Acta Informatica manuscript No.
(will be inserted by the editor)

Zdeněk Sawa · Petr Jančar

Hardness of Equivalence Checking for Composed Finite-State Systems*

27 November 2008

Abstract Computational complexity of comparing behaviours of systems composed from interacting finite-state components is considered. The main result shows that the respective problems are EXPTIME-hard for all relations between bisimulation equivalence and trace preorder, as conjectured by Rabinovich (1997). The result is proved for a specific model of parallel compositions where the components synchronize on shared actions but it can be easily extended to other similar models, e.g., to labelled 1-safe Petri nets. Further hardness results are shown for special cases of acyclic systems.

Key words: behavioural equivalences, communicating finite-state systems, computational complexity

1 Introduction

One problem arising in the area of (automated) verification consists in comparing two finite-state systems w.r.t. a behavioural relation, like bisimulation equivalence, simulation preorder, or trace equivalence. Systems are often presented as sets (parallel compositions) of communicating agents; the global state space of such a *composed system* is usually exponential in the size of the system presentation. This phenomenon is known as 'state explosion', which

* The authors gratefully acknowledge the support by the Czech Ministry of Education, Grant No. 1M0567

Z. Sawa · P. Jančar
FEI, Technical University of Ostrava, 17. listopadu 15, Ostrava-Poruba, 708 33, Czech republic
Tel.: ++420 597 323 476, Fax: ++420 596 919 597
E-mail: Zdenek.Sawa@vsb.cz, Petr.Jancar@vsb.cz

is the main challenge in the design of efficient algorithms for verification of such systems.

The straightforward approach, where the global state space is explicitly constructed and the behavioural relation is decided on the resulting system, requires exponential space. It is natural to ask if some more efficient algorithms exist in special cases. For example, Groote and Moller [4] have shown that if the components of the system can perform actions independently and there is no communication between them then the bisimulation equivalence (and some other equivalences that satisfy certain axioms) can be decided in polynomial time. As one may expect, the problem becomes harder when communication between components is allowed. Rabinovich [14] considered a general model of composed systems which can be called *Parallel Composition with Hiding* (PCH); the components (are forced to) synchronize via shared actions, and the identity of some actions can be 'hidden', which refers to replacing with a special action τ. He formulated the EXPTIME-hardness conjecture for PCH and any relation between bisimilarity and trace preorder; Figure 1 shows van Glabbeek's linear time – branching time spectrum [3], containing various behavioural equivalences in that range.

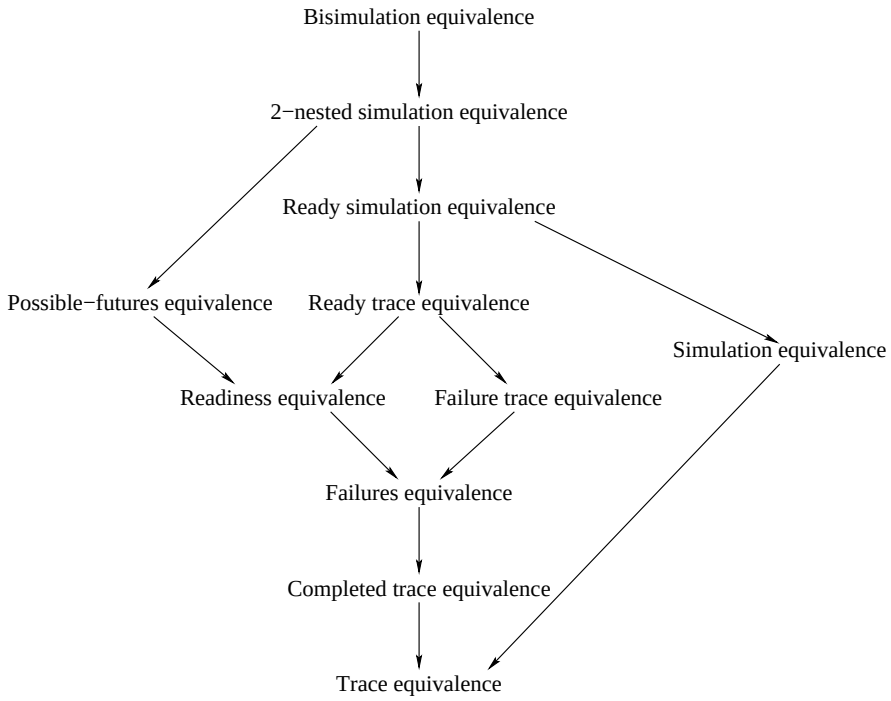

Fig. 1 The linear time – branching time spectrum

Nevertheless, Rabinovich proved only PSPACE-hardness in general, though he showed EXPSPACE-completeness for trace equivalence and mentioned EXPTIME-completeness for bisimilarity.

Laroussinie and Schnoebelen [9] approved the EXPTIME-hardness conjecture for all relations between bisimilarity and simulation preorder, even for composed systems with no hiding, denoted PC. It is hardly possible to extend EXPTIME-hardness for PC to all relations in the spectrum-range since trace equivalence is in PSPACE for this model, as was proved in [17]; see also [18] for results for other types of 'trace-like' equivalences and composed systems. But it is also not clear how the result of [9] could be extended using hiding.

In this paper, we approve Rabinovich's conjecture in the whole, deriving EXPTIME-hardness for PCH and any relation between bisimilarity and trace preorder. (A preliminary version of this proof appeared in [15].)
We also study acyclic PC and PCH, which were also considered in [14]. We provide an alternative (simpler) reduction, and raise the lower bound (of NP-hardness and coNP-hardness) to DP-hardness. We also derive PSPACE-hardness for acyclic PC and any relation between bisimilarity and simulation preorder. Finally we provide an overview of all known relevant results.

Our EXPTIME-hardness proof is achieved by a reduction from the acceptance problem of alternating LBAs (linear bounded automata), denoted ALBA-ACCEPT. As an important ingredient, we first provide a new proof of PTIME-hardness for *explicit* finite-state systems (given by listing the whole state space) and any relation between bisimilarity and trace preorder; this is achieved by a reduction from AGP, a problem on alternating graphs.
PTIME-hardness for bisimilarity on explicit finite-state systems was shown in [1], and a construction demonstrating PTIME-hardness for the whole spectrum-range was given in [16]; nevertheless, that construction could not be used for our aims here.

For describing the behaviour of the composed system arising by the reduction from an instance of ALBA-ACCEPT in the EXPTIME-hardness proof, it is also useful to introduce a 'reactive' version of linear bounded automata as an intermediate model.
These reactive LBAs can be easily modeled not only by PCH but also by other models of composed systems, like, e.g., labelled 1-safe Petri nets; the EXPTIME-hardness result is thus carried over to them as well.

The paper is organized as follows. Section 2 contains basic definitions. Section 3 provides a construction showing PTIME-hardness for explicit finite-state systems. In Section 4 we apply the ideas from Section 3 to derive EXPTIME-hardness for composed systems; to this aim, also reactive LBAs are introduced. Section 5 deals with acyclic PC and PCH. Section 6 contains an overview of all relevant results.

2 Definitions

2.1 Labelled transition systems, behavioural equivalences and preorders

A *labelled transition system* (LTS) is a tuple $\mathcal{T} = (S, Act, \longrightarrow)$ where S is a set of *states*, Act is a set of *actions* (sometimes called the *action alphabet*), and $\longrightarrow \subseteq S \times Act \times S$ is the *transition relation*. We write $s \xrightarrow{a} s'$ instead of $(s, a, s') \in \longrightarrow$. We also write $s \xrightarrow{w} s'$ for $w \in Act^*$; for $w = a_1 a_2 \cdots a_n$ ($a_i \in Act$) this means that there are some states $s_0, s_1, \ldots, s_n$ where $s = s_0$, $s' = s_n$, and $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$. For $s \in S$, $w \in Act^*$, the set of states *reachable* from s *by the action sequence* w is denoted

$$reach(s, w) = \{s' \in S \mid s \xrightarrow{w} s'\}\,.$$

We say that $w \in Act^*$ is *enabled* in s iff $reach(s, w) \neq \emptyset$; it is *disabled* otherwise. A *state* s is called *deterministic* iff $|reach(s, a)| \leq 1$ for each action $a \in Act$.

A *trace* from $s \in S$ is any $w \in Act^*$ such that $reach(s, w) \neq \emptyset$. We denote the set of all traces from s as $Traces(s)$. States s, t are in *trace preorder*, written $s \sqsubseteq_{tr} t$, iff $Traces(s) \subseteq Traces(t)$; they are *trace equivalent* iff $Traces(s) = Traces(t)$.

A relation $\mathcal{R} \subseteq S \times S$ is a *simulation* iff the following condition holds for each $s, t \in S$ such that $(s, t) \in \mathcal{R}$:
if $s \xrightarrow{a} s'$ for some state s' and some action a then there is some t' such that $t \xrightarrow{a} t'$ and $(s', t') \in \mathcal{R}$.
A symmetric simulation is a *bisimulation*. States s, s' are *bisimilar*, written $s \sim s'$, iff there is some bisimulation $\mathcal{R}$ such that $(s, s') \in \mathcal{R}$. The relation $\sim$ is called *bisimulation equivalence* or *bisimilarity*. States s, s' are in the *simulation preorder*, written $s \sqsubseteq_{sim} s'$, iff there is a simulation $\mathcal{R}$ such that $(s, s') \in \mathcal{R}$, and they are *simulation equivalent* iff $s \sqsubseteq_{sim} s'$ and $s' \sqsubseteq_{sim} s$.

We note that we can also naturally compare states from different LTSs $\mathcal{T}_1, \mathcal{T}_2$, viewing them as states in the LTS $\mathcal{T}$ which arises as the disjoint union of $\mathcal{T}_1, \mathcal{T}_2$.

It is usual to describe the above relations in terms of games. In particular, bisimilarity on a given LTS $\mathcal{T}$ can be described in terms of the *bisimulation game* played by two players, called Player I and Player II. Positions in the game are pairs of states of $\mathcal{T}$ and the game is played in rounds. In a round starting in position (s_1, s_2) Player I chooses $i \in \{1, 2\}$ and a transition $s_i \xrightarrow{a} s_i'$. Player II then chooses some s_{3-i}' such that $s_{3-i} \xrightarrow{a} s_{3-i}'$. The play then continues by the next round from (s_1', s_2'). If one of the players gets stuck, he loses and the other player wins. Player II is the winner if the play is infinite. The *simulation game* is the bisimulation game where Player I is always obliged to choose $i = 1$ (i.e., to play on the left-hand side).

