Why is Simulation Harder Than Bisimulation?

Antonín Kučera*[1] and Richard Mayr[2]

[1] Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic,
`tony@fi.muni.cz`
[2] Department of Computer Science, Albert-Ludwigs-University Freiburg
Georges-Koehler-Allee 51, D-79110 Freiburg, Germany.
`mayrri@informatik.uni-freiburg.de`

Abstract. Why is deciding simulation preorder (and simulation equivalence) computationally harder than deciding bisimulation equivalence on almost all known classes of processes? We try to answer this question by describing two general methods that can be used to construct direct one-to-one polynomial-time reductions from bisimulation equivalence to simulation preorder (and simulation equivalence). These methods can be applied to many classes of finitely generated transition systems, provided that they satisfy certain abstractly formulated conditions. Roughly speaking, our first method works for all classes of systems that can test for 'non-enabledness' of actions, while our second method works for all classes of systems that are closed under synchronization.

1 Introduction

In the last decade, a lot of research effort has been devoted to the study of decidability/complexity issues for checking various semantic equivalences between certain (classes of) processes. Formally, a *process* is (associated with) a state in a *transition system*.

Definition 1. *A* transition system *is a triple $\mathcal{T} = (T, Act, \rightarrow)$ where T is a set of* states, *Act is a finite set of* actions, *and $\rightarrow \subseteq T \times Act \times T$ is a* transition relation.

We write $t \xrightarrow{a} \bar{t}$ instead of $(t, a, \bar{t}) \in \rightarrow$ and we extend this notation to elements of Act^* in the natural way. A state $\bar{t}$ is *reachable* from a state t, written $t \rightarrow^* \bar{t}$, if $t \xrightarrow{w} \bar{t}$ for some $w \in Act^*$. A state $\bar{t}$ is an *a-successor* of a state t if $t \xrightarrow{a} \bar{t}$. The set of all a-successors of t is denoted by $succ(t, a)$. Assuming some implicit linear ordering on $succ(t, a)$, we denote by $t(a, j)$ the j^{th} a-successor of t for each $1 \leq j \leq |succ(t, a)|$. The *branching degree* of $\mathcal{T}$, denoted $d(\mathcal{T})$, is the least number n such that $n \geq |\bigcup_{a \in Act} succ(t, a)|$ for every $t \in T$. If there is no such n then $d(\mathcal{T}) = \infty$.

The notion of 'behavioral sameness' of two processes can be formally captured in many different ways (see, e.g., [vG99] for an overview). Among those behavioral equivalences, *simulation* and *bisimulation* equivalence enjoy special attention.

Definition 2. *Let $\mathcal{S} = (S, Act, \rightarrow)$ and $\mathcal{T} = (T, Act, \rightarrow)$ be transition systems. A relation $\mathcal{R} \subseteq S \times T$ is a* simulation *iff whenever $(s, t) \in \mathcal{R}$, then*

* Supported by the Grant Agency of the Czech Republic, grant No. 201/00/0400.

– *for each $s \xrightarrow{a} \bar{s}$ there is some $t \xrightarrow{a} \bar{t}$ such that $(\bar{s}, \bar{t}) \in \mathcal{R}$.*

A process s is simulated *by t, written $s \sqsubseteq t$, iff there is a simulation $\mathcal{R}$ such that $(s, t) \in \mathcal{R}$. Processes s, t are* simulation equivalent, *written $s \simeq t$, iff they can simulate each other.*

A bisimulation is a simulation whose inverse is also a simulation; a more explicit definition follows.

Definition 3. *Let $\mathcal{S} = (S, Act, \rightarrow)$ and $\mathcal{T} = (T, Act, \rightarrow)$ be transition systems. A relation $\mathcal{R} \subseteq S \times T$ is a* bisimulation *iff whenever $(s, t) \in \mathcal{R}$, then*

– *for each $s \xrightarrow{a} \bar{s}$ there is some $t \xrightarrow{a} \bar{t}$ such that $(\bar{s}, \bar{t}) \in \mathcal{R}$;*
– *for each $t \xrightarrow{a} \bar{t}$ there is some $s \xrightarrow{a} \bar{s}$ such that $(\bar{s}, \bar{t}) \in \mathcal{R}$.*

Processes s and t are bisimulation equivalent (bisimilar), *written $s \sim t$, iff they are related by some bisimulation.*

Simulations (and bisimulations) can also be viewed as *games* [Sti98, Tho93] between two players, the attacker and the defender. In a simulation game the attacker wants to show that $s \not\sqsubseteq t$, while the defender attempts to frustrate this. Imagine that there are two tokens put on states s and t. Now the two players, attacker and defender, start to play a *simulation game* which consists of a (possibly infinite) number of *rounds* where each round is performed as follows: The attacker takes the token which was put on s originally and moves it along a transition labeled by (some) a; the task of the defender is to move the other token along a transition with the same label. If one player cannot move then the other player wins. The defender wins every infinite game. It can be easily shown that $s \sqsubseteq t$ iff the defender has a winning strategy. The only difference between a simulation game and a *bisimulation game* is that the attacker can *choose* his token at the beginning of every round (the defender has to respond by moving the other token). Again we get that $s \sim t$ iff the defender has a winning strategy.

Let **A**, **B** be classes of processes. The problem whether a given process s of **A** is simulated by a given process t of **B** is denoted by **A** $\sqsubseteq$ **B**. Similarly, the problem if s and t are simulation equivalent (or bisimilar) is denoted by **A** $\simeq$ **B**, (or **A** $\sim$ **B**, respectively).

One reason why simulation preorder/equivalence and bisimilarity found their way to many practical applications is their special computational tractability—both equivalences are decidable in polynomial time for finite-state processes (in fact, they are **P**-complete [BGS92, SJ01]) and remain decidable even for certain classes of infinite-state processes [Mol96]. By contrast, all trace-based equivalences are **PSPACE**-complete for finite-state systems and become undecidable for infinite-state systems. Further evidence is provided by recent results on equivalence-checking between infinite and finite-state systems—see [KM02b] for an overview. Although the formal definitions of simulation and bisimulation are quite similar, one cannot immediately transfer techniques and algorithms developed for one of the two equivalences to the other one. Nevertheless, there *is* some kind of connection between them—for example, in [KM02b] it has been shown that simulation equivalence can be 'reduced' to bisimulation equivalence in the following sense: given a transition system $\mathcal{T}$, one can define a transition system $\mathcal{T}'$ with the same set of states as $\mathcal{T}$ such that for all states s, t we have that s, t are simulation equivalent in $\mathcal{T}$ iff s, t are bisimilar in $\mathcal{T}'$. Although this 're-duction' works for arbitrary finitely-branching transition systems, it is not effective in general (the only known class of infinite-state processes where the method is effective is the class of one-counter nets [JKM00]; for finite-state processes, the method is even efficient, i.e., polynomial-time). Actually, our present knowledge indicates that there cannot be any general efficient reduction from simulation equivalence to bisimilarity, because all of the existing results confirm a general rule saying that

"*simulation is computationally harder than bisimilarity.*"

