

# Hardness of preorder checking for basic formalisms

Laura Bozzelli<sup>1</sup>, Axel Legay<sup>1</sup>, Sophie Pinchinat<sup>1</sup>

IRISA, Campus de Beaulieu, 35042 Rennes Cedex, FRANCE.

**Abstract.** We investigate the complexity of preorder checking when the specification is a flat finite-state system whereas the implementation is either a non-flat finite-state system or a standard timed automaton. In both cases, we show that simulation checking is EXPTIME-hard, and for the case of a non-flat implementation, the result holds even if there is no synchronization between the parallel components and their alphabets of actions are pairwise disjoint. Moreover, we show that the considered problems become PSPACE-complete when the specification is assumed to be deterministic. Additionally, we establish that comparing a synchronous non-flat system with no hiding and a flat system is PSPACE-hard for any relation between trace containment and bisimulation equivalence.

## 1 Introduction

One popular approach to formal verification of reactive systems is equivalence/preorder checking between a specification and an implementation which formally describe at a different level of abstraction a given system. This scenario may arise either because the design is being carried out in an incremental fashion, or because the system is too complex and an abstraction needs to be used to verify its properties. In this context, the verification problem is mathematically formulated as a question whether a behavioral equivalence or preorder holds between the labeled state-transition graphs of the specification and the implementation. Decidability and complexity issues for equivalence/preorder checking have been addressed for various computational models of reactive systems (see [10] for a survey of the existing results for infinite-state systems). Moreover, many notions of equivalences or preorders have been investigated, which turn out to be useful for specific aims. Van Glabbeek [21] classified such equivalences/preorders in a hierarchy, where bisimulation equivalence is the finest and trace containment is the coarsest.

**Non-flat finite-state systems.** Finite-state systems are a natural and a primary target in formal verification. When the systems are given as explicit (flat) state-transition graphs, then many relevant verification problems are tractable. For example, simulation-preorder checking and bisimulation-equivalence checking are PTIME-complete [2, 15]. However, in a concurrent setting, the system under consideration is typically the parallel composition of many components (we call such systems *non-flat systems*). As a consequence, the size of the global state-transition graph is usually exponential in the size of the system presentation. This phenomenon is known as ‘state explosion’, and coping with this problem is one of the most important issues in computer-aided verification. From a theoretical point of view, the goal is to understand better which verification problems have to face state explosion in an intrinsic way and which special manner of combining subsystems could avoid state explosion.

Different models of non-flat systems have been investigated. The simplest one is the fully asynchronous model (synchronization-free non-flat systems), where at each step exactly one component performs a transition [20, 13]. For more complex models, the components can communicate by shared actions (or by access to shared variables), and, additionally, actions may be ‘hidden’ (i.e., replaced by some special action) after the parallel composition [7, 12, 14]. For synchronization-free non-flat systems, some problems in equivalence/preorder checking are still tractable. In particular, Groote and Moller [5] have shown that if the alphabets of actions of the components are assumed to be pairwise disjoint, then checking bisimulation equivalence (and other equivalences which satisfy a certain set of axioms) is PTIME-complete. Moreover, for these systems and also for basic synchronous non-flat systems with no hiding (where the components are forced to synchronize on shared actions), checking trace containment/equivalence has the same complexity as for flat systems, i.e. it is PSPACE-complete [17, 20]. Simulation-preorder checking and bisimulation-equivalence checking for synchronous non-flat systems (with or without hiding) are hard, since they are EXPTIME-complete [9, 4, 11, 16]. Checking whether a *synchronous* non-flat system (where the communication is allowed by access to shared variables) is simulated by a *flat system* remains EXPTIME-hard [4]. Moreover, Rabinovich [14] has shown that preorder/equivalence checking between a synchronous non-flat system *with hiding* and a flat system is PSPACE-hard for any relation between trace containment and bisimulation. More recently, Muscholl and Walukiewicz [13] have obtained a surprising result: checking whether a deterministic *flat system* is simulated by a *synchronization-free* non-flat system whose components are deterministic remains EXPTIME-hard. The exact complexity for the converse direction, i.e., whether a synchronization-free non-flat system is simulated by a flat system is open.

**Timed automata.** Timed automata (TA) introduced by Alur and Dill [1] are a widely accepted formalism to model the behavior of real-time systems. Equivalence/preorder checking for this infinite-state computational model has been addressed in many papers. Timed language containment/equivalence is undecidable [1]. Timed bisimulation and timed simulation have been shown to be decidable and in EXPTIME in [19] and [18], respectively; matching lower bounds have been given in [11]. Time-abstract simulation and time-abstract bisimulation have been considered in [6] and are in EXPTIME.