Any bisimulation (simulation) containing (s, t) naturally provides a winning strategy for Player II in the bisimulation (simulation) game starting from

(s,t). Hence $s \sim t$ ($s \sqsubseteq_{sim} t$) iff Player II has a winning strategy in the bisimulation (simulation) game starting in (s,t); Player I has a winning strategy iff $s \not\sim t$ ($s \not\sqsubseteq_{sim} t$).

2.2 Operations on labelled transition systems

There are several possible ways how to define systems composed of components running in parallel and communicating with each other. In this paper we use operations of parallel composition and hiding as defined for example in [5] that were also used in [14].

We reserve symbol τ for denoting a special action; it serves for renaming the usual (visible) actions whose identity we want to hide.

Remark. For the above defined behavioural relations, this τ is taken as a normal action (so 'bisimilarity' means 'strong bisimilarity', etc.).

Given an LTS $\mathcal{T} = (S, Act \cup \{\tau\}, \longrightarrow)$, $\tau \notin Act$, and $\mathcal{B} \subseteq Act$, by

$$hide\ \mathcal{B}\ in\ \mathcal{T}$$

we denote the LTS $\mathcal{T}_1 = (S, (Act - \mathcal{B}) \cup \{\tau\}, \longrightarrow_1)$ where $s \xrightarrow{a}_1 s'$ in $\mathcal{T}_1$ iff

- either $a \in (Act - \mathcal{B}) \cup \{\tau\}$ and $s \xrightarrow{a} s'$ in $\mathcal{T}$,
- or $a = \tau$ and $s \xrightarrow{b} s'$ in $\mathcal{T}$ for some $b \in \mathcal{B}$.

The *parallel composition* of LTSs $\mathcal{T}_1 = (S_1, Act_1 \cup \{\tau\}, \longrightarrow_1)$, $\mathcal{T}_2 = (S_2, Act_2 \cup \{\tau\}, \longrightarrow_2)$ (where $\tau \notin (Act_1 \cup Act_2)$) is the LTS

$$\mathcal{T}_1 \parallel \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2 \cup \{\tau\}, \longrightarrow)$$

representing the product of $\mathcal{T}_1$ and $\mathcal{T}_2$ in which $\mathcal{T}_1$ and $\mathcal{T}_2$ synchronize on the shared non-τ actions, i.e., where $(s_1, s_2) \xrightarrow{a} (s'_1, s'_2)$ iff

- either $a \in Act_1 \cup Act_2$ and for each $i \in \{1,2\}$ we have: if $a \in Act_i$ then $s_i \xrightarrow{a}_i s'_i$ and otherwise $s'_i = s_i$,
- or $a = \tau$ and $s_i \xrightarrow{\tau}_i s'_i$ and $s_{3-i} = s'_{3-i}$ for some $i \in \{1,2\}$.

Hence if $a \in Act_1 \cap Act_2$ then both components perform a-transitions simultaneously; otherwise just one component does.

It is obvious that $\parallel$ is associative and commutative with respect to isomorphism, and we freely write

$$\mathcal{T} = \mathcal{T}_1 \parallel \mathcal{T}_2 \parallel \cdots \parallel \mathcal{T}_n$$

for the parallel composition of $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$.

Remark. The above described operations of parallel composition and hiding constitute one natural choice of how to model communication between

components running in parallel. Nevertheless our results can be extended to other similar operators defined in literature; though the operators we used allow multi-sided communication, our constructions are always realized by only using two-sided communication. For example, CCS parallel composition (where each action a has a complementary action $\bar{a}$) and restriction [12] could have been used in the constructions instead of the operations defined above.

2.3 Central models and problems

In this paper we consider only *finite* labelled transition systems, where the sets of states and actions are finite. By an *explicit finite-state system*, denoted FS, we mean a finite LTS together with its presentation which lists all states, actions and transitions. The size $|\mathcal{T}|$ of FS $\mathcal{T} = (S, Act, \longrightarrow)$ is $|S| + |Act| + |\longrightarrow|$.

By a system presented as *parallel composition without hiding*, denoted PC, we mean a presentation (of an LTS) in the form

$$\mathcal{T} = \mathcal{T}_1 \parallel \mathcal{T}_2 \parallel \cdots \parallel \mathcal{T}_n$$

where $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ are explicit finite-state systems, called *components* of PC $\mathcal{T}$; we also assume that their action alphabets do not contain τ. The size of PC $\mathcal{T}$ is $|\mathcal{T}| = |\mathcal{T}_1| + |\mathcal{T}_2| + \ldots + |\mathcal{T}_n|$. A state of $\mathcal{T}$, called a *global state*, is $E = (s_1, s_2, \ldots, s_n)$ where $s_1, s_2, \ldots, s_n$ are states of the components $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ respectively. We note that the number of (global) states of $\mathcal{T}$ can be exponential in $|\mathcal{T}|$.

Suppose that $Act_1, Act_2, \ldots, Act_n$ are action alphabets of $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ and note that if $E \xrightarrow{a} E'$ then *all* components $\mathcal{T}_i$ such that $a \in Act_i$ must perform an a-transition. We say that such components *participate* in the transition.

It can be easily shown (see [14]) that any system constructed from explicit finite-state systems by a finite number of applications of parallel composition and hiding can be transformed into an isomorphic system (of the same size) in the form

$$\textit{hide } \mathcal{B} \textit{ in } (\mathcal{T}_1 \parallel \mathcal{T}_2 \parallel \cdots \parallel \mathcal{T}_n)$$

where $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ are explicit finite-state systems. (We can use the fact that $(\textit{hide } \mathcal{B} \textit{ in } \mathcal{T}) \parallel \mathcal{T}'$ is isomorphic to $\textit{hide } \mathcal{B} \textit{ in } (\mathcal{T} \parallel \mathcal{T}')$ provided that actions of $\mathcal{T}'$ do not occur in $\mathcal{B}$, which can be ensured by renaming of actions in $\mathcal{B}$ with fresh names when necessary.) We call a system presented in the specified form as *parallel composition with hiding*, denoted PCH, and we define its size as $|\mathcal{T}_1| + |\mathcal{T}_2| + \cdots + |\mathcal{T}_n| + |\mathcal{B}|$. PC is the special case of PCH where $\mathcal{B} = \emptyset$ and where components do not have τ actions.

The general form of problems we consider is

PROBLEM: C-REL$_{\mathcal{R}}$
INSTANCE: Presentation of an LTS $\mathcal{T}$ of type C, with two distinguished states s and s'.
QUESTION: Is $(s, s') \in \mathcal{R}$?

where C is a fixed type (class) of systems (such as FS, PC, or PCH) and $\mathcal{R}$ is some fixed binary relation defined on states of LTSs; in fact, we only consider relations $\mathcal{R}$ *between bisimilarity and trace preorder*, i.e., $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$.

Remark. Note that the more general problem where the instance consists of presentations of two LTS $\mathcal{T}_1$ and $\mathcal{T}_2$ with initial states s_1 and s_2 (and question is whether $(s_1, s_2) \in \mathcal{R}$) can be easily transformed into the problem C-REL$_{\mathcal{R}}$ by considering the disjoint union of $\mathcal{T}_1$ and $\mathcal{T}_2$.

We are thus mainly interested in problems

$$\text{FS-REL}_{\mathcal{R}},\ \text{PC-REL}_{\mathcal{R}},\ \text{PCH-REL}_{\mathcal{R}}$$

but we will also consider problems restricted to acyclic systems: a PCH $\mathcal{T}$ is *acyclic* iff (the graph of) every component of $\mathcal{T}$ is acyclic; the respective problems are denoted

$$\text{APC-REL}_{\mathcal{R}},\ \text{APCH-REL}_{\mathcal{R}}.$$

It is also worth to note that systems constructed in our reductions are in a special form: a PCH $\mathcal{T}$ is *centralized* iff it has a special *control component* $\mathcal{T}_c$ such that at most two components participate in each transition and $\mathcal{T}_c$ is always one of them.

We finish by stating a fact which we often use when establishing a hardness result.

Observation 1 *A problem P is* PTIME*-hard (*PSPACE*-hard,* EXPTIME*-hard) iff the complement of P is* PTIME*-hard (*PSPACE*-hard,* EXPTIME*-hard).*

Remark. In this paper we consider a problem P to be $\mathcal{C}$*-hard* (where $\mathcal{C} \in \{\text{PTIME}, \text{NP}, \text{coNP}, \text{PSPACE}, \text{EXPTIME}, \ldots\}$) iff every problem $P' \in \mathcal{C}$ can be reduced to P using a logspace reduction.

3 Explicit Finite-State Systems

Being inspired by the acceptance problem for ALBA (Alternating Linear Bounded Automata), and the related (alternating) graphs with configurations as nodes, we first define *Alternating Graph Problem* (AGP), a variant

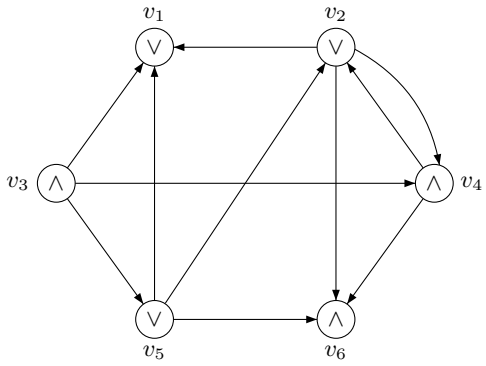

Fig. 2 Alternating graph

of the well known alternating reachability problem [6], which is PTIME-complete. Then we show a logspace reduction which, given an instance of AGP, constructs an FS $\mathcal{T}$ with two distinguished states s, s' so that:

- $s \sim s'$ if the answer to the AGP instance is NO,
- $s \not\sqsubseteq_{tr} s'$ if the answer to the AGP instance is YES.

This implies that FS-REL$_{\mathcal{R}}$ is PTIME-hard for any relation $\mathcal{R}$ such that $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$ (Theorem 8). Note that we reduce AGP to the complement of FS-REL$_{\mathcal{R}}$ and then use Observation 1.

Remark. Our construction here (providing an alternative proof of PTIME-hardness) is different from that in [16]. Its main aim is to allow a natural derivation of EXPTIME-hardness results in the next section.

3.1 Alternating Graphs

In this subsection we define formally *Alternating Graph Problem (*AGP*)* which is a slight generalization of the well known alternating reachability problem [6]. We also introduce some notions concerning alternating graphs that will be useful in the later constructions.

An *alternating graph* is a finite directed graph where each node is labelled either as *conjunctive* (universal) or as *disjunctive* (existential). Formally it is a structure $G = (V, E, t)$ where V is a finite set of *nodes*, $E \subseteq V \times V$ is a set of *edges*, and $t : V \to \{\wedge, \vee\}$ is a *node-labelling* function partitioning V into the sets of conjunctive and disjunctive nodes.