Indeed, bisimilarity tends to be 'more decidable' and 'more tractable' than simulation; to the best of our knowledge, there is (so far) no counterexample violating this 'rule of thumb' (maybe except for some artificial constructions). But why is that? One possible answer is that bisimilarity is so much finer than simulation equivalence that it has 'more structure' and becomes easier to decide. However, this is a rather vague statement. In this paper we try to provide a more convincing explanation/justification for the afore-mentioned rule. We show that there are possibilities how to 'transfer' winning strategies for both players from a bisimulation game to a simulation game. More precisely, given two states s and t in transition systems $\mathcal{S}$ and $\mathcal{T}$, we show how to construct states s' and t' in transition systems $\mathcal{S}'$ and $\mathcal{T}'$ in such a way that $s \sim t$ iff $s' \sqsubseteq t'$ (or $s' \simeq t'$). We propose two methods how to achieve that. The point is that both methods are appli-cable to certain (quite large) classes of models of concurrent systems where they result in *effective* and *polynomial-time* reductions. In fact, we formulate abstract conditions on process classes **A** and **B** under which the problem **A** $\sim$ **B** is polynomially reducible to the problem **A** $\sqsubseteq$ **B** (or **A** $\simeq$ **B**). Roughly speaking, the first method (introduced in Section 2) applies to models with a finite branching degree where one can in some sense identify what actions are not enabled. Examples include (subclasses of) pushdown sys-tems, queue systems, etc. The second method (Section 3) is applicable to models closed under (synchronized) parallel composition. Examples are, e.g., Petri nets with various extensions/restrictions. The applicability and limitations of each method are discussed in greater detail in the respective sections.

Although the two methods do not cover all of the existing models of concurrent processes (see Section 4), they reveal a kind of general relationship between winning strategies in bisimulation and simulation games. Moreover, it is now clear that sim-ulation is harder than bisimulation not by coincidence but due to some fundamental reason—one can polynomially reduce bisimilarity to simulation preorder/equivalence by general reductions whose underlying principle is independent of concrete process models. Hence, the paper presents a new piece of generic knowledge rather than a col-lection of technical results.

2 Reducing Bisimilarity to Simulation – Method 1

For the rest of this section, we fix a finite set of actions $Act = \{a_1, \ldots, a_k\}$ and two transition systems $\mathcal{S} = (S, Act, \to)$, $\mathcal{T} = (T, Act, \to)$ with a finite branching degree. Moreover, we put $d = \max\{d(\mathcal{S}), d(\mathcal{T})\}$. Our aim is to define two other transition systems $\mathcal{S}'$ and $\mathcal{T}'$ which extend the systems $\mathcal{S}$ and $\mathcal{T}$ by some new states and transitions in such a way that for all $s \in S$, $t \in T$ we have that $s \sim t$ iff $s' \sqsubseteq t'$, where s' and t' are the 'twins' of s and t in $\mathcal{S}'$ and $\mathcal{T}'$, respectively. Since, in the considered simulation game, the attacker plays within $\mathcal{S}'$ and the defender plays within $\mathcal{T}'$, we call $\mathcal{S}'$ an *A-extension* of $\mathcal{S}$, and $\mathcal{T}'$ a *D-extension* of $\mathcal{T}$. The idea behind the construction of $\mathcal{S}'$ and $\mathcal{T}'$ can be intuitively described as follows: one round of a bisimulation game between s and t is emulated by at most two rounds of a simulation game between s' and t'. The rules of the bisimulation game allow the attacker to make his move either from s or from t . If he chooses s and plays $s \xrightarrow{a_i} s(a_i, j)$, we can easily emulate his attack by $s' \xrightarrow{a_i} s(a_i, j)'$ (remember that $s(a_i, j)$ is the j^{th} a_i-successor of s). The defender's response $t \xrightarrow{a_i} t(a_i, \ell)$ is emulated by $t' \xrightarrow{a_i} t(a_i, \ell)'$ (t' has the 'same' a_i-successors as t). If the attacker chooses t and plays $t \xrightarrow{a_i} t(a_i, j)$, the emulation is more complicated and takes two rounds. First, to each successor $t(a_i, j)$ of t we associate a unique action λ_i^j. Now, we add to s' a family of transitions $s' \xrightarrow{\lambda_i^j} s'$ for all $1 \le i \le k$ and $1 \le j \le d$. To emulate the move $t \xrightarrow{a_i} t(a_i, j)$ in our simulation game, the attacker performs the λ_i^j-loop on s'. In our bisimulation game, the attack $t \xrightarrow{a_i} t(a_i, j)$ would be matched by (some) move $s \xrightarrow{a_i} s(a_i, \ell)$. First, we name the successors of s by a family of δ_i^j actions. Now, the response $s \xrightarrow{a_i} s(a_i, \ell)$ is emulated by $t' \xrightarrow{\lambda_i^j} r[a_i, j, \ell]$, where $r[a_i, j, \ell]$ is a newly added state. It finishes the first round, i.e., the first emulation phase. In the second round, t' is in a way *forced* to go to $t(a_i, j)'$ and the only response available to $r[a_i, j, \ell]$ is to enter $s(a_i, \ell)'$, which finishes the emulation. The mentioned 'forcing' is achieved by allowing $r[a_i, j, \ell]$ to go to a 'universal' state (i.e., to a state which can simulate everything) under all but one action δ_i^ℓ. It means that if any other action (different from δ_i^ℓ) is used by t' in the second round, the attacker loses the simulation game. Hence, his only chance is to play a δ_i^ℓ-transition; our construction ensures that the only such transition is $t' \xrightarrow{\delta_i^\ell} t(a_i, \ell)'$. Our construction also guarantees that any 'bad' move of one of the two players in the simulation game (which is not consistent with the above given scenario) is immediately 'punished' by allowing the other player to win. Formal definitions and proofs follow.

Let $\Lambda = \{\lambda_i^j \mid 1 \le i \le k, 1 \le j \le d\}$ and $\Delta = \{\delta_i^j \mid 1 \le i \le k, 1 \le j \le d\}$ be finite sets of actions such that Λ, Δ, and Act are pairwise disjoint. We define $Act' = Act \cup \Lambda \cup \Delta \cup \{\gamma\}$ where γ is a fresh action.

Definition 4. *An A-*extension *of $\mathcal{S}$ (see Fig.1) is a transition system $\mathcal{S}' = (S', Act', \rightarrow)$ together with an injective mapping $f : S \rightarrow S'$ satisfying the following conditions (where $f(s)$ is abbreviated to s'):*

- *If $s \xrightarrow{a} \bar{s}$ is a transition of $\mathcal{S}$, then $s' \xrightarrow{a} \bar{s}'$ is a transition of $\mathcal{S}'$.*
- *$s' \xrightarrow{\lambda} s'$ is a transition of $\mathcal{S}'$ for every $s \in S$ and $\lambda \in \Lambda$.*
- *For all $s \in S$, $1 \le i \le k$, and $1 \le j \le d$ we have the following:*
  - *if $j \le |succ(s, a_i)|$ then $s' \xrightarrow{\delta_i^j} s(a_i, j)'$ is a transition of $\mathcal{S}'$ (remember that $s(a_i, j)$ is the j^{th} a_i-successor of s in $\mathcal{S}$);*
  - *if $j > |succ(s, a_i)|$ then there is a transition $s' \xrightarrow{\delta_i^j} q_j$ to some state $q_j \in S'$ which can perform the action γ. This family of δ_i^j-transitions is indicated by a*