**Our contribution.** We investigate the complexity of preorder checking when the specification is a flat system and the implementation is either a timed automaton or a non-flat system. Note that the considered setting is relevant since the specification is more abstract than the implementation, and, thus, it is usually described by a simple formalism. The results obtained are as follows:

- Checking whether a timed automaton is time-abstract simulated by a flat system is EXPTIME-hard.
- Checking whether a synchronization-free non-flat system is simulated by a flat system is EXPTIME-hard even if the components of the implementation are assumed to be deterministic and with pairwise disjoint alphabets of actions.
- The two problems above are PSPACE-hard if the specification is deterministic.

- Comparing a synchronous non-flat system *with no hiding* and a flat system is PSPACE-hard for any relation lying in between trace containment and bisimilarity.

Our first result in a sense improves the EXPTIME-hardness result for timed simulation between timed automata, by showing that checking timed-abstract simulation remains EXPTIME-hard even if one of the compared TA is replaced by a flat (finite-state) system. Regarding our second and third results, they imply that refinement checking of non-flat systems is intractable even for the simplest model (action-based parallel compositions of deterministic components with pairwise disjoint alphabets) and even if the specification is flat (note that for deterministic specifications, simulation preorder and trace containment are equivalent notions). Finally, our fourth result significantly strengthens the PSPACE-hardness result of Rabinovich [14] in which hiding is involved.

It is interesting to observe that our second result is surprising for the following reasons: *if the alphabets of the components are assumed to be pairwise disjoint*, then bisimulation checking between non-flat systems is in PTIME [5], and simulation checking between a flat implementation and a non-flat specification (i.e., whether a flat system is simulated by a non-flat system) is in PTIME as well [13].<sup>1</sup>

Note that the lower bounds for the first three problems are optimal since they match well-known upper bounds in the literature (see Section 2).

Due to lack of space, for the omitted details we refer the interested reader to a forthcoming extended version of this paper.

## 2 Preliminaries

**Labeled transition systems and simulation preorder.** A *labeled transition system* (LTS) over a (possibly infinite) set of actions  $Act$  is a tuple  $\mathcal{G} = \langle Act, S, s_0, \Delta \rangle$ , where  $S$  is a (possibly infinite) set of states,  $s_0 \in S$  is a designated initial state, and  $\Delta \subseteq S \times Act \times S$  is the transition relation. A transition  $(s, a, s') \in \Delta$  is denoted by  $s \xrightarrow{a} s'$ . We say that  $\mathcal{G}$  is *deterministic* if for all  $s \in S$  and  $a \in Act$ , there is at most one transition of the form  $s \xrightarrow{a} s'$  for some state  $s'$ . The set of traces of  $\mathcal{G}$ ,  $Tr(\mathcal{G})$ , is the set of finite words  $a_1, \dots, a_n$  over  $Act$  such that there is a path in  $\mathcal{G}$  from the initial state of the form  $s_0 \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_n} s_n$ .

An  $Act$ -labeled tree is an unordered finite or infinite tree whose edges are labeled by actions in  $Act$ . Note that an  $Act$ -labeled tree is a particular LTS over  $Act$  whose initial state is the root. For a LTS  $\mathcal{G}$  over  $Act$  and a state  $s$  of  $\mathcal{G}$ , the *unwinding of  $\mathcal{G}$  from  $s$* , written  $Unw(\mathcal{G}, s)$ , is the  $Act$ -labeled tree defined in the usual way.

Given two LTS  $\mathcal{G}_1 = \langle Act_1, S_1, s_1^0, \Delta_1 \rangle$  and  $\mathcal{G}_2 = \langle Act_2, S_2, s_2^0, \Delta_2 \rangle$ , a *simulation from  $\mathcal{G}_1$  to  $\mathcal{G}_2$*  is a relation  $\mathcal{R} \subseteq S_1 \times S_2$  satisfying the following for all  $(s_1, s_2) \in \mathcal{R}$ : if  $s_1 \xrightarrow{a} s'_1 \in \Delta_1$  for some state  $s'_1 \in S_1$  and  $a \in Act_1$ , then there is some state  $s'_2 \in S_2$  so that  $s_2 \xrightarrow{a} s'_2 \in \Delta_2$  and  $(s'_1, s'_2) \in \mathcal{R}$ . If, additionally, the inverse of  $\mathcal{R}$  is a simulation from  $\mathcal{G}_2$  to  $\mathcal{G}_1$ , then we say that  $\mathcal{R}$  is a *bisimulation* from  $\mathcal{G}_1$  to  $\mathcal{G}_2$ . Given states  $s_1 \in S_1$  and  $s_2 \in S_2$ , we say that  $s_1$  is *simulated by  $s_2$*  (resp.,  $s_1$  and  $s_2$  are *bisimilar*) if there is a simulation

<sup>1</sup> in [13], membership in PTIME is shown for the case in which the components of the specification and the implementations are assumed to be deterministic. However, the proof can be easily extended to the case in which this requirement is relaxed.