See Figure 2 for an example of an alternating graph.

We use $\sigma(v)$ to denote the set of *successors* of a node v, i.e.,

$$\sigma(v) = \{v' \in V \mid (v, v') \in E\}\,.$$

Each conjunctive node v ($t(v) = \wedge$) with $\sigma(v) = \emptyset$ is called *accepting*, each disjunctive node v ($t(v) = \vee$) with $\sigma(v) = \emptyset$ is called *rejecting*.

For example, v_6 is accepting and v_1 is rejecting in Figure 2.

We now define the set SUCC_G (or SUCC when G is clear from context) of *successful nodes* (i.e. those from which accepting nodes are 'alternation-reachable'). At the same time we also define a mapping $rank : \text{SUCC} \to \mathbb{N}$ which is naturally related to the inductive definition of SUCC.
SUCC is defined as the *least* subset of V satisfying (for each $v \in V$):

- if $t(v) = \wedge$ and $\sigma(v) \subseteq \text{SUCC}$ then $v \in \text{SUCC}$,
- if $t(v) = \vee$ and $\sigma(v) \cap \text{SUCC} \neq \emptyset$ then $v \in \text{SUCC}$.

The mapping *rank* is determined by the following (inductive) conditions:

- for a conjunctive $v \in \text{SUCC}$ ($t(v) = \wedge$), $rank(v) = 1 + \max\{rank(v') \mid v' \in \sigma(v)\}$, where we assume $\max \emptyset = 0$;
- for a disjunctive $v \in \text{SUCC}$ ($t(v) = \vee$), $rank(v) = 1 + \min\{rank(v') \mid v' \in \sigma(v) \cap \text{SUCC}\}$.

We note that each accepting node is successful (belongs to SUCC) and its rank is 1; on the other hand, each rejecting node is unsuccessful (i.e., not successful).

For example, for the alternating graph in Figure 2 we have $\text{SUCC} = \{v_2, v_4, v_5, v_6\}$, $rank(v_6) = 1$, $rank(v_2) = rank(v_5) = 2$, and $rank(v_4) = 3$.

Now we define *Alternating graph problem* (AGP):

INSTANCE: An alternating graph $G = (V, E, t)$ and a node $v \in V$.
QUESTION: Is v successful?

Remark. In the alternating reachability problem as defined in [6] it is required that there is exactly one accepting node in G and also no loops are allowed.

Problem AGP is PTIME-hard since it is a generalization of the alternating reachability problem which is known to be PTIME-hard (see for example [6] for a proof). It is clear from the given inductive definition that SUCC can be computed in polynomial time (measured in the size of the given G), and AGP is thus in PTIME. Hence we have the following fact.

Fact 2 AGP *is* PTIME*-complete.*

3.2 Reducing AGP to equivalence (and preorder) problems

Given an alternating graph $G = (V, E, t)$, with a rejecting node z, we aim at constructing an LTS $\mathcal{T}_G = (V, Act, \longrightarrow)$ (where the states correspond to

the nodes of G) so that $z \sim v$ for each $v \in V - \text{SUCC}_G$ and $z \not\sqsubseteq_{tr} v$ for each $v \in \text{SUCC}_G$.
We describe the construction of $\mathcal{T}_G$ (i.e. of Act and $\longrightarrow$) in stepwise manner:

- For each $v \in V$ we define a set of actions $Act(v)$:
  - When $t(v) = \wedge$, $Act(v) = \{c_v\}$.
  - When $t(v) = \vee$, $Act(v) = \{d_{(v,v')} \mid v' \in \sigma(v)\}$.
    (Note that $Act(v) = \emptyset$ when v is rejecting.)
- $Act = \bigcup_{v \in V} Act(v)$.
- The transition relation $\longrightarrow$ is defined so that for each $v, u, u' \in V$:

$$reach(v, c_u) = \begin{cases} \sigma(u) & \text{if } u = v \\ \sigma(u) \cup \{v\} & \text{if } u \neq v \end{cases}$$

$$reach(v, d_{(u,u')}) = \begin{cases} \{u'\} & \text{if } u = v \\ \{v\} & \text{if } u \neq v \end{cases}$$

(Recall that $reach(v, a) = \{v' \in V \mid v \xrightarrow{a} v'\}$ for $v \in V$, $a \in Act$.)

The construction can be equivalently described as follows:

a) For each edge $(v, v') \in E$ there is a transition $v \xrightarrow{a} v'$ labelled with $a = c_v$ when $t(v) = \wedge$ or with $a = d_{(v,v')}$ when $t(v) = \vee$.
b) For each $v \in V$ and $a \in Act - Act(v)$ there is a loop $v \xrightarrow{a} v$.
c) For each $u, u', v \in V$ such that $t(u) = \wedge$ and $(u, u') \in E$ there is a transition $v \xrightarrow{c_u} u'$.

Observation 3 *Logarithmic workspace is sufficient for the construction of $\mathcal{T}_G$ from G.*

As an example, LTS $\mathcal{T}_G$ constructed for the graph G in Figure 2 is depicted in Figure 3. In this figure we write c_i and $d_{i,j}$ instead of c_{v_i} and $d_{(v_i,v_j)}$. The sets of actions are $Act(v_1) = \emptyset$, $Act(v_2) = \{d_{2,1}, d_{2,4}, d_{2,6}\}$, $Act(v_3) = \{c_3\}$, $Act(v_4) = \{c_4\}$, $Act(v_5) = \{d_{5,1}, d_{5,2}, d_{5,6}\}$, $Act(v_6) = \{c_6\}$. Transitions specified by item c) above are omitted in the figure. To complete the figure, there should be the following transitions for each $v \in V$:

$$v \xrightarrow{c_3} v_1 \qquad v \xrightarrow{c_3} v_4 \qquad v \xrightarrow{c_3} v_5 \qquad v \xrightarrow{c_4} v_2 \qquad v \xrightarrow{c_4} v_6$$

Observation 4 *Action $a \in Act$ is disabled in state v of $\mathcal{T}_G$ iff $a = c_v$ and v is accepting in G (i.e. $t(v) = \wedge$ and $\sigma(v) = \emptyset$).*

Proposition 5 *Let v be an unsuccessful node (i.e., $v \in V - \text{SUCC}_G$) and $a \in Act$ an action:*

1. *There is some unsuccessful v' (maybe $v' = v$) such that $v \xrightarrow{a} v'$ in $\mathcal{T}_G$. (Hence $Traces(v) = Act^*$ for each unsuccessful v.)*

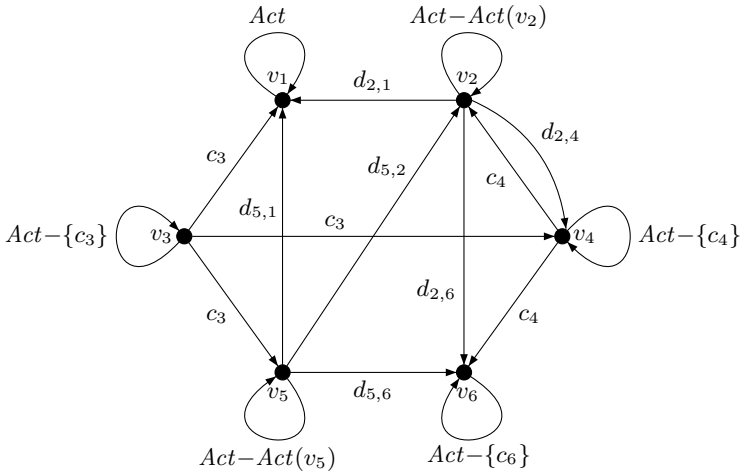

Fig. 3 Example of $\mathcal{T}_G$ (with some transitions omitted)

2. *If* $v \xrightarrow{a} v'$ *in* $\mathcal{T}_G$ *where* $v' \in \text{SUCC}_G$ *then* $a = c_u$ *for some (conjunctive)* u *(maybe* $v = u$*) and we have* $v'' \xrightarrow{a} v'$ *for each* $v'' \in V$*.*

Proof (1) It follows directly from considering the cases in the definition of $reach(v, a)$ using the fact that if v is unsuccessful, all nodes in $\sigma(v)$ are unsuccessful when $t(v) = \vee$ and there is at least one unsuccessful node in $\sigma(v)$ when $\sigma(v) = \wedge$.

(2) Suppose $v \xrightarrow{a} v'$ for some unsuccessful v and successful v'. If $a = d_{(v,v')}$ then $t(v) = \vee$ and $v' \in \text{SUCC}_G$ implies $v \in \text{SUCC}_G$ which is impossible. Also $v \neq v'$ (since $v \notin \text{SUCC}_G$ and $v' \in \text{SUCC}_G$) so $v \xrightarrow{a} v'$ is not a loop. The only remaining possibility is $a = c_u$ for some (conjunctive) u. □

Proposition 6 *If* $v_1, v_2 \in V$ *are unsuccessful in* G *then* $v_1 \sim v_2$ *in* $\mathcal{T}_G$*.*

Proof It is sufficient to show that the relation

$$\mathcal{R} = \{(v, v') \mid v, v' \in V - \text{SUCC}_G\} \cup \{(v, v) \mid v \in V\}$$

is a bisimulation. For each pair $(v, v') \in \mathcal{R}$ it is easy to check that any transition in v can be matched by some transition in v' and vice versa. This is trivial for $v = v'$, so suppose $v \neq v'$ and $v, v' \in V - \text{SUCC}_G$ and consider some transition $v \xrightarrow{a} v_1$. If $v_1 \notin \text{SUCC}_G$ then by Proposition 5 (1) there is some $v_1' \in V - \text{SUCC}_G$ such that $v' \xrightarrow{a} v_1'$ and we have $(v_1, v_1') \in \mathcal{R}$. If $v_1 \in \text{SUCC}_G$ then $a = c_u$ for some u and $v'' \xrightarrow{a} v_1$ for each $v'' \in V$ by Proposition 5 (2), in particular $v' \xrightarrow{a} v_1$. The remaining cases are symmetric. □

Proposition 7 *For each* G *there is (a fixed)* $w \in Act^*$ *such that* $w \notin Traces(v)$ *in* $\mathcal{T}_G$ *for any* $v \in \text{SUCC}_G$*.*

Proof We attach the *witness action* to each $v \in \text{SUCC}_G$:

- if v is conjunctive then its witness action is c_v,
- if v is disjunctive then we choose some $d_{(v,v')}$ for which $\mathit{rank}(v) = 1 + \mathit{rank}(v')$ as its witness action ($v' \in \text{SUCC}_G$).