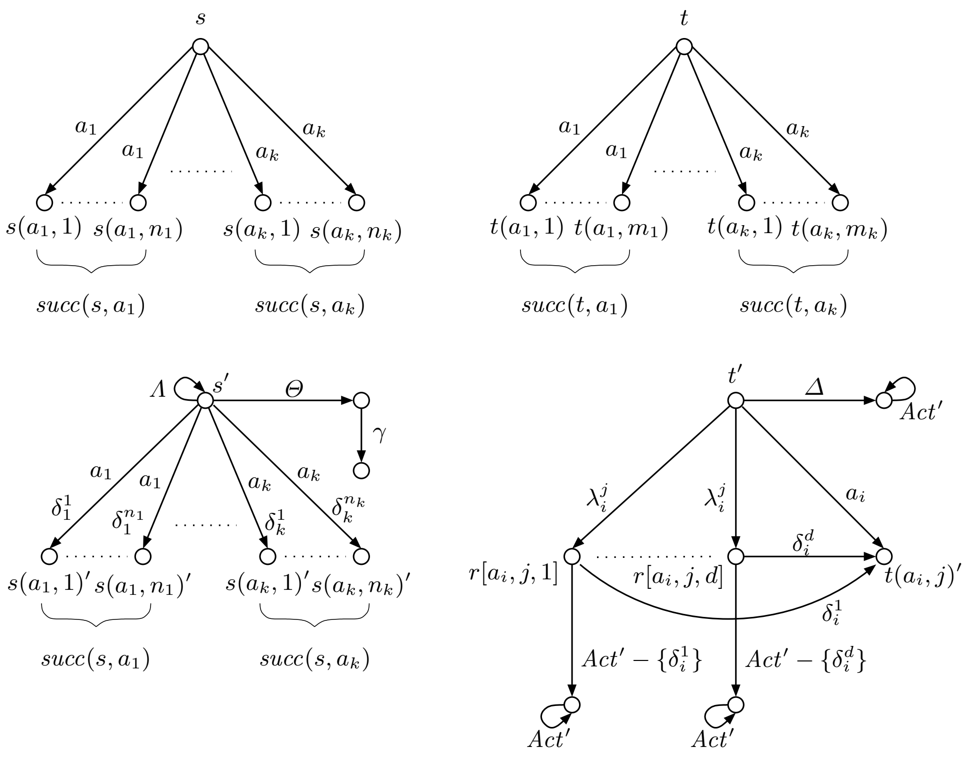


Fig. 1. States s of $\mathcal{S}$ and t of $\mathcal{T}$, and the corresponding states s' of $\mathcal{S}'$ and t' of $\mathcal{T}'$ (some of the out-going transitions of t' are omitted).

Θ-labeled arrow in Fig. 1. Also observe that we do not impose any additional restrictions on q_j (i.e., q_j can also emit other actions).

– *For each $s \in S$ the state s' has only the transitions admitted above.*

A simple observation about A-extensions is

Lemma 5. *If $\mathcal{S}$ is deterministic, then $\mathcal{S}$ has a deterministic A-extension.*

A state u is *universal* if $u \xrightarrow{a} u$ for every $a \in Act'$ (u can also have other out-going transitions). Observe that a universal state can simulate *any* process which emits actions of Act'. In the next definition, we write $t \xrightarrow{a} U$ to indicate that $t \xrightarrow{a} u$ for some universal state u.

Definition 6. *A D-*extension *of $\mathcal{T}$ (see Fig. 1) is a transition system $\mathcal{T}' = (T', Act', \to)$ together with an injective mapping $g : T \to T'$ satisfying the following conditions (where $g(t)$ is abbreviated to t'):*

– *If $t \xrightarrow{a} \bar{t}$ is a transition of $\mathcal{T}$, then $t' \xrightarrow{a} \bar{t}'$ is a transition of $\mathcal{T}'$.*
– *$t' \xrightarrow{\delta} U$ for every $t \in T$ and $\delta \in \Delta$.*
– *For all $t \in T$, $1 \le i \le k$, and $1 \le j \le d$ we have the following:*

  - *If $j \leq |succ(t, a_i)|$ then $t' \xrightarrow{\lambda_i^j} r[a_i, j, \ell]$ is a transition of $\mathcal{S}'$ for each $1 \leq \ell \leq d$. Here $r[a_i, j, \ell]$ is a state of T' which has*
    - *exactly one δ_i^ℓ-transition $r[a_i, j, \ell] \xrightarrow{\delta_i^\ell} t(a_i, j)'$ (remember that $t(a_i, j)$ is the j^{th} a_i-successor of t in $\mathcal{T}$);*
    - *a transition $r[a_i, j, \ell] \xrightarrow{a} U$ for each $a \in Act' - \{\delta_i^\ell\}$.*
  - *If $j > |succ(t, a_i)|$ then there is a transition $t' \xrightarrow{\lambda_i^j} U$.*
- *For each $t \in T$ the state t' has only the transitions admitted above.*

Theorem 7. *Let $s \in S$ and $t \in T$. Let $\mathcal{S}'$ be an A-extension of $\mathcal{S}$ and $\mathcal{T}'$ a D-extension of $\mathcal{T}$. We have that $s \sim t$ iff $s' \sqsubseteq t'$.*

Proof. We start with the '$\Longrightarrow$' direction. Let us suppose that $s \sim t$. We show that the defender has a winning strategy in a simulation game initiated in (s', t'). Let $\mathcal{R} = \{(s', t') \mid s \in S, t \in T, s \sim t\}$. We prove that $\mathcal{R}$ can be extended to a simulation relation (which means that indeed $s' \sqsubseteq t'$). Let $(s', t') \in \mathcal{R}$. We show that the defender can play in such a way that after at most two rounds he achieves either a configuration from $\mathcal{R}$, or a configuration where he 'obviously' wins. So, let $s' \xrightarrow{a} p$ be an attacker's move. We consider three possibilities:

- If $a \in Act$, then $p = \bar{s}'$ for some $\bar{s} \in S$. Since $s \sim t$, there is $t \xrightarrow{a} \bar{t}$ such that $\bar{s} \sim \bar{t}$. Hence, the defender can respond by $t' \xrightarrow{a} \bar{t}'$ and thus enter a configuration from $\mathcal{R}$.
- If $a \in \Delta$, then there is a transition $t' \xrightarrow{a} U$ (cf. Definition 6). Hence, the defender can use this transition and from that point on he can simulate 'everything'.
- If $a = \lambda_i^j$, then $p = s'$ and there are two possibilities:
  - The state t has fewer than j a_i-successors. Then there is a transition $t' \xrightarrow{\lambda_i^j} U$ and the defender wins.
  - Otherwise, let us consider the attack $t \xrightarrow{a_i} t(a_i, j)$ in the *bisimulation* game between s and t. Since $s \sim t$, there is a move $s \xrightarrow{a_i} s(a_i, \ell)$ such that $s(a_i, \ell) \sim t(a_i, j)$. In our simulation game, the defender uses the $t' \xrightarrow{\lambda_i^j} r[a_i, j, \ell]$ transition as a response. The current game situation (after playing one round) is $(s', r[a_i, j, \ell])$. Now, if the attacker plays $s' \xrightarrow{\delta_i^\ell} s(a_i, \ell)'$, the defender can respond by $r[a_i, j, \ell] \xrightarrow{\delta_i^\ell} t(a_i, j)'$ and enter a configuration from $\mathcal{R}$. If the attacker uses any other attack, the defender can go to a universal state and thus he wins.