(resp., bisimulation)  $\mathcal{R}$  from  $\mathcal{G}_1$  to  $\mathcal{G}_2$  such that  $(s_1, s_2) \in \mathcal{R}$ . The *simulation preorder*  $\preceq$  (resp., the *bisimulation equivalence*  $\sim_{bis}$ ) is the binary relation over LTS defined as:  $\mathcal{G}_1 \preceq \mathcal{G}_2$  (resp.,  $\mathcal{G}_1 \sim_{bis} \mathcal{G}_2$ ) iff the initial state  $s_1^0$  of  $\mathcal{G}_1$  is simulated by the initial state  $s_2^0$  of  $\mathcal{G}_2$  (resp.,  $s_1^0$  and  $s_2^0$  are bisimilar). Moreover, the *trace containment preorder*  $\sqsubseteq_{tr}$  is defined as:  $\mathcal{G}_1 \sqsubseteq_{tr} \mathcal{G}_2$  iff  $Tr(\mathcal{G}_1) \subseteq Tr(\mathcal{G}_2)$ . Note that for each  $\preceq \in \{\sim_{bis}, \preceq, \sqsubseteq_{tr}\}$ ,  $\mathcal{G}_1 \preceq \mathcal{G}_2$  iff  $Unw(\mathcal{G}_1, s_1^0) \preceq Unw(\mathcal{G}_2, s_2^0)$ .

**Flat and Non-flat systems.** A *flat system* (FS) is an LTS  $\mathcal{A} = \langle Act, Q, q_0, \Delta \rangle$  such that  $Act$  and the set of states  $Q$  are both finite. The size of  $\mathcal{A}$  is  $|\mathcal{A}| = |Q| + |\Delta|$ .

A *synchronization-free Non-Flat System* (NFS) is a tuple  $\mathcal{S} = \langle \mathcal{A}_1, \dots, \mathcal{A}_k \rangle_{SF}$  such that each component  $\mathcal{A}_i = \langle Act_i, Q_i, q_i^0, \Delta_i \rangle$  is a FS.  $\mathcal{S}$  induces the FS  $\llbracket \mathcal{S} \rrbracket$  given by

$$\llbracket \mathcal{S} \rrbracket = \left\langle \bigcup_{i=1}^{i=k} Act_i, Q_1 \times \dots \times Q_k, (q_1^0, \dots, q_k^0), \Delta_{SF} \right\rangle$$

where  $\Delta_{SF}$  is defined as follows:  $((q_1, \dots, q_k), a, (q'_1, \dots, q'_k)) \in \Delta_{SF}$  iff for some  $i$ ,  $(q_i, a, q'_i) \in \Delta_i$  and for all  $j \neq i$ , we have  $q'_j = q_j$ .

We also consider *synchronous NFS*  $\mathcal{S} = \langle \mathcal{A}_1, \dots, \mathcal{A}_k \rangle$ , where the components  $\mathcal{A}_i = \langle Act_i, Q_i, q_i^0, \Delta_i \rangle$  communicate by synchronization on common actions. Formally,  $\mathcal{S}$  induces the FS given by

$$\llbracket \mathcal{S} \rrbracket = \left\langle \bigcup_{i=1}^{i=k} Act_i, Q_1 \times \dots \times Q_k, (q_1^0, \dots, q_k^0), \Delta \right\rangle$$

where  $\Delta$  is defined as:  $((q_1, \dots, q_k), a, (q'_1, \dots, q'_k)) \in \Delta$  iff for each  $i$ ,  $(q_i, a, q'_i) \in \Delta_i$  if  $a \in Act_i$ , and  $q'_i = q_i$  otherwise. Note that all the components  $\mathcal{A}_i$  with  $a \in Act_i$  must perform a transition labeled by  $a$ . Moreover, note that if distinct components have no actions in common, then  $\llbracket \langle \mathcal{A}_1, \dots, \mathcal{A}_k \rangle \rrbracket = \llbracket \langle \mathcal{A}_1, \dots, \mathcal{A}_k \rangle_{SF} \rrbracket$ . The size of  $\mathcal{S}$  is  $|\mathcal{S}| = \sum_{i=1}^{i=k} |\mathcal{A}_i|$ .