We now order the elements of SUCC_G into a sequence $v_1, v_2, \ldots, v_m$ so that $i \leq j$ implies $\mathit{rank}(i) \leq \mathit{rank}(j)$, and we use a_i to denote the witness action of v_i ($1 \leq i \leq m$). We put $w = a_m a_{m-1} \cdots a_1$. To show that $w \notin \mathit{Traces}(v)$ for any $v \in \text{SUCC}_G$ we use the following claim that can be then easily verified (note that if v_i is a successful conjunctive node then each $v' \in \sigma(v_i)$ is successful and $v' = v_k$ for some $k < i$):

Claim. If $v_j \xrightarrow{a_i} v'$ for $j \leq i$ then $v' \in \text{SUCC}_G$ and $v' = v_k$ for some $k < i$.

(The claim generalizes the fact that $\mathit{reach}(v_1, a_1) = \emptyset$; recall Observation 4 and note that v_1 must be accepting in G.) Hence $\mathit{reach}(v, a_m a_{m-1} \cdots a_i) \subseteq \{v_{i-1}, v_{i-2}, \ldots, v_1\}$ for any $v \in \text{SUCC}_G$, which also means that $w = a_m a_{m-1} \ldots a_1$ is disabled in all $v \in \text{SUCC}_G$. □

Theorem 8 FS-REL$_\mathcal{R}$ *is* PTIME*-hard for any* $\mathcal{R}$ *such that* $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$.

Proof Given an alternating graph G and a distinguished node x (i.e., an instance of AGP), we can choose (or add) a rejecting node z and construct $\mathcal{T}_G$ (by the above logspace construction). Proposition 6 guarantees that if $x \notin \text{SUCC}_G$ then $z \sim x$ and thus $(z, x) \in \mathcal{R}$. Proposition 5 (1) and Proposition 7 guarantee that if $x \notin \text{SUCC}_G$ then $z \not\sqsubseteq_{tr} x$ and thus $(z, x) \notin \mathcal{R}$.
We have thus indeed shown the reduction announced at the beginning of this section. □

4 Composed Systems

The main result presented in this section is EXPTIME-hardness of PCH-REL$_\mathcal{R}$ for all relations $\mathcal{R}$ such that $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$. This is achieved by a logspace reduction from the problem ALBA-ACCEPT, the problem whether a given alternating linear bounded automaton accepts a given word, which is known to be EXPTIME-complete [2]. In fact, we also consider so called reactive linear bounded automata (RLBAs) as a technically convenient intermediate step, and derive EXPTIME-completeness for RLBA-REL$_\mathcal{R}$ as well.

4.1 Alternating Linear Bounded Automata

We start with recalling the notion of (standard) linear bounded automata. A (nondeterministic) *linear bounded automaton* (LBA) is a tuple

$$A = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$

where Q is a finite set of control states, Σ is an input alphabet, $\Gamma \supseteq \Sigma$ is a tape alphabet, $\delta \subseteq (Q - \{q_{acc}, q_{rej}\}) \times \Gamma \times Q \times \Gamma \times \{-1, 0, +1\}$ is a set of transitions, $q_0, q_{acc}, q_{rej} \in Q$ are the initial state, the accepting state and the rejecting state, respectively. The alphabet Γ contains special symbols $\vdash, \dashv \notin \Sigma$ which play the role of the endmarkers.

A *configuration* of A is a triple $\alpha = (q, w, i)$ where q is a control state, $w = \vdash a_1 a_2 \cdots a_n \dashv$ (where $a_i \in \Gamma - \{\vdash, \dashv\}$) is the tape content, and $0 \leq i \leq n+1$ is the head position.

A configuration $\alpha' = (q', w', i')$ is a *successor* of $\alpha = (q, w, i)$, written $\alpha \vdash_A \alpha'$ (or just $\alpha \vdash \alpha'$ when A is obvious), iff there is $(q, a, q', a', d) \in \delta$ such that w contains a on position i, w' is obtained from w by writing a' on position i, and $i' = i + d$. We stipulate that the endmarkers may not be overwritten, and the transitions are constrained so that the head never moves left from $\vdash$ nor right from $\dashv$.

We define the *length* $|\alpha|$ of configuration $\alpha = (q, \vdash u \dashv, i)$ as $|\alpha| = |u|$. Obviously $|\alpha| = |\alpha'|$ when $\alpha \vdash \alpha'$. The set of all configurations of A of length n is denoted by

$$Conf_{(A,n)}\,.$$

Given an LBA A and $n \in \mathbb{N}$, there is the corresponding graph $G_{(A,n)} = (V, E)$ where $V = Conf_{(A,n)}$ and $E \subseteq V \times V$ contains an edge (α, α') iff $\alpha \vdash_A \alpha'$.

The *initial configuration* for an input $u \in \Sigma^*$ is $\alpha^A_{ini}(u) = (q_0, \vdash u \dashv, 1)$. A configuration (q, w, i) is *accepting* iff $q = q_{acc}$, and *rejecting* iff $q = q_{rej}$. LBA A *accepts* $u \in \Sigma^*$ iff some accepting $\alpha \in Conf_{(A,|u|)}$ is reachable from $\alpha^A_{ini}(u)$ in $G_{(A,|u|)}$.

An *alternating LBA* (ALBA) is an LBA extended with a function

$$t : Q \to \{\wedge, \vee\}$$

that labels each control state as either conjunctive or disjunctive; we stipulate $t(q_{acc}) = \wedge$, $t(q_{rej}) = \vee$. The mapping t is extended to configurations: for $\alpha = (q, w, i)$ we put $t(\alpha) = t(q)$.

An ALBA A and $n \in \mathbb{N}$ thus determine the alternating graph

$$G_{(A,n)} = (V, E, t)$$

which arises from $G_{(A,n)}$ of the underlying LBA by adding the mapping t (determined by A). Note that accepting (resp. rejecting) configurations of A are accepting (resp. rejecting) nodes in $G_{(A,n)}$.

It is convenient to define acceptance as follows:

$$\text{ALBA } A \textit{ accepts } w \in \Sigma^* \text{ iff } \alpha^A_{ini}(w) \text{ is successful in } G_{(A,|w|)}.$$

Problem ALBA-ACCEPT

INSTANCE: An ALBA A and a word $w \in \Sigma^*$.
QUESTION: Does A accept w?

is well-known to be EXPTIME-complete [2]. By $|(A, w)|$ we denote the size of a standard description of instance (A, w).

4.2 Idea of reducing ALBA-ACCEPT to PCH-REL$_{\mathcal{R}}$

Let (A, w) be an instance of ALBA-ACCEPT where $A = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}, t)$, $w \in \Sigma^*$ and $|w| = n$.
If we consider $G_{(A,n)}$ (of exponential size) and then apply the construction of Subsection 3.2, we obtain an (explicit) FS, denoted $\mathcal{T}_{(A,n)}$. Both the set of nodes of $G_{(A,n)}$ and the set of states of $\mathcal{T}_{(A,n)}$ coincide with $Conf_{(A,n)}$; if α_{rej} is a rejecting configuration in $Conf_{(A,n)}$ then we have (in $\mathcal{T}_{(A,n)}$):

- $\alpha_{rej} \not\sqsubseteq_{tr} \alpha^A_{ini}(w)$ when A accepts w;
- $\alpha_{rej} \sim \alpha^A_{ini}(w)$ when A does not accept w.

Our aim is, when given (A, w), to construct a PCH denoted $M_{(A,n)}$ which will represent ('realize') $\mathcal{T}_{(A,n)}$; moreover, we aim at a construction of $M_{(A,n)}$ that can be done in logarithmic space. (Note that the logspace construction will guarantee that the size of $M_{(A,n)}$ is polynomial in $|(A, w)|$.)

We know that a PCH can easily represent an LTS of exponential size; nevertheless a technical problem is that not only the number of states in $\mathcal{T}_{(A,n)}$ (i.e., the cardinality of $Conf_{(A,n)}$) is exponential but it is the case also for the alphabet $Act_{(A,n)}$ of $\mathcal{T}_{(A,n)}$; recall that $Act_{(A,n)}$ contains symbols of the types c_α and $d_{(\alpha,\alpha')}$ where $\alpha, \alpha' \in Conf_{(A,n)}$.

To handle the alphabet-cardinality problem, we choose some straightforward (injective) encoding

$$repr : Act_{(A,n)} \to \{0,1\}^m \tag{1}$$

which encodes all elements of $Act_{(A,n)}$ by $0,1$-*strings of the same length* m ($\forall a \in Act_{(A,n)} : |repr(a)| = m$) polynomial in $|(A, w)|$. The details of *repr* are not important. It is sufficient to require that decoding, including checking if $x \in \{0,1\}^m$ is in the range of *repr*, can be computed easily, which means that it is performable by the reactive LBA $B_{(A,n)}$ described later.

The LTS represented by $M_{(A,n)}$ will be finer than $\mathcal{T}_{(A,n)}$, having more states, but its action alphabet will be just $\{0, 1, \tau\}$. For each state α of $\mathcal{T}_{(A,n)}$ there will be a corresponding global state $corresp(\alpha)$ of $M_{(A,n)}$, and in the LTS represented by $M_{(A,n)}$ the following will be ensured:

- $corresp(\alpha) \sim corresp(\alpha')$ iff $\alpha \sim \alpha'$ in $\mathcal{T}_{(A,n)}$;
- $corresp(\alpha) \not\sqsubseteq_{tr} corresp(\alpha')$ iff $\alpha \not\sqsubseteq_{tr} \alpha'$ in $\mathcal{T}_{(A,n)}$.

Each transition $\alpha \xrightarrow{a} \alpha'$ in $\mathcal{T}_{(A,n)}$ will be 'simulated' by a sequence of transitions $corresp(\alpha) \xrightarrow{u} corresp(\alpha')$ performed by $M_{(A,n)}$ where u arises from the sequence $repr(a) \in \{0,1\}^m$ by a suitable 'padding' with (occurrences of)

action τ; moreover, for all transitions $\alpha \xrightarrow{a} \alpha'$, the corresponding sequences u will have the same length, i.e. $|u| = \ell$ for a constant $\ell > m$.

It is technically convenient to specify the intended behaviour of $M_{(A,n)}$ by means of a special LBA, called a reactive LBA, and then show how reactive LBAs can be naturally implemented by PCH. This is done in the following two subsections.

4.3 Reducing ALBA-ACCEPT to RLBA-REL$_{\mathcal{R}}$

We can imagine that each step $(\alpha \vdash \alpha')$ in computations of an LBA also comprises 'emitting' action τ; so each computation can be 'observed' as a sequence of τ-actions by an 'external observer'. A *reactive linear bounded automaton*, an *RLBA*, is an LBA which is moreover equipped with a set Act of non-τ actions ($\tau \notin Act$); each step now emits either τ or $a \in Act$. For technical convenience, we require that the transitions emitting a non-τ action depend only on the current control state and do not change the tape nor the head position. RLBAs serve us for describing behaviours, not for accepting (rejecting) inputs; therefore we do not need an input alphabet nor accepting/rejecting states in the following formal definition.