Now we show the '$\Longleftarrow$' direction, i.e., we assume $s' \sqsubseteq t'$ and prove $s \sim t$. To do that, we demonstrate that $\mathcal{R} = \{(s, t) \mid s \in S, t \in T, s' \sqsubseteq t'\}$ is a bisimulation. So, let $(s, t) \in \mathcal{R}$. An attack which comes from the first component is easy to handle—if $s \xrightarrow{a} \bar{s}$, then also $s' \xrightarrow{a} \bar{s}'$ and since $s' \sqsubseteq t'$, there is $t' \xrightarrow{a} \bar{t}'$ such that $\bar{s}' \sqsubseteq \bar{t}'$. Hence, $t \xrightarrow{a} \bar{t}$ where $(\bar{s}, \bar{t}) \in \mathcal{R}$. Now let us consider an attack $t \xrightarrow{a_i} t(a_i, j)$. To find an appropriate response for the defender, let us examine the *simulation* game between s' and t'. Here, the attacker can play $s' \xrightarrow{\lambda_i^j} s'$. The defender must respond by some $t' \xrightarrow{\lambda_i^j} r[a_i, j, \ell]$ (there is surely no transition $t' \xrightarrow{\lambda_i^j} U$). If the ℓ was greater than the number of a_i-successors of s, the defender's response would be definitely wrong because then the attacker could win in two rounds by performing the transitions $s' \xrightarrow{\delta_i^\ell} q_\ell \xrightarrow{\gamma} q$. So, we see that the ℓ must be less than or equal to the number of a_i-successors of s. The attacker can further play $s' \xrightarrow{\delta_i^\ell} s(a_i, \ell)'$, and the defender can respond only by $r[a_i, j, \ell] \xrightarrow{\delta_i^\ell} t(a_i, j)'$. Thus, we obtain that $s(a_i, \ell)' \sqsubseteq t(a_i, j)'$. It means that, in our bisimulation game, the defender can use the transition $s \xrightarrow{a_i} s(a_i, \ell)$ and enter a configuration from $\mathcal{R}$. □

2.1 Applications

Theorem 7 allows to construct direct one-to-one polynomial-time reductions from the problem **A** $\sim$ **B** to the problem **A** $\sqsubseteq$ **B** (and **A** $\simeq$ **B**) for many process classes **A** and **B**. All we need to show is that the syntax of **A** and **B** admits an efficient construction of A- and D-extensions, respectively. It can be done, e.g., for (various subclasses of) pushdown automata, BPA systems, one-counter automata, queue automata (where the queue can be tested for emptiness), channel systems, 1-safe Petri nets, and others (the list is surely not exhaustive; interested reader can probably add some of his own favorite models). To illustrate this, we discuss the model of pushdown automata in greater detail. The limitations of our first method are mentioned at the end of this section.

A *pushdown automaton (PDA)* is a tuple $\mathcal{M} = (Q, \Gamma, Act, \eta)$ where Q is a finite set of *control states*, Γ is a finite *stack alphabet*, Act is a finite *input alphabet*, and $\eta : (Q \times \Gamma) \rightarrow \mathcal{P}(Act \times (Q \times \Gamma^*))$ is a *transition function* with finite image (here $\mathcal{P}(M)$ denotes the powerset of M). In the rest of this paper we adopt a more intuitive notation, writing $pA \xrightarrow{a} q\beta \in \eta$ instead of $(a, (q, \beta)) \in \eta(p, A)$. To $\mathcal{M}$ we associate the transition system $\mathcal{T}_\mathcal{M}$ where $Q \times \Gamma^*$ is the set of states (we write $p\alpha$ instead of (p, α)), Act is the set of actions, and the transition relation is defined by $pA\alpha \xrightarrow{a} q\beta\alpha$ iff $pA \xrightarrow{a} q\beta \in \eta$. The set of all states of $\mathcal{T}_\mathcal{M}$ is also denoted by *States*$(\mathcal{T}_\mathcal{M})$.

A PDA $\mathcal{M} = (Q, \Gamma, Act, \eta)$ is

- *deterministic* if for all $p \in Q$, $A \in \Gamma$, and $a \in Act$ there is at most one $q\beta \in Q \times \Gamma^*$ such that $pA \xrightarrow{a} q\beta$;
- *normed* if for every $p\alpha \in Q \times \Gamma^*$ there is $q \in Q$ such that $p\alpha \rightarrow^* q\varepsilon$;
- *stateless* if $|Q| = 1$;
- *one-counter automaton* if $\Gamma = \{I, Z\}$ and each element of η is either of the form $pZ \xrightarrow{a} qI^jZ$ where $j \in \mathbb{N}_0$ (such transitions are called *zero-transitions*), or of the form $pI \xrightarrow{a} qI^j$ where $j \in \mathbb{N}_0$ (these transitions are *non-zero-transitions*). Hence, the Z can be viewed as a bottom marker (which cannot be removed), and the number of pushed I's represents the counter value.

The classes of all pushdown processes, stateless pushdown processes, one-counter processes, and finite-state processes are denoted by **PDA**, **BPA**, **OC**, and **FS**, respectively. The normed subclasses of **PDA** and **BPA** are denoted by **nPDA** and **nBPA** (one could also consider normed **OC** processes, but these are not so important). If **A** is any of the so-far defined classes, then **det-A** denotes the subclass of all deterministic processes of **A**. For example, **det-nBPA** is the class of all deterministic normed **BPA** processes. Let

- $\mathcal{D} = \{\textbf{PDA}, \textbf{BPA}, \textbf{OC}, \textbf{nPDA}, \textbf{nBPA}, \textbf{FS}\}$,

– $\mathcal{A} = \mathcal{D} \cup \{\textbf{det-A} \mid \textbf{A} \in \mathcal{D}\}$.

Lemma 8. *Let* $\textbf{\textit{A}} \in \mathcal{A}$ *and let* $\mathcal{M} \in \textbf{\textit{A}}$ *be an automaton of* $\textbf{\textit{A}}$*. Then there is* $\mathcal{M}' \in \textbf{\textit{A}}$ *and a mapping* $f : \mathit{States}(\mathcal{T}_{\mathcal{M}}) \to \mathit{States}(\mathcal{T}_{\mathcal{M}'})$ *constructible in polynomial time (in the size of* $\mathcal{M}$*) such that* $\mathcal{T}_{\mathcal{M}'}$ *together with* f *is an* A*-extension of* $\mathcal{M}$*.*

Proof. We construct $\mathcal{M}'$ by extending $\mathcal{M}$. First, if $\mathcal{M}$ is not a one-counter automaton, it can possibly empty its stack and therefore we add a new 'bottom' symbol Z to the stack alphabet. The mapping f then maps every configuration $p\alpha$ to $p\alpha Z$ (in the case of one-counter automata, f is just identity). The λ_i^j-loops are added by extending the transition function with all rules of the form $pX \xrightarrow{\lambda_i^j} pX$. Since the outgoing transitions of a given state $pX\alpha$ are completely determined by p and X, we can also easily add the δ_i^j-transitions; the family of Θ-transitions (see Fig. 1) is implemented by changing the top stack symbol to a fresh symbol Y, without changing the control state (this works both for PDA and BPA; in the case of one-counter automata we instead change the control to a newly-added control state without modifying the stack). Then, the action γ is emitted and Y is removed from the stack. Note that this construction preserves normedness and determinism. Obviously, the reduction works in polynomial time (and even in logarithmic space). □

The next lemma can be proved in a similar way. Note that the construction does not preserve determinism (see Fig. 1).