**Timed automata.** Let  $\mathbb{R}_{\geq 0}$  be the set of non-negative reals. Fix a finite set of *clock variables*  $X$ . The set  $C(X)$  of *clock constraints* (over  $X$ ) is the set of boolean combinations of formulas of the form  $x \leq c$  or  $x < c$ , where  $x \in X$ , and  $c$  is a natural number. A *(clock) valuation* (over  $X$ ) is a function  $v : X \rightarrow \mathbb{R}_{\geq 0}$  that maps every clock to a non-negative real number. Whether a valuation  $v$  *satisfies* a clock constraint  $g \in C(X)$ , denoted  $v \models g$ , is defined in a natural way. For  $t \in \mathbb{R}_{\geq 0}$ , the valuation  $v + t$  is defined as  $(v + t)(x) = v(x) + t$  for all  $x \in X$ . For  $Y \subseteq X$ , the valuation  $v[Y := 0]$  is defined as  $(v[Y := 0])(x) = 0$  if  $x \in Y$  and  $(v[Y := 0])(x) = v(x)$  otherwise.

**Definition 1.** [1] A *timed automaton* (TA) over a finite set of actions  $Act$  is a tuple  $\mathcal{T} = \langle Act, X, Q, q_0, \rho \rangle$ , where  $Q$  is a finite set of locations,  $q_0 \in Q$  is the initial location, and  $\rho \subseteq Q \times Act \times C(X) \times 2^X \times Q$  is a finite transition relation.

The TA  $\mathcal{T}$  induces an infinite-state LTS  $\llbracket \mathcal{T} \rrbracket = \langle \mathbb{R}_{\geq 0} \times Act, S, s_0, \Delta \rangle$  over  $\mathbb{R}_{\geq 0} \times Act$ , where  $S$  is the set of pairs  $(q, v)$  such that  $q \in Q$  and  $v$  is a clock valuation,  $s_0 = (q_0, \vec{0})$  ( $\vec{0}$  assigns to each clock value 0), and  $\Delta$  is defined as follows:  $(q, v) \xrightarrow{(t, a)} (q', v') \in \Delta$  iff there is a transition  $(q, a, g, Y, q') \in \rho$  such that  $v + t \models g$  and  $v' = (v + t)[Y := 0]$ .

The *abstract* LTS associated with  $\mathcal{T}$  is  $\llbracket \mathcal{T} \rrbracket_{abs} = \langle Act, S, s_0, \Delta_{abs} \rangle$ , where  $(q, v) \xrightarrow{a} (q', v') \in \Delta_{abs}$  iff  $(q, v) \xrightarrow{(t, a)} (q', v') \in \Delta$  for some  $t \geq 0$ . We say that  $\mathcal{T}$  is *strongly timed-deterministic* if  $\llbracket \mathcal{T} \rrbracket_{abs}$  is deterministic and for each  $(q, v) \xrightarrow{a} (q', v') \in \Delta_{abs}$ , there is exactly one timestamp  $t$  such that  $(q, v) \xrightarrow{(t, a)} (q', v') \in \Delta$ .

**Investigated problems.** We consider the following decision problems:

**Problem 1:** given a TA  $\mathcal{T}$  and a FS  $\mathcal{B}$ , does  $\llbracket \mathcal{T} \rrbracket_{abs} \preceq \mathcal{B}$  hold?

**Problem 2:** given a *synchronization-free* NFS  $\mathcal{S}$  and a FS  $\mathcal{B}$ , does  $\llbracket \mathcal{S} \rrbracket \preceq \mathcal{B}$  hold?

**Problem 3 ( $\trianglelefteq$ ):** given a *synchronous* NFS  $\mathcal{S}$  and a FS  $\mathcal{B}$ , does  $\llbracket \mathcal{S} \rrbracket \trianglelefteq \mathcal{B}$  hold?

where  $\trianglelefteq$  is a fixed binary relation on LTS. We also consider the *deterministic versions* of Problems 1 and 2, where the FS  $\mathcal{B}$  above is assumed to be deterministic.

**Theorem 1.** *Problems 1 and 2 are in EXPTIME, while their deterministic versions are in PSPACE.*

*Proof.* Membership in EXPTIME for Problems 1 and 2 directly follows from the following:

**Fact 1 [2]:** given two FS  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , checking whether  $\mathcal{A}_1 \preceq \mathcal{A}_2$  is in PTIME.

**Fact 2:** for a NFS  $\mathcal{S}$ , the size of the FS  $\llbracket \mathcal{S} \rrbracket$  is singly exponential in the size of  $\mathcal{S}$ .

**Fact 3 [6]:** given two TA  $\mathcal{T}_1$  and  $\mathcal{T}_2$ , checking whether  $\llbracket \mathcal{T}_1 \rrbracket_{abs} \preceq \llbracket \mathcal{T}_2 \rrbracket_{abs}$  is in EXPTIME.