An RLBA is a tuple $B = (Q_c, Q_r, \Gamma, Act, \delta, R)$, where Q_c and Q_r are finite sets of *computational control states* and *reactive control states*, respectively ($Q_c \cap Q_r = \emptyset$), Γ is a tape alphabet, Act is a finite set of actions, $\tau \notin Act$, $\delta \subseteq Q_c \times \Gamma \times Q \times \Gamma \times \{-1, 0, +1\}$, where $Q = Q_c \cup Q_r$, is the set of *computational transitions*, and $R \subseteq Q_r \times (Act \cup \{\tau\}) \times Q$ is the set of *reactive transitions*. Configurations and the successor relation $\alpha \vdash_B \alpha'$ are defined as in the case of LBAs (where $(q, a, q') \in R$ is understood as if $(q, x, q', x, 0) \in \delta$ for all $x \in \Gamma$).

Given $n \in \mathbb{N}$, RLBA B determines the LTS

$$\mathcal{T}(B, n) = (Conf_{(B,n)}, Act \cup \{\tau\}, \longrightarrow)$$

where $Conf_{(B,n)}$ is the set of all configurations of B of size n, and $\longrightarrow$ contains a transition $(q, w, i) \xrightarrow{a} (q', w', i')$ iff

- either $q \in Q_c$, $(q, w, i) \vdash_B (q', w', i')$ and $a = \tau$,
- or $q \in Q_r$, $(q, a, q') \in R$, $w = w'$ and $i = i'$.

It is now crucial to observe that, given an ALBA A with an input word w, $|w| = n$, we can construct an RLBA $B_{(A,n)}$ described as follows. Here we assume that *each configuration of A has at most two successors* (which is no real restriction, in fact), though the construction can be easily adapted to handle any fixed number of possible successors.

$B_{(A,n)}$ has a tape of length $m + 2$ (m taken from (1)), with five tracks as sketched in Figure 4.

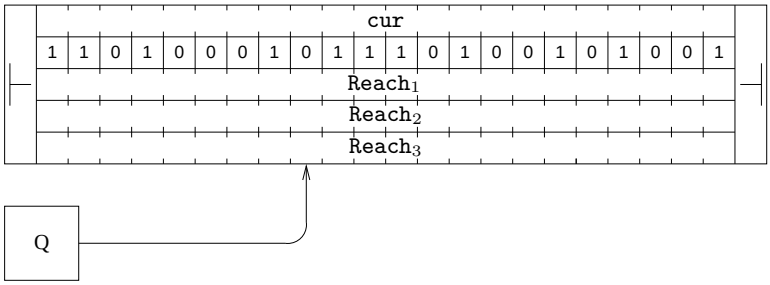

Fig. 4 RLBA $B_{(A,n)}$

Track 1 serves for storing (the description of) a configuration of A; we might view Track 1 as a variable "`cur`" (meaning 'current'). Track 2, viewed as a variable "`act`" serves for storing $x \in \{0,1\}^m$ which might (or might not) coincide with $repr(a)$ for an action a of $\mathcal{T}_{(A,n)}$. Tracks 3,4,5 will serve for storing the configurations (nodes of $\mathcal{T}_{(A,n)}$) to which the transitions labelled with $repr^{-1}(\texttt{act})$ lead from `cur` in $\mathcal{T}_{(A,n)}$.

$B_{(A,n)}$ performs the (possibly infinite) loop described by the pseudocode in Figure 5.

Step 1: (Track 1 contains (the description of) a configuration `cur` of A. Now the content of Track 2, referred to as `act`, is generated:)

for $i := 1$ **to** m **do**
  nondeterministically select and emit action $y \in \{0,1\}$;
  `act`$[i] := y$ (i.e., write y on the i-th position of Track 2).

Step 2: (Compute the value of `Reach`, i.e., write (the descriptions of) the nodes of $\mathcal{T}_{(A,n)}$ reachable from `cur` by $repr^{-1}(\texttt{act})$ to (some of) Tracks 3, 4, 5:)

when `act` is $repr(c_\beta)$ for some β such that $t(\beta) = \wedge$:
  if `cur` $= \beta$ **then**
    `Reach` $:= \{\beta' \mid \beta \vdash_A \beta'\}$
  else
    `Reach` $:= \{\beta' \mid \beta \vdash_A \beta'\} \cup \{\texttt{cur}\}$
when `act` is $repr(d_{(\beta,\beta')})$ for some β, β' such that $t(\beta) = \vee$ and $\beta \vdash_A \beta'$:
  if `cur` $= \beta$ **then**
    `Reach` $:= \{\beta'\}$
  else
    `Reach` $:= \{\texttt{cur}\}$
otherwise (when `act` is not $repr(a)$ for any $a \in Act_{(A,n)}$):
  `Reach` $:= \emptyset$

Step 3: (Select a new state from `Reach`:)

nondeterministically select $\alpha \in$ `Reach` (from those of Tracks 3,4,5 which have been rewritten in the previous step; halt if `Reach` $= \emptyset$);
`cur` $:= \alpha$ (i.e., copy α to Track 1)
goto Step 1

Fig. 5 Behaviour of RLBA $B_{(A,n)}$

Without giving further technical details, it should be clear that the activity described by the pseudocode is indeed realizable by an RLBA, and moreover, such $B_{(A,n)}$ can be constructed from (A, w) using only logarithmic workspace. To see that logarithmic workspace is sufficient for this construction, just note that this can be done using some fixed number of pointers pointing into the instance (A, w) of ALBA-ACCEPT and some fixed number of counters bounded by the size of (A, w). When configurations of A are encoded in a standard way (as sequences of the tape symbols representing the tape content with the control state added to the symbol on the current position of the head, all this encoded in binary), the structure of instructions of $B_{(A,n)}$ is quite regular and most of these instructions depend only on n and Q and Δ in the definition of A (in fact only on the sizes of these sets). The only place where δ (from the definition of A) plays some role is the computation of `Reach` where the set $\{\beta' \mid \beta \vdash_A \beta'\}$ must be constructed for β stored on Track 1. It is obvious that instructions of $B_{(A,n)}$ that perform this computation can be constructed by processing tuples in δ in the definition of A one by one for which a fixed number of pointers is sufficient.

The only non-τ actions emitted during a computation of $B_{(A,n)}$ are actions 0,1 emitted in Step 1. Each reactive state q causing such emitting can be assumed deterministic, in the sense that there is just one q_1 and just one q_2 such that $(q, 0, q_1), (q, 1, q_2) \in R$. Otherwise the computation is fully deterministic with the only exception of the first computational step in Step 3.

We can easily observe that the number of steps performed in any iteration of the loop can be made constant (by padding the shorter computations with τ-actions), and we can also make constant the number of steps after which the nondeterministic choice in Step 3 is reached.

For any $\alpha \in \mathcal{T}_{(A,n)}$ we define $\mathit{corresp}(\alpha)$ as the state in the LTS generated by $B_{(A,n)}$ which corresponds to the initial state (the beginning of Step 1) with $\mathtt{cur} = \alpha$ (i.e., α written in Track 1).

Proposition 9 *In the LTS generated by $B_{(A,n)}$ (i.e., in $\mathcal{T}(B_{(A,n)}, m)$) we have:*

- $\mathit{corresp}(\alpha) \sim \mathit{corresp}(\alpha')$ *iff* $\alpha \sim \alpha'$ *in* $\mathcal{T}_{(A,n)}$;
- $\mathit{corresp}(\alpha) \not\sqsubseteq_{tr} \mathit{corresp}(\alpha')$ *iff* $\alpha \not\sqsubseteq_{tr} \alpha'$ *in* $\mathcal{T}_{(A,n)}$.

Proof idea Due to the above described tight simulation of transitions in $\mathcal{T}_{(A,n)}$ by the loop in Figure 5, verification of the claim is straightforward. (The bisimulation game on $\mathcal{T}_{(A,n)}$ is tightly mimicked in the game on the configurations of $B_{(A,n)}$; Player I can gain nothing by generating a sequence in $\{0, 1\}^m$ outside the range of *repr*. Recall that transitions of $B_{(A,n)}$ are deterministic in the sense that they depend only on the (visible) action performed with the only exception of the step when nondeterministic choice of Step 3 is performed, and this step is always reached at the same time in both configurations.) □

For the problem RLBA-REL$_{\mathcal{R}}$

INSTANCE: An RLBA B and two configurations α, α', $|\alpha| = |\alpha'|$.
QUESTION: Is $(\alpha, \alpha') \in \mathcal{R}$ in $\mathcal{T}(B, |\alpha|)$?

we have thus derived the following theorem.

Theorem 10 RLBA-REL$_{\mathcal{R}}$ *is* EXPTIME*-hard for any* $\mathcal{R}$ *such that* $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$.

Proof For each such $\mathcal{R}$ there is a logspace reduction from EXPTIME-complete problem ALBA-ACCEPT to RLBA-REL$_{\mathcal{R}}$: given an ALBA A and w, we construct the above described RLBA $B_{(A,n)}$ with two configurations $corresp(\alpha^A_{ini}(w))$ and $corresp(\alpha^A_{rej})$, where α_{rej} is a rejecting configuration in $G_{(A,n)}$. The rest follows similarly as in the proof of Theorem 8. □

4.4 Implementation of RLBA by PCH

An RLBA can be implemented by a PCH in a straightforward way, similarly as, e.g., Rabinovich [14] did for LBA.

Lemma 11 *Given an RLBA* B *and* $n \in \mathbb{N}$*, logarithmic workspace is sufficient to construct a centralized PCH* $P_{(B,n)}$ *which represents an LTS isomorphic to* $\mathcal{T}(B, n)$.

Proof Assume $B = (Q_c, Q_r, \Gamma, Act, \delta, R)$. The constructed PCH $P_{(B,n)}$ will be of the form

$$hide\ \mathcal{B}\ in\ (\mathcal{T}_c \parallel \mathcal{T}_0 \parallel \mathcal{T}_1 \parallel \cdots \parallel \mathcal{T}_{n+1})$$

where $\mathcal{T}_c$ is a control component used to model the control unit of B and to store the head position and where each of $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{n+1}$ models one individual cell of the tape of B. (We assume that the tape cells are numbered by $0, 1, \ldots, n+1$, the endmarkers being at positions 0 and $n+1$.)