Lemma 9. *Let* $\textbf{\textit{D}} \in \mathcal{D}$ *and let* $\mathcal{M} \in \textbf{\textit{D}}$ *be an automaton of* $\textbf{\textit{D}}$*. Then there is* $\mathcal{M}' \in \textbf{\textit{D}}$ *and a mapping* $f : \mathit{States}(\mathcal{T}_{\mathcal{M}}) \to \mathit{States}(\mathcal{T}_{\mathcal{M}'})$ *constructible in polynomial time (in the size of* $\mathcal{M}$*) such that* $\mathcal{T}_{\mathcal{M}'}$ *together with* f *is an* D*-extension of* $\mathcal{M}$*.*

Now, we can formulate two interesting corollaries of Theorem 7.

Corollary 10. *Let* $\textbf{\textit{A}} \in \mathcal{A}$ *and* $\textbf{\textit{D}} \in \mathcal{D}$*. The problem* $\textbf{\textit{A}} \sim \textbf{\textit{D}}$ *is polynomially reducible to the problem* $\textbf{\textit{A}} \sqsubseteq \textbf{\textit{D}}$*.*

Corollary 11. *Let* $\textbf{\textit{A}}, \textbf{\textit{B}} \in \mathcal{D}$ *such that* $\textbf{\textit{B}} \subseteq \textbf{\textit{A}}$*. Then the problem* $\textbf{\textit{A}} \sim \textbf{\textit{B}}$ *is polynomially reducible to the problem* $\textbf{\textit{A}} \simeq \textbf{\textit{B}}$*.*

Proof. There is a general one-to-one reduction from the problem $\textbf{A} \sqsubseteq \textbf{B}$ to the problem $\textbf{A} \simeq \textbf{B}$, which is applicable also in our case—given two processes s and t, we construct other processes s' and t' with transitions $s' \xrightarrow{a} s$, $s' \xrightarrow{a} t$, and $t' \xrightarrow{a} t$. We see that $s \sqsubseteq t$ iff $s' \simeq t'$. □

Our first method is applicable to a wide variety of models, but it has its limitations. For example, in the case of A-extensions there can be difficulties with the family of Θ-transitions. In order to implement them, the model must be able to (somehow) 'detect' the missing transitions. It is not always possible; for example, general Petri nets cannot test a place for emptiness and hence the Θ-transitions cannot be implemented. Nevertheless, the method is applicable to some subclasses/extensions of Petri nets. For example, 1-safe Petri nets *can* in a way test their places for emptiness—to construct an A-extension of a given 1-safe net $\mathcal{N}$, we just equip each place p with its 'twin' $\bar{p}$

and restructure the transitions so that they have the same effect on the 'old' places and preserve the following invariant: $\bar{p}$ is marked iff p is unmarked. It is quite easy; then, we can easily implement the family of Θ-transitions (by testing appropriate 'twins' for being marked). Another example are Petri nets with inhibitor arcs, where our first method applies without any problems.

Hence, we can extended the $\mathcal{A}$ and $\mathcal{D}$ classes by many other models and the obtained corollaries are still valid. In this way one can 'generate' a long list of results, of which some were already known while others are new. Some of these results are 'exotic' (for example, **det-nPDA** $\sim$ **1-PN** is polynomially reducible (and hence not harder than) **det-nPDA** $\sqsubseteq$ **1-PN** where **1-PN** is the class of 1-safe Petri nets; both problems are decidable but their complexity has not yet been analyzed in detail). However, some of the obtained consequences actually improve our knowledge about previously studied problems. For example, **PDA** $\sim$ **FS** is known to be **PSPACE**-hard [May00] while the best known lower bound for **PDA** $\sqsubseteq$ **FS** was **coNP** [KM02b]. Our method allows to improve this lower bound to **PSPACE**[1].

3 Reducing Bisimilarity to Simulation – Method 2

As in the previous section, we first fix a finite set of actions $Act = \{a_1, \ldots, a_k\}$ and two transition systems $\mathcal{S} = (S, Act, \rightarrow)$, $\mathcal{T} = (T, Act, \rightarrow)$ with a finite branching degree. We also define $d = \max\{d(\mathcal{S}), d(\mathcal{T})\}$.

Definition 12. *By a* parallel composition *of transition systems $\mathcal{T}_1 = (T_1, Act, \rightarrow)$ and $\mathcal{T}_2 = (T_2, Act, \rightarrow)$ we mean a transition system $\mathcal{T}_1 \| \mathcal{T}_2 = (T_1 \times T_2, Act, \rightarrow)$ where $(t_1, t_2) \xrightarrow{a} (\bar{t}_1, \bar{t}_2)$ iff either $t_1 \xrightarrow{a} \bar{t}_1$ and $\bar{t}_2 = t_2$, or $t_2 \xrightarrow{a} \bar{t}_2$ and $\bar{t}_1 = t_1$.*

Intuitively, our second method works for all classes of systems that are closed under parallel composition and synchronization (see Definition 17). The idea is as follows: For $\mathcal{S}$ and $\mathcal{T}$ one constructs new systems A-$comp(\mathcal{S}, \mathcal{T})$ and D-$comp(\mathcal{S}, \mathcal{T})$ by composing (and synchronizing) $\mathcal{S}$ and $\mathcal{T}$. Hence, the sets of states of A-$comp(\mathcal{S}, \mathcal{T})$ and D-$comp(\mathcal{S}, \mathcal{T})$ subsume $S \times T$ (to prevent confusion, states of A-$comp(\mathcal{S}, \mathcal{T})$ are marked by a horizontal bar; hence, $\overline{(s,t)}$ is a state of A-$comp(\mathcal{S}, \mathcal{T})$ while (s,t) is a state of D-$comp(\mathcal{S}, \mathcal{T})$). The goal is to obtain the property that, for all $s \in S$, $t \in T$ we have that $s \sim t$ iff $\overline{(s,t)} \sqsubseteq (s,t)$. Note that each player has his own copy of $\mathcal{S}$ and $\mathcal{T}$.

The simulation game proceeds as follows: The attacker (playing in A-$comp(\mathcal{S}, \mathcal{T})$) chooses either $\mathcal{S}$ or $\mathcal{T}$ and makes a move there. Let us assume that the attacker chooses $\mathcal{S}$ (the other case is symmetric). Then the defender (playing in D-$comp(\mathcal{S}, \mathcal{T})$) must make exactly the same move in his copy of $\mathcal{S}$ as the attacker, but also some move in his copy of $\mathcal{T}$. The defender can choose which move in $\mathcal{T}$ he makes, provided it has the same action as the attacker's move. Furthermore, the defender 'threatens' to go to a universal state (that can simulate everything) unless the attacker does a specific action in the next round. In the next round the attacker must make exactly the same move in

[1] Very recently [KM02a], the authors proved that **PDA** $\sqsubseteq$ **FS** is actually **EXPTIME**-complete and **PDA** $\sim$ **FS** is **PSPACE**-complete.

his (the attacker's) copy of $\mathcal{T}$ as the defender did in his (the defender's) copy of $\mathcal{T}$ in the previous round. The defender responds to this by ending his threat to become universal. Otherwise the defender can make his side universal and wins the simulation game.

This construction ensures that the two copies of $\mathcal{S}$ and $\mathcal{T}$ on both sides are kept consistent. One round of the bisimulation game between $\mathcal{S}$ and $\mathcal{T}$ is thus emulated by two rounds of the simulation game between A-$comp(\mathcal{S},\mathcal{T})$ and D-$comp(\mathcal{S},\mathcal{T})$.