Membership in PSPACE for the deterministic versions of Problems 1 and 2 directly follows from Fact 2 and the following:

**Fact 4:** for two LTS  $\mathcal{G}_1$  and  $\mathcal{G}_2$  such that  $\mathcal{G}_2$  is deterministic,  $\mathcal{G}_1 \preceq \mathcal{G}_2$  iff  $\mathcal{G}_1 \sqsubseteq_{tr} \mathcal{G}_2$ .

**Fact 5 [1]:** given a TA  $\mathcal{T}$ , one can construct a FS  $\mathcal{A}_{\mathcal{T}}$  (*region automaton*) of size singly exponential in the size of  $\mathcal{T}$  such that  $Tr(\mathcal{A}_{\mathcal{T}}) = Tr(\llbracket \mathcal{T} \rrbracket_{abs})$ .

**Fact 6:** given a deterministic FS  $\mathcal{A}$  over  $Act$ , one can trivially construct in linear-time a standard finite-state automaton which accepts all and only the words in  $Act^* \setminus Tr(\mathcal{A})$ .

**Fact 7 [8]:** checking emptiness of the intersection of the languages accepted by two (nondeterministic) finite-state automata is in NLOGSPACE.

In the rest of this paper, we provide lower bounds for Problems 1 and 2 (and their deterministic versions) which match the upper bounds of Theorem 1. Moreover, we show that Problem 3 ( $\trianglelefteq$ ) is PSPACE-hard for any binary relation  $\trianglelefteq$  lying in between trace containment and bisimulation equivalence.

### 3 EXPTIME-hardness of Problems 1 and 2

In this section, we show that Problems 1 and 2 are both EXPTIME-hard by polynomial-time reductions from the acceptance problem for *linearly-bounded alternating* Turing Machines (TM) with a binary branching degree, which is EXPTIME-complete [3].

In the rest of this section, we fix such a TM machine  $\mathcal{M} = \langle A, Q = Q_{\forall} \cup Q_{\exists} \cup \{q_{acc}, q_{rej}\}, q_0, \delta, \{q_{acc}\} \rangle$ , where  $A$  is the input alphabet,  $Q_{\exists}$  (resp.,  $Q_{\forall}$ ) is the set of existential (resp., universal) states,  $q_0$  is the initial state,  $q_{acc} \notin Q_{\forall} \cup Q_{\exists}$  is the (terminal) accepting state,  $q_{rej} \notin Q_{\forall} \cup Q_{\exists}$  is the (terminal) rejecting state, and  $\delta : (Q_{\forall} \cup Q_{\exists}) \times$

$A \rightarrow (Q \times A \times \{+1, -1\}) \times (Q \times A \times \{+1, -1\})$  is the transition function. In each non-terminal step (i.e., the current state is in  $Q_{\forall} \cup Q_{\exists}$ ),  $\mathcal{M}$  overwrites the tape cell being scanned, and the tape head moves one position to the left ( $-1$ ) or right ( $+1$ ). Moreover, we fix an input  $\alpha \in A^*$  and consider the parameter  $n = |\alpha|$ .

Since  $\mathcal{M}$  is linearly bounded, w.l.o.g. we assume that  $\mathcal{M}$  uses exactly  $n$  tape cells when started on the input  $\alpha$ . Hence, a TM configuration (of  $\mathcal{M}$  over  $\alpha$ ) is a word  $C = \beta_1, (a, q), \beta_2 \in A^* \cdot (A \times Q) \cdot A^*$  of length exactly  $n$  denoting that the tape content is  $\beta_1, a, \beta_2$ , the current state is  $q$ , and the tape head is at position  $|\beta_1| + 1$ . The initial configuration  $C_\alpha$  is given by  $(\alpha(1), q_0), \alpha(2), \dots, \alpha(n)$ . Moreover, w.l.o.g. we assume that when started on  $C_\alpha$ , no matter what are the universal and existential choices,  $\mathcal{M}$  always *halts* by reaching a terminal configuration  $C$ , i.e. such that the associated state, denoted by  $q(C)$ , is in  $\{q_{acc}, q_{rej}\}$  (this assumption is standard, see [3]).