The state set of $\mathcal{T}_i$ $(0 \leq i \leq n+1)$ is Γ (the current state of $\mathcal{T}_i$ represents the current content of cell i), its action alphabet is $Act_i = \{\langle b, b', i\rangle \mid b, b' \in \Gamma\}$, and there is a transition $b \xrightarrow{\langle b,b',i\rangle} b'$ for each $b, b' \in \Gamma$.

The state set of $\mathcal{T}_c$ is $\{\langle q, i\rangle \mid q \in Q, 0 \leq i \leq n+1\}$ and its action alphabet is

$$Act_c = Act \cup Act_0 \cup Act_1 \cup \cdots \cup Act_{n+1}$$

(for Act, taken from B, we assume $Act \cap Act_i = \emptyset$ for each i). For each $(q, b, q', b', d) \in \delta$ (where $q \in Q_c$) and i such that $0 \leq i \leq n+1$ and $0 \leq i + d \leq n+1$ there is a transition

$$\langle q, i\rangle \xrightarrow{\langle b,b',i\rangle} \langle q', i+d\rangle\,,$$

and for each $(q, a, q') \in R$ (where $q \in Q_r$, $q' \in Q$, and $a \in Act \cup \{\tau\}$) there are transitions $\langle q, i\rangle \xrightarrow{a} \langle q', i\rangle$ for all i, $0 \leq i \leq n+1$.

The set of actions hidden in $P_{(B,n)}$ is $\mathcal{B} = Act_0 \cup Act_1 \cup \cdots \cup Act_{n+1}$.

Each configuration $\alpha = (q, \vdash a_1 a_2 \cdots a_n \dashv, i)$ of B is naturally represented as the global state

$$E_\alpha = (\langle q, i \rangle, \vdash, a_1, a_2, \ldots, a_n, \dashv)$$

of $P_{(B,n)}$. We note that $\alpha \xrightarrow{a} \alpha'$ iff $E_\alpha \xrightarrow{a} E_{\alpha'}$. The required isomorphism between LTSs can be obviously achieved; to this aim we restrict the state set of $\mathcal{T}_0$ to $\{\vdash\}$ etc.

Finally we note that logarithmic workspace is sufficient for the construction of $P_{(B,n)}$. □

From Theorem 10 we thus get the following theorem.

Theorem 12 *The problem* PCH-REL$_\mathcal{R}$ *is* EXPTIME*-hard for any* $\mathcal{R}$ *such that* $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$ *even for centralized PCH.*

Remark. The construction can be also used for similar proofs (of EXPTIME-hardness) for other types of composed systems which use other means of synchronization, as, e.g., labelled 1-safe Petri nets.
The lower bound of EXPTIME-hardness can not be improved in general; this follows from the results surveyed in Section 6. ('Simulation-like' equivalences are in EXPTIME.)
We also note that hiding is crucial. We can not hope for general EXPTIME-hardness in the case of PC since the trace preorder and 'trace-like' equivalences are in PSPACE for them (see Section 6).

5 Acyclic PC and PCH

5.1 DP-hardness for acyclic PCH

Problem APCH-REL$_\mathcal{R}$ (for acyclic PCH) is stated in [14] as NP-hard and coNP-hard for each $\mathcal{R}$, $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$. There is a proof of coNP-hardness and it is mentioned that showing NP-hardness is similar (though it is, in fact, a bit more complicated); also a modification for coNP-hardness of APC-REL$_\mathcal{R}$ (without hiding) is suggested. Here we show a different (simpler) construction which allows us to derive all the mentioned cases as well as DP-hardness for APCH-REL$_\mathcal{R}$. (We recall that a problem is in DP iff the set of its positive instances is the intersection of the sets of positive instances of two problems, one being in NP and the other in coNP.)

Theorem 13 *For any relation* $\mathcal{R}$ *such that* $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$*:*

- APCH-REL$_\mathcal{R}$ *is* DP*-hard for acyclic and centralized PCH.*
- APC-REL$_\mathcal{R}$ *is* coNP*-hard for acyclic and centralized PC.*

Proof In [14], the well-known NP-complete problem SAT was used; we start by recalling another NP-complete problem which seems more suitable to our aims. Given words $u, v \in \Sigma^*$, by *shuffle*(u, v) we denote the result of merging, or interleaving, i.e. the set of words of the form $u_1v_1u_2v_2 \cdots u_nv_n$ where u_i, v_i are (possibly empty) words from Σ^* such that $u = u_1u_2 \cdots u_n$ and $v = v_1v_2 \ldots v_n$. The operation can be naturally generalized to languages, *shuffle*$(L_1, L_2) = \bigcup_{u \in L_1, v \in L_2}$ *shuffle*(u, v), and the use *shuffle*$(w_1, w_2, \ldots, w_k)$ with more arguments has the obvious meaning. The following problem is known to be NP-complete [10,19]:

PROBLEM: SHUFFLE
INSTANCE: Words $w_1, w_2, \ldots, w_k \in \Sigma^*$ and $w \in \Sigma^*$ such that $|w_1| + |w_2| + \cdots + |w_k| = |w|$ (for some finite alphabet Σ).
QUESTION: Is it true that $w \in$ *shuffle*$(w_1, w_2, \ldots, w_k)$?

We start with showing NP-hardness of APCH-REL$_{\mathcal{R}}$. Given an instance $w_1, w_2, \ldots, w_k, w \in \Sigma^*$ of SHUFFLE, we construct a pair of acyclic centralized PCH $\mathcal{T}$, $\mathcal{T}'$ with initial global states E_0 and E'_0 so that:

- $E_0 \sim E'_0$ when $w \in$ *shuffle*$(w_1, w_2, \ldots, w_k)$ (the SHUFFLE-answer is YES),
- $E_0 \not\sqsubseteq_{tr} E'_0$ otherwise (the SHUFFLE-answer is NO).

The construction is obvious from Figure 6; the depicted fragment corresponds to a case where $\Sigma = \{a, b\}$, $w_1 = abaa$, $w_2 = bba$, $w_k = abab$, $w = abba \cdots ba$ and $|w| = n$. PCH $\mathcal{T}$ consists of one component (with $2n + 2$ states) as depicted in the figure. PCH $\mathcal{T}'$ has a component $\mathcal{S}_i$ for each word w_i, with alphabet $\Sigma_i = \{a_i \mid a \in \Sigma\}$ (the alphabets of $\mathcal{S}_i$ and $\mathcal{S}_j$ are disjoint when $i \neq j$); the control component $\mathcal{S}_c$ corresponds to w ($w = abba \cdots ba$) as depicted, its alphabet being $\Sigma_c = \Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_k \cup \{d\}$. PCH $\mathcal{T}'$ is defined as

$$\mathcal{T}' = \textit{hide } \mathcal{B} \textit{ in } (\mathcal{S}_c \parallel \mathcal{S}_1 \parallel \mathcal{S}_2 \parallel \cdots \parallel \mathcal{S}_k) \quad \text{where } \mathcal{B} = \Sigma_c - \{d\}.$$

The above announced global states are $E_0 = (s_0)$ and $E'_0 = (s'_0, v_{1,0}, v_{2,0}, \ldots, v_{k,0})$. We note that $\tau^n d \in \textit{Traces}(E_0)$ but $\tau^n d \in \textit{Traces}(E'_0)$ (i.e., a global state $(s'_n, \ldots)$ is reachable from E'_0) if and only if $w \in$ *shuffle*$(w_1, w_2, \ldots, w_k)$. This implies that $E_0 \not\sqsubseteq_{tr} E'_0$ when the answer for the instance of SHUFFLE is NO.

On the other hand, if $w \in$ *shuffle*$(w_1, w_2, \ldots, w_k)$ (the answer is YES) then $E_0 \sim E'_0$: in the bisimulation game, Player II can easily maintain during the play that s_n is reachable in $\mathcal{T}$ as long as $(s'_n, \ldots)$ is reachable in $\mathcal{T}'$ and vice versa. (Note that $E'_n = (s'_n, \ldots)$ is reachable from E'_0 and for each $E'_i = (s'_i, \ldots)$ (where $0 \leq i < n$) such that E'_n is reachable from E'_i there is some $E'_{i+1} = (s'_{i+1}, \ldots)$ such that $E'_i \xrightarrow{\tau} E'_{i+1}$ and E'_n is reachable from E'_{i+1}. Moreover, if $\mathcal{T}$ is in state (t_i) and $\mathcal{T}'$ in state $(t'_i, \ldots)$, the same number of τ-actions can performed on both sides but nothing else.)

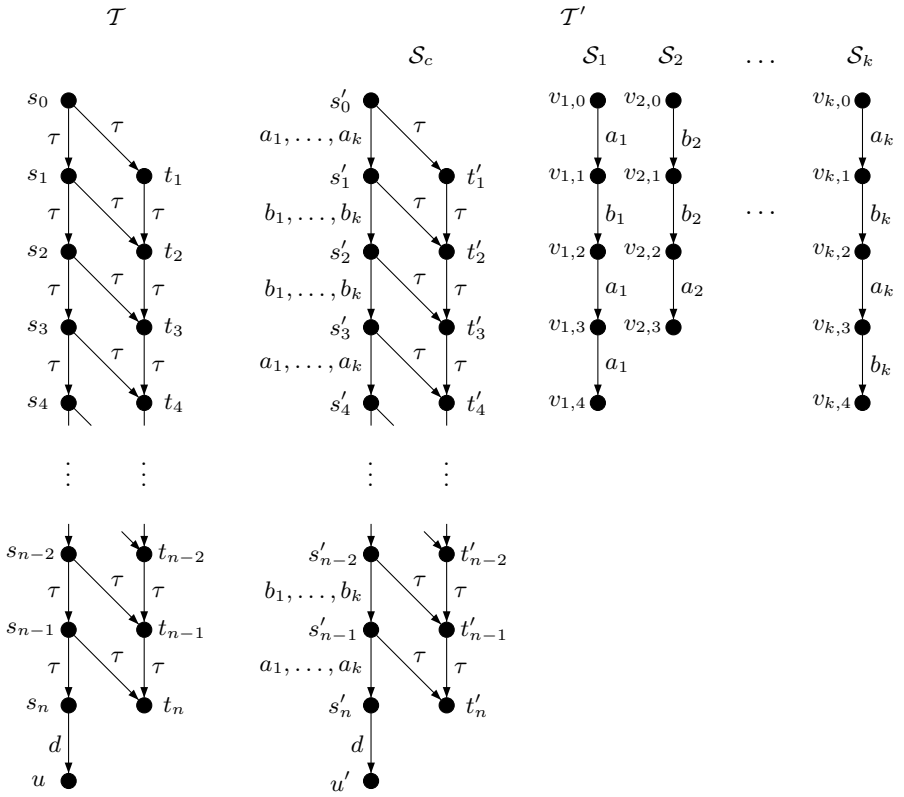

Fig. 6 $\mathcal{T}$ and $\mathcal{T}'$ for the proof of NP-hardness

Since logarithmic workspace is sufficient for constructing $\mathcal{T}, \mathcal{T}'$, we have established NP-hardness of APCH-REL$_{\mathcal{R}}$.