Of course, it is possible that for a given state s there are several different outgoing arcs labeled with the same action a_i. However, we need to construct new systems where outgoing transitions are labeled uniquely. Our notation is similar to the one used in the previous section: Let $\Lambda = \{\lambda_i^j \mid 1 \leq i \leq k, 1 \leq j \leq d\}$ and $\Delta = \{\delta_i^j \mid 1 \leq i \leq k, 1 \leq j \leq d\}$ be finite sets of actions such that Λ, Δ, and Act are pairwise disjoint. We define $Act' = \Lambda \cup \Delta$.

For any state s in S, the action δ_i^j is used to label the j-th outgoing arc that was previously labeled by action a_i. Note that the action δ_i^j can occur many times in the transition system. It is only unique among the labels of the outgoing arcs of any single state. Similarly, the actions λ_i^j are used to label the outgoing arcs of states t in T.

Fig. 2 illustrates the construction. The first row shows parts of the original systems $\mathcal{S}$ and $\mathcal{T}$. The second row shows A-$comp(\mathcal{S},\mathcal{T})$. The labels of the transitions have been changed as described above, and the modified systems have been put in parallel without any synchronization. The last row shows (a fragment of) D-$comp(\mathcal{S},\mathcal{T})$. Here the systems $\mathcal{S}$ and $\mathcal{T}$ have been composed and synchronized in such a way as to ensure the properties of the simulation game as described above.

Definition 13. *We define transition systems $\mathcal{S}' = (S, Act', \rightarrow)$ and $\mathcal{T}' = (T, Act', \rightarrow)$ where*

- *for every transition $s \xrightarrow{a_i} s(a_i, j)$ in $\mathcal{S}$ there is a transition $s \xrightarrow{\delta_i^j} s(a_i, j)$ in $\mathcal{S}'$;*
- *for every transition $t \xrightarrow{a_i} t(a_i, \ell)$ in $\mathcal{T}$ there is a transition $t \xrightarrow{\lambda_i^\ell} t(a_i, \ell)$ in $\mathcal{T}'$;*
- *there are no other transitions in $\mathcal{S}'$ and $\mathcal{T}'$.*

*The A-*composition *A-$comp(\mathcal{S},\mathcal{T})$ of $\mathcal{S}$ and $\mathcal{T}$ (see Fig.2) is the parallel composition $\mathcal{S}' \| \mathcal{T}'$. Configurations in A-$comp(\mathcal{S},\mathcal{T})$ are denoted by $\overline{(s,t)}$.*

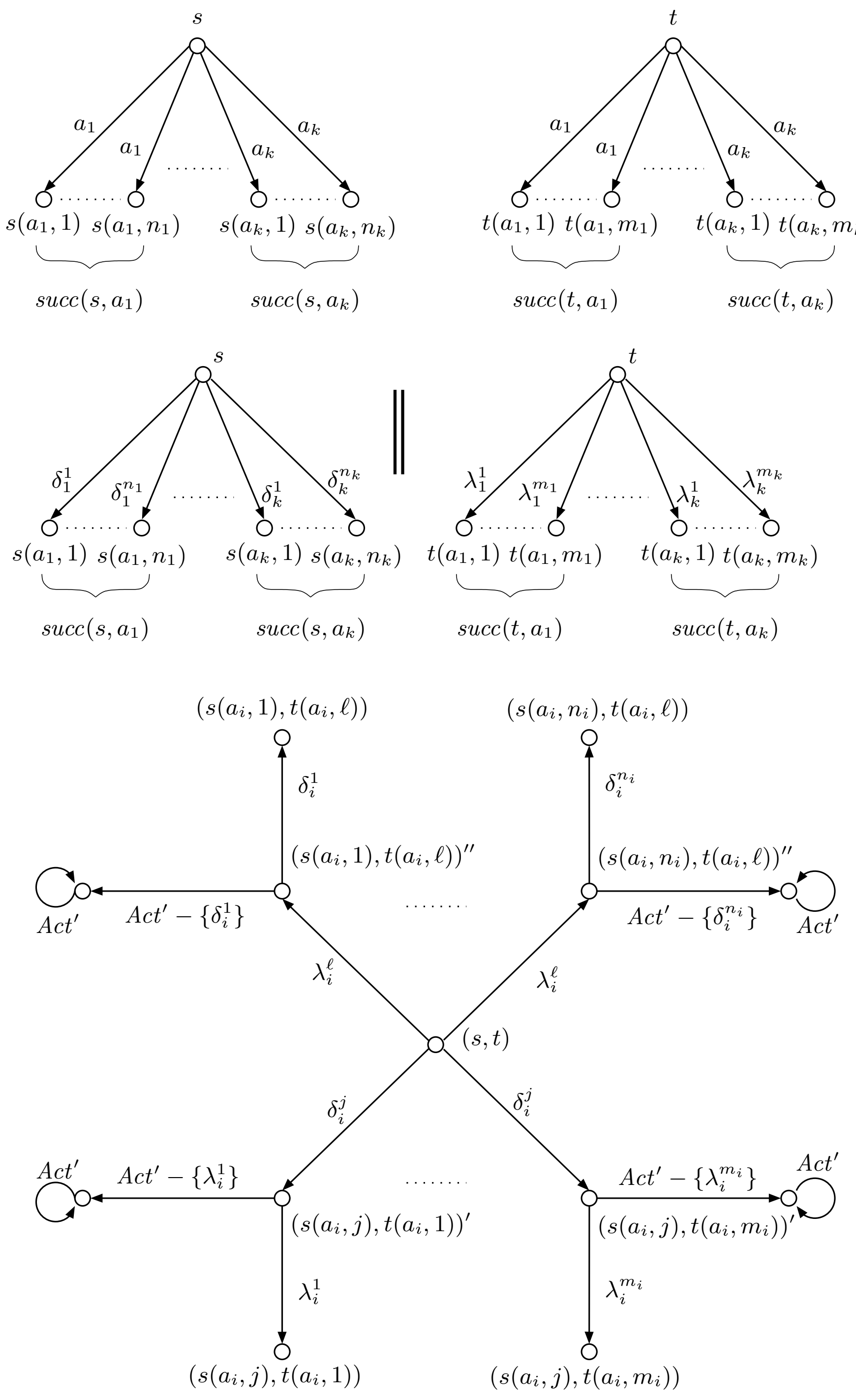
Remark 14. Observe that A-$comp(\mathcal{S},\mathcal{T})$ is always deterministic, even if $\mathcal{S}$ and $\mathcal{T}$ are nondeterministic.

A state u is *universal* if $u \xrightarrow{a} u$ for every $a \in Act'$ (u can also have other outgoing transitions). Observe that a universal state can simulate *any* process which emits actions of Act'. In the next definition, we write $t \xrightarrow{a} U$ to indicate that $t \xrightarrow{a} u$ for some universal state u.