It is convenient to define the notion of acceptance of  $\mathcal{M}$  as follows. For each  $q \in Q$ , define  $Val(q) = 1$  if  $q = q_{acc}$ , and  $Val(q) = 0$  otherwise. A (full) *pseudo-computation tree*  $T$  of  $\mathcal{M}$  from a TM configuration  $C$  is a binary tree whose nodes are labeled by TM configurations and such that the root is labeled by  $C$ , the internal nodes have two children, and the leaves are labeled by terminal configurations. If  $T$  is finite, then its *boolean value*  $Val(T) \in \{0, 1\}$  is defined as follows. If  $T$  consists just of the root, then  $Val(T) = Val(q(C))$ . Otherwise, let  $T_L$  and  $T_R$  be the trees rooted at the children of the root of  $T$ . Then,  $Val(T)$  is  $Val(T_L) \vee Val(T_R)$  if  $q(C) \in Q_{\exists}$ , and  $Val(T_L) \wedge Val(T_R)$  otherwise. The tree  $T$  *leads to acceptance* if  $Val(T) = 1$ . A (full) *computation tree* is a pseudo-computation tree which is faithful to the evolution of  $\mathcal{M}$ .  $\mathcal{M}$  accepts  $\alpha$  iff the computation tree of  $\mathcal{M}$  over  $C_\alpha$ , which by our assumption is finite, leads to acceptance.

In the rest of this section, we show that it is possible to construct in *polynomial time* (in the sizes of the fixed TM  $\mathcal{M}$  and input  $\alpha$ ) an instance  $I$  of Problem 1 (resp., Problem 2) such that  $\mathcal{M}$  accepts  $\alpha$  iff the instance  $I$  has a positive answer.

**Preliminary step: encoding of acceptance by simulation.** Before illustrating the polynomial reductions to Problems 1 and 2 (in Subsections 3.1 and 3.2, respectively), we consider a preliminary step in which we define for a given finite set of actions  $Act$  and a given encoding of the TM configurations by words over  $Act$ , two  $Act$ -labeled trees  $ET_{C_\alpha}$  and  $VT_{q(C_\alpha),1}$  such that  $\mathcal{M}$  accepts the fixed input  $\alpha$  iff the root of  $ET_{C_\alpha}$  is simulated by the root of  $VT_{q(C_\alpha),1}$ . The tree  $ET_{C_\alpha}$  is finite and deterministic, and it is a natural encoding of the computation tree of  $\mathcal{M}$  over  $C_\alpha$ , while the tree  $VT_{q(C_\alpha),1}$  is *infinite* and *non-deterministic*, and encodes in a suitable way all the possible *finite pseudo-computation trees* of  $\mathcal{M}$  which *lead to acceptance*. The two encodings ensure that the root of  $ET_{C_\alpha}$  is *simulated* by the root of  $VT_{q(C_\alpha),1}$  iff  $ET_{C_\alpha}$  is ‘contained’ in  $VT_{q(C_\alpha),1}$ , i.e. the full computation tree of  $\mathcal{M}$  over  $\alpha$  leads to acceptance. Now, we define these trees.

*Assumptions:* we assume that  $Act \supseteq Q \cup (Q \times \{0, 1\}) \cup \{L, R\}$  and each TM configuration  $C$  is encoded by a finite word over  $Act$ , denoted by  $code(C)$ . We denote by  $Codes$  the finite set of these codes, which are assumed to have the same length. The precise definition of  $Act$  and  $Codes$  will depend on the specific problem we consider (either Problem 1 or Problem 2).

For each  $code \in Codes$ , let  $T_{code}$  be the finite  $Act$ -labeled tree, which is a chain and whose unique maximal path from the root is labeled by  $code$ . For a *non-terminal*

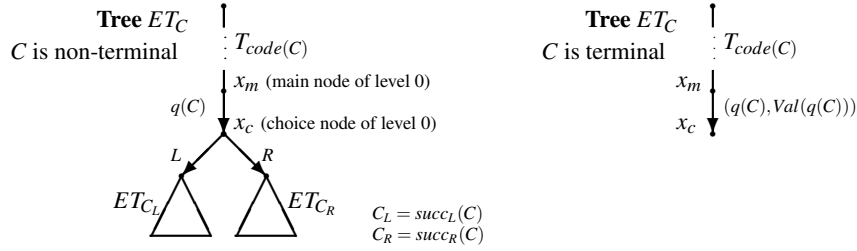
configuration  $C = \beta_1, (a, q), \beta_2$  (i.e., such that  $q \in Q_{\forall} \cup Q_{\exists}$ ),  $\text{succ}_L(C)$  (resp.,  $\text{succ}_R(C)$ ) denotes the TM successor of  $C$  obtained by choosing the left (resp., right) triple in  $\delta(q, a)$ . A *good* configuration is a configuration reachable from  $C_\alpha$ .