Remark. Formally, there is only one acyclic PCH in the instance of APCH-REL$_{\mathcal{R}}$. It is straightforward to construct this one PCH from $\mathcal{T}, \mathcal{T}'$ by taking their disjoint union (possibly with some components extended with some auxiliary states when necessary).

A proof of coNP-hardness of APC-REL$_{\mathcal{R}}$ (using no hiding) can look as follows. We take $\mathcal{T}'' = (\mathcal{S}''_c \parallel \mathcal{S}_1 \parallel \mathcal{S}_2 \parallel \cdots \parallel \mathcal{S}_k)$ and $\mathcal{T}''' = (\mathcal{S}'''_c \parallel \mathcal{S}_1 \parallel \mathcal{S}_2 \parallel \cdots \parallel \mathcal{S}_k)$ where $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ are the same as in the previous construction, $\mathcal{S}''_c$ is obtained from $\mathcal{S}_c$ by deleting all states $\{t'_1, t'_2, \ldots, t'_n\}$ (with the associated transitions), and $\mathcal{S}'''_c$ is obtained from $\mathcal{S}''_c$ by deleting the transition $s'_n \xrightarrow{d} u'$. Let E''_0, E'''_0 be the initial global states of $\mathcal{T}''$ and $\mathcal{T}'''$ defined analogously as E'_0 for $\mathcal{T}'$.

Obviously, $\tau^n d \notin \mathit{Traces}(E'''_0)$ but $\tau^n d \in \mathit{Traces}(E''_0)$ if and only if $w \in \mathit{shuffle}(w_1, w_2, \ldots, w_k)$. This implies that $E''_0 \not\sqsubseteq_{tr} E'''_0$ when the answer for

the instance of SHUFFLE is YES. On the other hand, if the answer is NO then obviously $E_0'' \sim E_0'''$.

For showing DP-hardness of APCH-REL$_{\mathcal{R}}$, we can naturally use a reduction from the following DP-complete problem.

PROBLEM: SHUFFLE-NONSHUFFLE
INSTANCE: Words $u_1, u_2, \ldots, u_k, u \in \Sigma^*$ and $v_1, v_2, \ldots, v_k, v \in \Delta^*$, where $\Sigma \cap \Delta = \emptyset$.
QUESTION: Is it true that $u \in \mathit{shuffle}(u_1, u_2, \ldots, u_k)$ and $v \notin \mathit{shuffle}(v_1, v_2, \ldots, v_k)$?

Let $\mathcal{T}$ and $\mathcal{T}' = \mathit{hide}\ \mathcal{B}\ \mathit{in}\ (\mathcal{S}_c \parallel \mathcal{S}_1 \parallel \mathcal{S}_2 \parallel \cdots \parallel \mathcal{S}_k)$ be the two PCH constructed for $u_1, u_2, \ldots, u_k, u \in \Sigma^*$ similarly as above when we were showing NP-hardness, and let $\mathcal{T}'' = (\mathcal{S}_c'' \parallel \mathcal{S}_1' \parallel \mathcal{S}_2' \parallel \cdots \parallel \mathcal{S}_k')$ and $\mathcal{T}''' = (\mathcal{S}_c''' \parallel \mathcal{S}_1' \parallel \mathcal{S}_2' \parallel \cdots \parallel \mathcal{S}_k')$ be the two PC constructed for $v_1, v_2, \ldots, v_k, v \in \Sigma^*$ similarly as above when we were showing coNP-hardness; the action alphabets used in $\mathcal{T}, \mathcal{T}'$ are now disjoint with those used in $\mathcal{T}'', \mathcal{T}'''$, and the components S_i' (corresponding to v_i) are different than S_i (corresponding to u_i).

We can now take

$$\mathcal{T}_1 = (\mathcal{S}_c^1 \parallel \mathcal{S}_1' \parallel \mathcal{S}_2' \parallel \cdots \parallel \mathcal{S}_k')$$

and

$$\mathcal{T}_2 = \mathit{hide}\ \mathcal{B}\ \mathit{in}\ (\mathcal{S}_c^2 \parallel \mathcal{S}_1 \parallel \mathcal{S}_2 \parallel \cdots \parallel \mathcal{S}_k \parallel \mathcal{S}_1' \parallel \mathcal{S}_2' \parallel \cdots \parallel \mathcal{S}_k')$$

where $\mathcal{S}_c^1$ comprises a copy of $\mathcal{T}$, with its initial state s_0, a copy of S_c'', with its initial state s_0'', and an additional state s^1, which is the initial state of $\mathcal{S}_c^1$, and transitions $s^1 \xrightarrow{f} s_0$, $s^1 \xrightarrow{g} s_0''$ for freshly chosen actions f, g; $\mathcal{S}_c^2$ similarly comprises S_c and S_c''' and the initial state s^2 with the transitions $s^2 \xrightarrow{f} s_0'$, $s^2 \xrightarrow{g} s_0'''$ leading to the initial states of S_c and S_c''', respectively. (In process algebraic terms, $S_c^1 = f.\mathcal{T} + g.S_c''$ and $S_c^2 = f.S_c + g.S_c'''$.)
The set $\mathcal{B}$ of hidden actions is inherited from $\mathcal{T}'$, i.e., it is the union of the action alphabets of $S_1, S_2, \ldots, S_k$.

It can be easily verified that if the SHUFFLE-NONSHUFFLE-answer is YES then (the initial global states in) $\mathcal{T}_1$, $\mathcal{T}_2$ are bisimilar; if the answer is NO then $\mathcal{T}_1$ has a trace which is disabled in $\mathcal{T}_2$. □

Due to the following proposition APC-REL$_{\mathcal{R}}$ is not NP-hard in general unless NP = coNP.

Proposition 14 APC-REL$_{\sqsubseteq_{tr}}$ *is in* coNP.

Proof Given two acyclic PC P_1, P_2 (and their initial states), for showing that $P_1 \not\sqsubseteq_{tr} P_2$ it is sufficient to guess a trace w (of size $|P_1|$ at most) and verify that w is enabled in P_1 but disabled in P_2. To find out if a given

$w = a_1 a_2 \dots a_n$ is enabled in a given PC (with no τ-actions and no hiding), by applying the usual subset construction for nondeterministic finite automata to the components we can successively represent all global states reachable by a_1, $a_1 a_2$, $a_1 a_2 a_3$, ... (as used, e.g., in [18]). □

It also seems unlikely that the lower bound of DP-hardness for APCH-REL$_{\mathcal{R}}$ can be much improved, as the next proposition shows.

Proposition 15 APCH-REL$_{\sqsubseteq_{tr}}$ *is in* Π_2^p *(in the polynomial hierarchy).*

Proof Given two acyclic PCH P_1, P_2 (and their initial states), $P_1 \sqsubseteq_{tr} P_2$ means that for every trace w (of size $|P_1|$ at most) which is enabled in P_1 there is a sequence of global states in P_2 which shows that w is enabled in P_2. □

Nevertheless, at least for 'simulation-like' relations we can derive PSPACE-hardness, even with no hiding, as the next subsection shows.

5.2 PSPACE-hardness of simulation-like relations on acyclic PC

Theorem 16 *Problem* APC-REL$_{\mathcal{R}}$ *is* PSPACE*-hard for any* $\mathcal{R}$ *between bisimulation equivalence and simulation preorder (i.e.,* $\sim \subseteq \mathcal{R} \sqsubseteq_{sim}$*), even when restricted to (acyclic and) centralized PC.*

Proof We use the well-known PSPACE-complete problem QBF (truth of quantified boolean formulas) in the following form:

INSTANCE: $\varphi = \exists x_1 \forall x_2 \cdots \exists x_{n-1} \forall x_n F(x_1, x_2, \dots, x_n)$ (where n is even).
QUESTION: Is φ true?

We assume that F is in CNF, i.e., in the form $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ where each *clause* $C_j = \ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3}$ contains exactly three *literals* (a literal being x_i or $\neg x_i$).

Roughly speaking, our reduction will implement the following game:

1. Player I and Player II alternately assign boolean values to variables $x_1, x_2, \dots, x_n$ (in this order).
2. After the assignment, Player II chooses a clause C_j.
3. If some of literals $\ell_{j,1}, \ell_{j,2}, \ell_{j,3}$ is true under the assignment, Player I wins; if not, Player II wins.

It is obvious that φ is true iff Player I has a winning strategy. The following implementation of the above game is another application of so called 'Defender's Choice technique' which has been used for similar results (see [7] for a recent use).

For $\varphi = \exists x_1 \forall x_2 \cdots \exists x_{n-1} \forall x_n F(x_1, x_2, \ldots, x_n)$, where $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, we will construct an acyclic centralized PC

$$\mathcal{T} = \mathcal{S}_c \parallel \mathcal{S}_1 \parallel \mathcal{S}_2 \parallel \cdots \parallel \mathcal{S}_n$$

and two global states E, E' so that $E \not\sqsubseteq_{sim} E'$ when φ is true and $E \sim E'$ when φ is false.

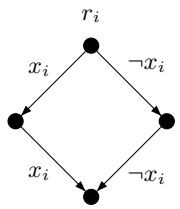

Fig. 7 Component S_i representing variable x_i

The component S_i ($i \in \{1, 2, \ldots, n\}$), corresponding to the boolean variable x_i, is depicted in Figure 7 (the topmost state r_i being initial); its action alphabet is $\Sigma_i = \{x_i, \neg x_i\}$.

(A fragment of) the control component $\mathcal{S}_c$ is depicted in Figure 8; its action alphabet is $\Sigma_c = \Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_n \cup \{c_1, c_2, \ldots, c_m, a\}$. The arrows leading from $s_1, s'_1, \ldots, s_n, s'_n$ (with no labels in the figure) are deemed to have label a.

Let us consider the bisimulation game starting on the pair (E, E') of global states (of $\mathcal{T}$) where $E = (s_0, r_1, r_2, \ldots, r_n)$ (on the left-hand side) and $E' = (s'_0, r_1, r_2, \ldots, r_n)$ (on the right-hand side). In the first round, Player I freely chooses action x_1 or $\neg x_1$; after Player II's response, the control components are in states s_1, s'_1 respectively, and the copies of S_1 (on both sides) either both enable x_1 or both enable $\neg x_1$ – according to the choice having been made by Player I.