Definition 15. *The D-*composition *D-$comp(\mathcal{S},\mathcal{T})$ of $\mathcal{S}$ and $\mathcal{T}$ (see Fig. 2) is the transition system $\mathcal{D} = (D, Act', \rightarrow)$, where D is the set*

$$\{(s,t) \,|\, s \in S, t \in T\} \cup \{(s,t)' \,|\, s \in S, t \in T\} \cup \{(s,t)'' \,|\, s \in S, t \in T\}$$

and the transition relation of $\mathcal{D}$ is defined as follows. Let $1 \leq i \leq k$, $1 \leq j \leq n_i$ and $1 \leq \ell \leq m_i$.



s
a_1
a_1
a_k
a_k
$s(a_1,1)$ $s(a_1,n_1)$ $s(a_k,1)$ $s(a_k,n_k)$
$succ(s,a_1)$ $succ(s,a_k)$
t
$t(a_1,1)$ $t(a_1,m_1)$ $t(a_k,1)$ $t(a_k,m_k)$
$succ(t,a_1)$ $succ(t,a_k)$
δ_1^1 $\delta_1^{n_1}$ δ_k^1 $\delta_k^{n_k}$
λ_1^1 $\lambda_1^{m_1}$ λ_k^1 $\lambda_k^{m_k}$
$\|$
$(s(a_i,1),t(a_i,\ell))$
$(s(a_i,n_i),t(a_i,\ell))$
δ_i^1
$\delta_i^{n_i}$
$(s(a_i,1),t(a_i,\ell))''$
$(s(a_i,n_i),t(a_i,\ell))''$
Act'
$Act'-\{\delta_i^1\}$
$Act'-\{\delta_i^{n_i}\}$
λ_i^ℓ
λ_i^ℓ
(s,t)
δ_i^j
δ_i^j
$Act'-\{\lambda_i^1\}$
$Act'-\{\lambda_i^{m_i}\}$
$(s(a_i,j),t(a_i,1))'$
$(s(a_i,j),t(a_i,m_i))'$
λ_i^1
$\lambda_i^{m_i}$
$(s(a_i,j),t(a_i,1))$
$(s(a_i,j),t(a_i,m_i))$

Fig. 2. States s and t of $\mathcal{S}$ and $\mathcal{T}$, then the A-composition of $\mathcal{S}$ and $\mathcal{T}$, and finally the D-composition of $\mathcal{S}$ and $\mathcal{T}$.

– *If there are transitions $s \xrightarrow{a_i} s(a_i,j)$ in $\mathcal{S}$ and $t \xrightarrow{a_i} t(a_i,\ell)$ in $\mathcal{T}$ then there are transitions $(s,t) \xrightarrow{\delta_i^j} (s(a_i,j),t(a_i,\ell))'$ and $(s,t) \xrightarrow{\lambda_i^\ell} (s(a_i,j),t(a_i,\ell))''$ in $\mathcal{D}$.*
– $(s(a_i,j),t(a_i,\ell))' \xrightarrow{\lambda_i^\ell} (s(a_i,j),t(a_i,\ell))$
– $(s(a_i,j),t(a_i,\ell))' \xrightarrow{b} U$*, for each* $b \in Act' - \{\lambda_i^\ell\}$.
– $(s(a_i,j),t(a_i,\ell))'' \xrightarrow{\delta_i^j} (s(a_i,j),t(a_i,\ell))$
– $(s(a_i,j),t(a_i,\ell))'' \xrightarrow{c} U$*, for each* $c \in Act' - \{\delta_i^j\}$.

Theorem 16. *Let $s \in S$, $t \in T$, and let $\overline{(s,t)}$ and (s,t) be the corresponding states in A-$comp(\mathcal{S},\mathcal{T})$ and D-$comp(\mathcal{S},\mathcal{T})$, respectively. We have that $s \sim t$ iff $\overline{(s,t)} \sqsubseteq (s,t)$.*

Proof. We start with the '$\Longrightarrow$' direction. Let us suppose that $s \sim t$. We show that the defender has a winning strategy in a simulation game initiated in $(\overline{(s,t)},(s,t))$. Let $\mathcal{R} = \{(\overline{(s,t)},(s,t)) \mid s \in S, t \in T, s \sim t\}$. We prove that $\mathcal{R}$ can be extended to a simulation relation (which means that indeed $\overline{(s,t)} \sqsubseteq (s,t)$). To do that, we show that the defender can play in such a way that after at most two rounds he achieves either a configuration from $\mathcal{R}$, or a configuration where he 'obviously' wins.

Let us assume that the attacker makes a move $\overline{(s,t)} \xrightarrow{\delta_i^j} \overline{(s(a_i,j),t)}$. (The other case where the attacker makes a move $\overline{(s,t)} \xrightarrow{\lambda_i^\ell} \overline{(s,t(a_i,\ell))}$ is symmetric.) Then the defender responds by a move $(s,t) \xrightarrow{\delta_i^j} (s(a_i,j),t(a_i,\ell))'$ such that $s(a_i,j) \sim t(a_i,\ell)$. Such a move must exist by the definition of D-$comp(\mathcal{S},\mathcal{T})$ and the fact that $s \sim t$. Then there are two cases:

– If the attacker makes the move $\overline{(s(a_i,j),t)} \xrightarrow{\lambda_i^\ell} \overline{(s(a_i,j),t(a_i,\ell))}$ then the defender responds by the move $(s(a_i,j),t(a_i,\ell))' \xrightarrow{\lambda_i^\ell} (s(a_i,j),t(a_i,\ell))$. The resulting pair $(\overline{(s(a_i,j),t(a_i,\ell))},(s(a_i,j),t(a_i,\ell)))$ is in $\mathcal{R}$, because $s(a_i,j) \sim t(a_i,\ell)$.
– If the attacker makes any other move then the defender can go to a universal state where he can simulate everything.

Now we show the '$\Longleftarrow$' direction, i.e., we assume $\overline{(s,t)} \sqsubseteq (s,t)$ and prove $s \sim t$. To do that, we demonstrate that $\mathcal{R} = \{(s,t) \mid s \in S, t \in T, \overline{(s,t)} \sqsubseteq (s,t)\}$ is a bisimulation. So, let $(s,t) \in \mathcal{R}$. Assume that the attacker makes a move $s \xrightarrow{a_i} s(a_i,j)$. (The other case where the attacker makes a move $t \xrightarrow{a_i} t(a_i,\ell)$ is symmetric.) Thus, there is an attacker's move $\overline{(s,t)} \xrightarrow{\delta_i^j} \overline{(s(a_i,j),t)}$. Since $\overline{(s,t)} \sqsubseteq (s,t)$ there must be a defender move $(s,t) \xrightarrow{\delta_i^j} (s(a_i,j),t(a_i,\ell))'$ such that $\overline{(s(a_i,j),t)} \sqsubseteq (s(a_i,j),t(a_i,\ell))'$. Also there must be a move $t \xrightarrow{a_i} t(a_i,\ell)$ in $\mathcal{T}$. Furthermore, against the attacker move $\overline{(s(a_i,j),t)} \xrightarrow{\lambda_i^\ell} \overline{(s(a_i,j),t(a_i,\ell)}$ there is just one defender move $(s(a_i,j),t(a_i,\ell))' \xrightarrow{\lambda_i^\ell} (s(a_i,j),t(a_i,\ell))$ such that $\overline{(s(a_i,j),t(a_i,\ell)} \sqsubseteq (s(a_i,j),t(a_i,\ell))$. Thus, we obtain $(s(a_i,j),t(a_i,\ell)) \in \mathcal{R}$. □

3.1 Applications

The range of applicability of Theorem 16 is incomparable with the one of Theorem 7.

