* **99**(1): 80–108.

Larsen, K. G. and Skou, A. (1991). Bisimulation through probabilistic testing, *Information and Computation* **94**(1): 1–28.

Lee, D. and Yannakakis, M. (1992). Online minimization of transition systems (extended abstract), *Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing*, Victoria, British Columbia, Canada, pp. 264–274.

Lichtenstein, O. and Pnueli, A. (1985). Checking that finite state concurrent programs satisfy their linear specification, *Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, New Orleans, Louisiana, pp. 97–107.

Manna, Z. and Pnueli, A. (1989). The anchored version of the temporal framework, *in* J. de Bakker, W. d. Roever and G. Rozenberg (eds), *REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency,* Noordwijkerhout, The Netherlands, May/June 1988, Vol. 354 of *Lecture Notes*

in Computer Science, Springer-Verlag, pp. 201–284.

Mayr, R. (1997a). Combining Petri nets and PA-processes, *Proceedings of the 3rd International Symposium on Theoretical Aspects of Computer Software (TACS'97)*, Vol. 1281 of *LNCS*, Springer-Verlag, pp. 547–561.

Mayr, R. (1997b). Tableau methods for PA-processes, *Proceedings of International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'97)*, Vol. 1227 of *LNCS*, Springer-Verlag, pp. 276–290.

Mayr, R. (1998). *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*, PhD thesis, TU-München.

Mayr, R. (2000). Process rewrite systems, *Information and Computation* **156(1)**: 264–286.

McMillan, K. (1993). *Symbolic Model Checking*, Kluwer Academic Publishers.

Milner, R. (1980). *A Calculus of Communicating Systems*, Vol. 92 of *Lecture Notes in Computer Science*, Springer-Verlag.

Milner, R. (1989). *Communication and Concurrency*, Prentice-Hall International, Englewood Cliffs.

Minsky, M. (1967). *Computation: Finite and Infinite Machines*, Prentice-Hall International.

Moller, F. (1996). Infinite results, *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, Vol. 1119 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 195–216.

Moller, F., Smolka, S. and Srba, J. (2004). On the computational complexity of bisimulation, redux, *Information and Computation* **192**(2): 129–143.

Nain, S. and Vardi, M. Y. (2007). Branching vs. linear time: Semantical perspective, *in* K. S. Namjoshi, T. Yoneda, T. Higashino and Y. Okamura (eds), *Automated Technology for Verification and Analysis, 5th International Symposium, ATVA 2007, Tokyo, Japan, October 22–25, 2007, Proceedings*, Vol. 4762 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 19–34.

Paige, R. and Tarjan, R. E. (1987). Three partition refinement algorithms, *SIAM Journal on Computing* **16**(6): 973–989.

Papadimitriou, C. (1994). *Computational Complexity*, Addison-Wesley, New York.

Peterson, J. (1981). *Petri Net Theory and the Modelling of Systems*, Prentice-Hall.

Petri, C. (1962). *Kommunikation mit Automaten*, PhD thesis, Darmstadt.

Pnueli, A. (1977). The temporal logic of programs, *Proceedings 18th Annual Symposium on Foundations of Computer Science*, IEEE, pp. 46–57.

Rabinovich, A. M. (1997). Complexity of equivalence problems for concurrent systems of finite agents, *Information and Computation* **139**(2): 111–129.

Ray, A. and Cleaveland, R. (2004). Unit verification: the cara experience, *Software Tools for Technology Transfer* **5**(4): 351–369.

Redei, L. (1965). *The Theory of Finitely Generated Commutative Semigroups*, Oxford University Press, NY.

Rounds, W. and Brookes, S. (1981). Possible futures, acceptances, refusals and communicating processes, 22nd *Annual Symposium on Foundations of Computer Science,* Nashville, Tennessee, IEEE, New York, pp. 140–149.

Sawa, Z. (2003). Equivalence checking of non-flat systems is EXPTIME-hard, *Proceedings of the 14th International Conference on Concurrency Theorem (CONCUR'03)*, Vol.

2761 of *LNCS*, Springer-Verlag, pp. 233–248.

Sawa, Z. and Jančar, P. (2001). P-hardness of equivalence testing on finite-state processes, *Proceedings of the 28th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'01)*, Vol. 2234 of *LNCS*, Springer-Verlag, pp. 326–345.

Senizergues, G. (1998). Decidability of bisimulation equivalence for equational graphs of finite out-degree, *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, IEEE, pp. 120–129.

Sipser, M. (2006). *Introduction to the Theory of Computation*, Course Technology.

Srba, J. (2001). On the power of labels in transition systems, *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'01)*, Vol. 2154 of *LNCS*, Springer-Verlag, pp. 277–291.

Srba, J. (2002). Note on the tableau technique for commutative transition systems, *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'02)*, Vol. 2303 of *LNCS*, Springer-Verlag, pp. 387–401.

Srba, J. (2003). Strong bisimilarity of simple process algebras: Complexity lower bounds, *Acta Informatica* **39**: 469–499.

Srba, J. (2004). *Roadmap of Infinite results*, Vol. 2: Formal Models and Semantics, World Scientific Publishing Co. An updated version can be downloaded from the author's home-page.

Stirling, C. (1995). Local model checking games, *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, Vol. 962 of *LNCS*, Springer-Verlag, pp. 1–11.

Stirling, C. (1998). Decidability of bisimulation equivalence for normed pushdown processes, *Theoretical Computer Science* **195**(2): 113–131.

Stirling, C. (2000). Decidability of bisimulation equivalence for pushdown processes. Submitted for publication. Available from the author's homepage.

Stirling, C. (2001). Decidability of weak bisimilarity for a subset of basic parallel processes, *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'01)*, Vol. 2030 of *LNCS*, Springer-Verlag, pp. 379–393.

Stirling, C. (2003). Bisimulation and language equivalence, *Logic for concurrency and synchronisation*, Vol. 18 of *Trends Log. Stud. Log. Libr.*, Kluwer Acad. Publ., Dordrecht, pp. 269–284.

Stirling, C. and Walker, D. (1991). Local model checking in the modal mu-calculus, *Theoretical Computer Science* **89**(1): 161–177.

Stockmeyer, L. J. and Meyer, A. R. (1973). Word problems requiring exponential time, *Proceedings 5th ACM Symposium on Theory of Computing, Austin, Texas*, pp. 1–9.

Tan, L. and Cleaveland, R. (2001). Simulation revisited, *in* T. Margaria and W. Yi (eds), *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2–6, 2001, Proceedings*, Vol. 2031 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 480–495.

Thomas, W. (1993). On the Ehrenfeucht-Fraïssé game in theoretical computer science (extended abstract), *Proceedings of the 4th International Joint Conference CAAP/FASE, Theory and Practice of Software Development (TAPSOFT'93)*, Vol.

668 of *LNCS*, Springer-Verlag, pp. 559–568.

Vaandrager, F. (1990). Some observations on redundancy in a context, *in* Baeten (1990), pp. 237–260.