In the next round, Player I is forced to play $s_1 \xrightarrow{a} t_2$ – otherwise Player II establishes equality of global states on both sides and wins easily. Now it is Player II who chooses either $s'_1 \xrightarrow{a} t'_{2,T}$ or $s'_1 \xrightarrow{a} t'_{2,F}$ (which corresponds to setting x_2 true or false). To avoid equality, Player I must perform x_2 or $\neg x_2$ according to the choice of Player II; after Player II's response, the control components are in states s_2, s'_2 respectively, and the copies of S_2 either both enable x_2 or both enable $\neg x_2$ – according to the choice having been made by Player II.

In the similar manner, the players alternately choose values for $x_3, x_4, \ldots, x_{n-1}, x_n$; after this phase, the control components are in states s_n, s'_n and the copies of $S_1, S_2, \ldots, S_n$ remember the previous choices for $x_1, x_2, \ldots, x_n$.

Player I is now forced to move $s_n \xrightarrow{a} u$ and Player II freely chooses $s'_n \xrightarrow{a} u'_j$ (which corresponds to choosing clause C_j). Player I has to respect this choice

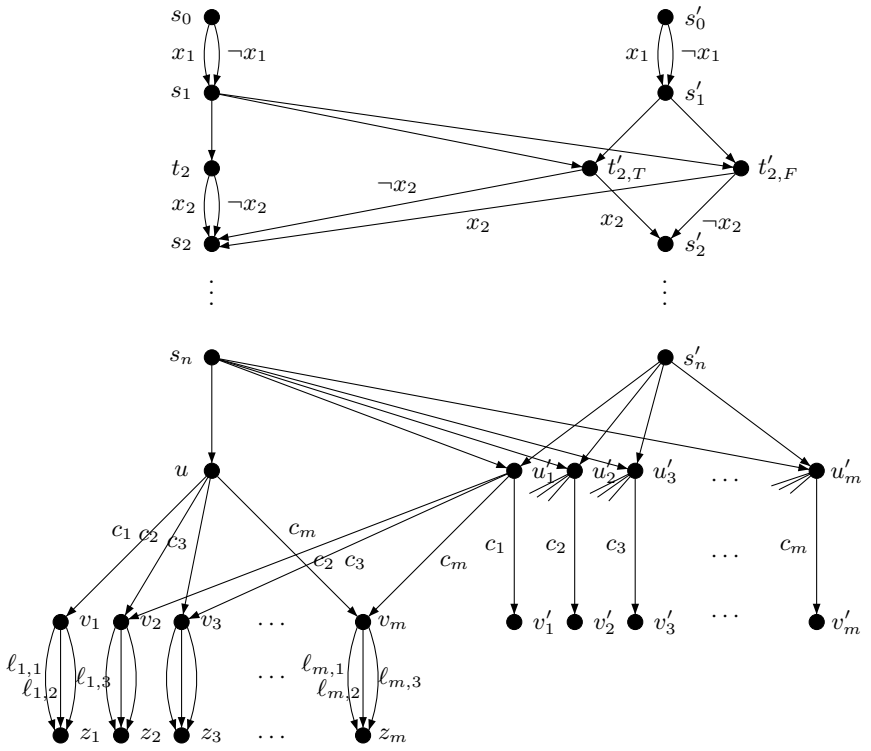

Fig. 8 Control component $\mathcal{S}_c$

by playing $u \xrightarrow{c_j} v_j$ (or $u'_j \xrightarrow{c_j} v'_j$); this is forced by the transitions $u'_j \xrightarrow{c_{j'}} v_{j'}$ for $j \neq j'$ (which are not fully indicated in the figure). The resulting control component states are v_j, v'_j, and Player I can proceed (and win) if and only if there is a literal x_i or $\neg x_i$ among $\ell_{j,1}, \ell_{j,2}, \ell_{j,3}$ which is enabled in the component S_i.

It is thus clear that if φ is true then *Player I* has a winning strategy in the bisimulation game starting from (E, E'); moreover, Player I wins by playing on the left-hand side only, which means that $E \not\sqsubseteq_{sim} E'$. On the other hand, if φ is false then Player II obviously has a winning strategy, which means that $E \sim E'$.

Since $\mathcal{T}$ (with E, E') can be constructed by using only logarithmic workspace (wrt the size of φ), the proof of the theorem is finished. □

The PSPACE-hardness lower bound in Theorem 16 cannot be improved in general, as the following proposition shows.

Proposition 17 APCH-REL$_\sim$ *and* APCH-REL$_{\sqsubseteq_{sim}}$ *are in* PSPACE.

Proof All plays of the (bi)simulation game on APCH P_1, P_2 have length (i.e., number of rounds) at most $|P_1|$. They can be naturally organized in a tree

		$\sim$	$\sqsubseteq_{sim}$	$\sqsubseteq_{tr}$
FS	upper	PTIME (a)	PTIME (a)	PSPACE (b)
				PSPACE-hard (c)
	lower	PTIME-hard (d)		
APC	upper	PSPACE (e)	PSPACE (e)	coNP (f)
		PSPACE-hard (g)		
	lower	coNP-hard (h)		
APCH	upper	PSPACE (e)	PSPACE (e)	Π_2^p (i)
		PSPACE-hard (g)		
	lower	DP-hard (h)		
PC	upper	EXPTIME (j)	EXPTIME (j)	PSPACE (k)
		EXPTIME-hard (l)		
	lower	PSPACE-hard (m)		
PCH	upper	EXPTIME (j)	EXPTIME (j)	EXPSPACE (n)
		EXPTIME-hard (l)		EXPSPACE-hard (o)
	lower	EXPTIME-hard (p)		

Table 1 Overview of complexity results

which can be examined in polynomial space (by using the depth-first search), by which the player who has a winning strategy is determined. □

6 Summary

Table 1 provides a summary of the known results for the equivalence-checking problems considered in this paper. The 'big' rows in the table contain results for different types of systems – FS, APC (acyclic PC), APCH (acyclic PCH), PC (parallel compositions), and PCH (parallel compositions with hiding). For each such type of systems, the known upper and lower complexity bounds are presented. The columns correspond to specific relations: bisimilarity $\sim$, simulation preorder $\sqsubseteq_{sim}$ and trace preorder $\sqsubseteq_{tr}$. The cells that span the columns for $\sim$ and $\sqsubseteq_{sim}$ contain hardness results holding for any $\mathcal{R}$ such that $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{sim}$. The cells that span the columns from $\sim$ to $\sqsubseteq_{tr}$ contain hardness results holding for any $\mathcal{R}$ such that $\sim \subseteq \mathcal{R} \subseteq \sqsubseteq_{tr}$.
All hardness results in the table hold even for centralized systems.

Symbols (a)–(p) in the table refer to the following explanations.

a) Polynomiality easily follows by a greatest fixpoint construction; for more efficient algorithms see e.g. [8,13].
b) It is a special case of language inclusion for nondeterministic finite automata (NFA), which is reducible to language equivalence – a well known PSPACE-complete problem (see, e.g., [11]).

c) This is easily derivable from the PSPACE-hardness of the above problem for NFA.
d) Proved in [16]; Theorem 8 in this paper provides an alternative proof.
e) Proposition 17.
f) Proposition 14.
g) Theorem 16.
h) Theorem 13.
i) Proposition 15.
j) The global transition system (of exponential size) can be constructed explicitly, for which a polynomial time algorithm from (a) can be used.
k) We can use the idea from [18], mentioned in Proposition 14. (It is sufficient to generate and verify a distinguishing trace of at most exponential length).
l) Proved in [9] (a reduction from ALBA-ACCEPT using a variant of the Defender Choice technique).
m) Proved in [14] (by a 'master reduction').
n) The explicitly constructed global transition system is an NFA of exponential size, to which we can apply a polynomial space algorithm (see (b)).
o) Proved in [14] by a reduction from RE^2 (equivalence of regular expressions with squaring).
p) Theorem 12.

References

1. Balcázar, J., Gabarró, J., Sántha, M.: Deciding bisimilarity is P-complete. Formal Aspects of Computing **4**(6A), 638–648 (1992)
2. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM **28**(1), 114–133 (1981)
3. van Glabbeek, R.: Handbook of process algebra. In: J. Bergstra, A. Ponse, S. Smolka (eds.) Handbook of Process Algebra, chap. The Linear Time—Branching Time Spectrum, pp. 3–99. Elsevier (2001)
4. Groote, J.F., Moller, F.: Verification of parallel systems via decomposition. In: Proc. of Third International Conference on Concurrency Theory, *Lecture Notes in Computer Science*, vol. 630, pp. 62–76. Springer Verlag (1992)
5. Hoare, C.A.R.: Communcating Sequential Processes. Prentice-Hall (1985)
6. Immerman, N.: Descriptive Complexity, pp. 53–54. Springer-Verlag (1998)
7. Jančar, P., Srba, J.: Undecidability of bisimilarity by Defender's forcing. J. ACM **55**(1) (2008)
8. Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence. Information and Computation **86**(1), 43–68 (1990)
9. Laroussinie, F., Schnoebelen, P.: The state explosion problem from trace to bisimulation equivalence. In: Proc. 3rd Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'2000), Berlin, Germany, Mar.-Apr. 2000, *Lecture Notes in Computer Science*, vol. 1784, pp. 192–207. Springer (2000)
10. Mansfield, A.: On the computational complexity of a merge recognition problem. DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science **5**, 119–122 (1983)
11. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: 13th Annual Symposium on Switching and Automata Theory, pp. 125–129. IEEE (1972)
12. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)

13. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM Journal on Computing **16**(6), 973–989 (1987)
14. Rabinovich, A.: Complexity of equivalence problems for concurrent systems of finite agents. Information and Computation **139**(2), 111–129 (1997)
15. Sawa, Z.: Equivalence checking of non-flat systems is EXPTIME-hard. In: Proceedings of CONCUR 2003, *Lecture Notes in Computer Science*, vol. 2761, pp. 237–250. Springer (2003)
16. Sawa, Z., Jančar, P.: Behavioural equivalences on finite-state systems are PTIME-hard. Computing and Informatics **24**(5), 513–528 (2005)
17. Shukla, S.K., Hunt, H.B., Rosenkrantz, D.J., Stearns, R.E.: On the complexity of relational problems for finite state processes. In: Proceedings of ICALP'96, *Lecture Notes in Computer Science*, vol. 1099, pp. 466–477. Springer-Verlag (1996)
18. Valmari, A., Kervinen, A.: Alphabet-based synchronisation is exponentially cheaper. In: Proceedings of CONCUR 2002, *Lecture Notes in Computer Science*, vol. 2421, pp. 161–176 (2002)
19. Warmuth, M.K., Haussler, D.: On the complexity of iterated shuffle. Journal of Computer and System Sciences **28**(3), 345–358 (1984)