Definition 17. *Let us denote by $\mathcal{C}$ the class of all models **C** to which our second method applies. The following conditions are sufficient for membership of $\mathcal{C}$.*

1. *For any $\mathcal{M} \in$ **C**, the transition system $\mathcal{T}_{\mathcal{M}}$ determined by $\mathcal{M}$ satisfies the following condition: the out-going transitions of a given state in $\mathcal{T}_{\mathcal{M}}$ are determined by a finite number of labeled 'transition rules' which are a part of $\mathcal{M}$. The transition rules must be* injective *in the sense that each rule generates at most one out-going transition in every state of $\mathcal{T}_{\mathcal{M}}$. The label of this outgoing transition is the same as the label of the associated transition rule.*
2. ***C** is efficiently closed under parallel composition (i.e., for all $\mathcal{M}_1, \mathcal{M}_2 \in$ **C** there is $\mathcal{M}_3 \in$ **C** computable in polynomial time such that $\mathcal{T}_{\mathcal{M}_3} = \mathcal{T}_{\mathcal{M}_1} \| \mathcal{T}_{\mathcal{M}_2}$).*
3. ***C** is efficiently closed under synchronization in the following sense: for any two given transition rules it is possible to (efficiently) define a new transition rule that has the effect of both.*
4. ***C** subsumes the class of finite automata.*

For example, the above conditions are satisfied by 1-safe Petri nets, general Petri nets, reset Petri nets, transfer Petri nets [Pet81], VASS (vector addition systems with states) [BM99], FIFO-channel systems [AJ93], etc. However, there are also some classes that not in $\mathcal{C}$. For example, Basic Parallel Processes [Chr93] are not in $\mathcal{C}$, because they are not closed under synchronization (condition 3). Pushdown automata are not in $\mathcal{C}$, because they are not closed under parallel composition (condition 2). PA-processes [BK85] are not in $\mathcal{C}$, because they are not closed under synchronization and because their transition rules are not injective (they do not satisfy conditions 1 and 3).

Lemma 18. *Let **C** be a process model satisfying the above conditions and let $\mathcal{M}_1, \mathcal{M}_2 \in$ **C**. Then there is $\mathcal{M}' \in$ **C** constructible in polynomial time such that $\mathcal{T}_{\mathcal{M}'}$ is (isomorphic to) A-$comp(\mathcal{T}_{\mathcal{M}_1}, \mathcal{T}_{\mathcal{M}_2})$.*

Proof. Follows immediately from condition 1 (which enables efficient renaming of actions) and condition 2 (efficient parallel composition). □

Lemma 19. *Let **C** be a process model satisfying the above conditions and let $\mathcal{M}_1, \mathcal{M}_2 \in$ **C**. Then there is $\mathcal{M}' \in$ **C** constructible in polynomial time such that $\mathcal{T}_{\mathcal{M}'}$ is (isomorphic to) D-$comp(\mathcal{T}_{\mathcal{M}_1}, \mathcal{T}_{\mathcal{M}_2})$.*

Proof. $\mathcal{M}'$ is obtained by constructing the synchronization of $\mathcal{M}_1$ and $\mathcal{M}_2$ according to Definition 15 (this is possible by conditions 1 and 3). It is also necessary to include the states of the form $(s,t)'$ and $(s,t)''$ into the new system. This is possible because **C** subsumes the class of finite automata (condition 4) and is thus also closed under synchronization with them (condition 3). □

Corollary 20. *Let **C** be a process model satisfying the above conditions, and let **det-C** be the subclass of all $\mathcal{M} \in$ **C** which generate a deterministic transition system. Then the problem **C** $\sim$ **C** is polynomially reducible to **det-C** $\sqsubseteq$ **C** (and also to **C** $\simeq$ **C**).*

Proof. Immediately from Theorem 16, Remark 14, Lemma 18, and Lemma 19. The reduction to **C** $\simeq$ **C** is achieved in the same way as in Corollary 11. □

4 Conclusion

We have described two general methods to construct direct one-to-one reductions from bisimulation equivalence to simulation preorder (or simulation equivalence) on labeled transition systems. On many classes of finitely generated transition systems these reductions are even effectively computable in polynomial time. Generally speaking, the first method is effective for all classes of systems that can test for non-enabledness of actions, like finite-state systems, pushdown automata, context-free processes (BPA), one-counter machines, FIFO-channel systems with explicit test for queue-emptiness, 1-safe Petri nets, Petri nets with inhibitor arcs, and various subclasses of all these. The second method is effective for all classes of systems that are closed under parallel composition and synchronization like 1-safe Petri nets, general Petri nets, reset Petri nets, transfer Petri nets, VASS (vector addition systems with states) [BM99] and FIFO-channel systems [AJ93].

Thus, for all these classes of systems, deciding simulation preorder/equivalence must be computationally at least as hard as deciding bisimulation equivalence. This provides a formal justification of the general rules of thumb mentioned in Section 1. It is interesting to compare these results to the results in [KM02b], where an abstract, (but not generally effective) one-to-one reduction from simulation equivalence to bisimulation equivalence on general labeled transition systems was presented (i.e., a reduction in the other direction). Therefore, these results further clarify the relationship between simulation equivalence and bisimulation equivalence.

For some classes of systems both methods are effective, e.g., for finite-state systems, FIFO-channel systems with explicit test for emptiness, or for 1-safe Petri nets. However, for general Petri nets only the second method works (the first one fails since Petri nets cannot test for non-enabledness of actions). For pushdown automata it is vice-versa. They can test for non-enabledness of actions, but are not closed under parallel composition.

Finally, there remain a few classes of systems for which none of our methods is effective, like Basic Parallel Processes (BPP) and PA-processes. Also for these classes, simulation preorder/equivalence is computationally harder than bisimulation equivalence [KM02b], but no effective *direct* reduction from bisimulation equivalence to simulation preorder is known for them yet (although effective indirect reductions exist).

References

[AJ93] P.A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proceedings of LICS'93*, pages 160–170. IEEE Computer Society Press, 1993.

[BGS92] J. Balcázar, J. Gabarró, and M. Sántha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4(6A):638–648, 1992.

[BK85] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[BM99] A. Bouajjani and R. Mayr. Model-checking lossy vector addition systems. In *Proceedings of STACS'99*, volume 1563 of *LNCS*, pages 323–333. Springer, 1999.

[Chr93] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, The University of Edinburgh, 1993.

[JKM00] P. Jančar, A. Kučera, and F. Moller. Simulation and bisimulation over one-counter processes. In *Proceedings of STACS 2000*, volume 1770 of *LNCS*, pages 334–345. Springer, 2000.

[KM02a] A. Kučera and R. Mayr. On the complexity of semantic equivalences for pushdown automata and BPA. In *Proceedings of MFCS2002*, LNCS. Springer, 2002. To appear.

[KM02b] A. Kučera and R. Mayr. Simulation preorder over simple process algebras. *Information and Computation*, 173(2):184–198, 2002.

[May00] R. Mayr. On the complexity of bisimulation problems for pushdown automata. In *Proceedings of IFIP TCS'2000*, volume 1872 of *LNCS*, pages 474–488. Springer, 2000.

[Mol96] F. Moller. Infinite results. In *Proceedings of CONCUR'96*, volume 1119 of *LNCS*, pages 195–216. Springer, 1996.

[Pet81] J.L. Peterson. *Petri Net Theory and the Modelling of Systems*. Prentice-Hall, 1981.

[SJ01] Z. Sawa and P. Jančar. P-hardness of equivalence testing on finite-state processes. In *Proceedings of SOFSEM'2001*, volume 2234 of *LNCS*, pages 326–335. Springer, 2001.

[Sti98] C. Stirling. The joys of bisimulation. In *Proceedings of MFCS'98*, volume 1450 of *LNCS*, pages 142–151. Springer, 1998.

[Tho93] W. Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science. In *Proceedings of TAPSOFT'93*, volume 668 of *LNCS*, pages 559–568. Springer, 1993.

[vG99] R. van Glabbeek. The linear time—branching time spectrum. *Handbook of Process Algebra*, pages 3–99, 1999.