**Definition 2.** [Emulation trees] For each good TM configuration  $C$ , the finite deterministic Act-labeled emulation tree of  $\mathcal{M}$  from  $C$ , denoted by  $ET_C$ , is inductively defined as follows. The tree  $ET_C$  is obtained from  $T_{\text{code}(C)}$  by adding an edge from the leaf  $x_m$  of  $T_{\text{code}(C)}$  (main node of level 0) to a new node  $x_c$  (choice node of level 0) such that:

- $C$  is terminal:  $x_c$  is a leaf and the edge from  $x_m$  to  $x_c$  is labeled by  $(q(C), \text{Val}(q(C)))$ .
- $C$  is not terminal: let  $C_L = \text{succ}_L(C)$  and  $C_R = \text{succ}_R(C)$ . Then,  $x_c$  has two children  $x_L$  and  $x_R$  so that the subtree rooted at  $x_L$  (resp.,  $x_R$ ) is isomorphic to  $ET_{C_L}$  (resp.,  $ET_{C_R}$ ). The edge from the main node  $x_m$  to the choice node  $x_c$  is labeled by  $q(C)$ , and the edge from  $x_c$  to  $x_L$  (resp., to  $x_R$ ) is labeled by  $L$  (resp.,  $R$ ).

The structure of the emulation trees  $ET_C$  is depicted in Figure 1. The boolean value  $\text{Val}(ET_C)$  of  $ET_C$  is the boolean value of the computation tree of  $\mathcal{M}$  from  $C$ .

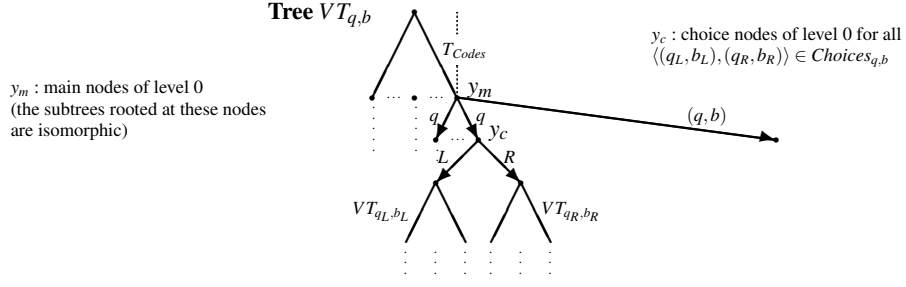
*Remark 1.*  $\mathcal{M}$  accepts  $\alpha$  iff  $\text{Val}(ET_{C_\alpha}) = 1$ .



**Fig. 1.** Structure of the Act-labeled emulation tree  $ET_C$  for a good TM configuration  $C$ .

Let  $T_{\text{Codes}}$  be the tree encoding of *Codes*, i.e. the unique *deterministic* finite Act-labeled tree such that the set of sequences of edge-labels associated with all maximal paths from the root is exactly *Codes*. For all  $q \in Q$  and boolean value  $b \in \{0, 1\}$ , let  $\text{Choices}_{q,b}$  be the non-empty finite set of pairs  $\langle (q_1, b_1), (q_2, b_2) \rangle$  such that  $q_1, q_2 \in Q$ ,  $b_1, b_2 \in \{0, 1\}$ , and  $b_1 \vee b_2 = b$  if  $q$  is an existential state, and  $b_1 \wedge b_2 = b$  otherwise.

**Definition 3.** [Valuation trees] For all  $q \in Q$  and  $b \in \{0, 1\}$ , the Act-labeled  $(q, b)$ -valuation tree of  $\mathcal{M}$ , denoted by  $VT_{q,b}$ , is the Act-labeled infinite tree satisfying the following.  $VT_{q,b}$  is obtained from  $T_{\text{Codes}}$  by adding for each leaf  $y_m$  (main node of level 0) of  $T_{\text{Codes}}$  exactly  $|\text{Choices}_{q,b}|$  edges from  $y_m$  labeled by  $q$  and one edge from  $y_m$  to a leaf node labeled by  $(q, b)$ . Moreover, for each  $\langle (q_L, b_L), (q_R, b_R) \rangle \in \text{Choices}_{q,b}$ , one of these new edges labeled by  $q$  leads to a node  $y_c$  (choice node of level 0) so that: (1)  $y_c$  has two children  $y_L$  and  $y_R$ , (2) the edge from  $y_c$  to  $y_L$  (resp., to  $y_R$ ) is labeled by  $L$  (resp.,  $R$ ), and (3) the subtree rooted at  $y_L$  (resp.,  $y_R$ ) is isomorphic to  $VT_{q_L, b_L}$  (resp.,  $VT_{q_R, b_R}$ ). Note that the subtrees rooted at the main nodes of level 0 are isomorphic. The structure of  $VT_{q,b}$  is depicted in Figure 2.



**Fig. 2.** Structure of the infinite *Act*-labeled valuation tree  $VT_{q,b}$ .

**Lemma 1.** Fix a good TM configuration  $C$ , a state  $q \in Q$ , and  $b \in \{0, 1\}$ . Then, the root of  $ET_C$  is simulated by the root of  $VT_{q,b}$  iff  $q = q(C)$  and  $b = \text{Val}(ET_C)$ .

Lemma 1 can be easily proved by structural induction over the (finite) tree  $ET_C$ . By Remark 1 and Lemma 1, we obtain the following result.

**Lemma 2.**  $\mathcal{M}$  accepts  $\alpha$  iff the root of  $ET_{C_\alpha}$  is simulated by the root of  $VT_{q(C_\alpha), 1}$ .

### 3.1 EXPTIME-hardness of Problem 1

In the case of Problem 1, the set *Act* of actions is given by

$$\text{Act} = Q \cup (Q \times \{0, 1\}) \cup \{L, R\} \cup A \cup (A \times Q) \cup \{\lambda_0, \lambda_1\}$$

The code of each TM configuration  $C = C(1), \dots, C(n)$  is the word over *Act* of length  $3n$  defined as:

$$\text{code}(C) = \lambda_0, C(1), \lambda_1, \dots, \lambda_0, C(n), \lambda_1$$

Note that each symbol  $u \in A \cup (A \times Q)$  is encoded by the word  $\lambda_0, u, \lambda_1$ . Let  $K$  be the size of  $A \cup (A \times Q)$ . We fix an ordering  $\{a_1, \dots, a_K\}$  of the elements in  $A \cup (A \times Q)$  and we associate to each  $a_i$  the timestamp  $\tau(a_i)$  given by  $i$ .

We construct a strongly timed-deterministic TA  $\mathcal{T}_{em}$  over *Act* and a nondeterministic FS  $\mathcal{A}_{val}$  over *Act* of sizes polynomial in the sizes of the fixed TM  $\mathcal{M}$  and input  $\alpha$  so that: the unwinding of  $\llbracket \mathcal{T}_{em} \rrbracket_{abs}$  from its initial state is the emulation tree  $ET_{C_\alpha}$  of Definition 2, and the unwinding of  $\mathcal{A}_{val}$  from its initial state is the valuation tree  $VT_{q(C_\alpha), 1}$  of Definition 3. By Lemma 2 it follows that  $\mathcal{M}$  accepts  $\alpha$  iff  $\llbracket \mathcal{T}_{em} \rrbracket_{abs} \preceq \mathcal{A}_{val}$ . Hence, EXPTIME-hardness of Problem 1 follows.

**Theorem 2.** Given a TA  $\mathcal{T}$  and a nondeterministic FS  $\mathcal{A}$ , checking whether  $\llbracket \mathcal{T} \rrbracket_{abs} \preceq \mathcal{A}$  is EXPTIME-hard, even if the TA  $\mathcal{T}$  is assumed to be strongly timed-deterministic.

In the rest of this subsection, we illustrate the construction of  $\mathcal{T}_{em}$  (the construction of  $\mathcal{A}_{val}$  is an easy task). Note that for a TM configuration  $C = C(1), \dots, C(n)$ , the ‘value’  $u_i$  of the  $i$ -th symbol of the left (resp., right) successor of  $C$  is completely determined by the values  $C(i-1)$ ,  $C(i)$  and  $C(i+1)$  (taking  $C(i+1)$  for  $i = n$  and  $C(i-1)$  for  $i = 1$  to be some special symbol, say  $\perp$ ). We denote by  $\text{next}_L(C(i-1), C(i), C(i+1))$  (resp.,

$next_R(C(i-1), C(i), C(i+1))$  our expectation for  $u_i$  (these functions can be trivially obtained from the transition function  $\delta$  of the fixed TM  $\mathcal{M}$ ).

$\mathcal{T}_{em}$  uses  $n+1$  clocks  $x_0, x_1, \dots, x_n$  in order to ensure a correct emulation of the evolution of  $\mathcal{M}$ . Clock  $x_0$  is reset on generating the special action  $\lambda_1$  (and only in this circumstance). On generating the special action  $\lambda_0$ , we require  $(x_0 = 0)$  to hold, on generating  $u \in A \cup (A \times Q)$ , we require  $(x_0 = \tau(u))$  to hold, and on generating  $\lambda_1$ , we require  $(x_0 = K)$  to hold. This ensures that the durations of the consecutive three steps at which the code  $\lambda_0, u, \lambda_1$  of  $u \in A \cup (A \times Q)$  is generated are  $0, \tau(u)$ , and  $K - \tau(u)$ , respectively. Hence, the overall duration of these steps is exactly  $K$  (independent on the specific action  $u \in A \cup (A \times Q)$ ). Moreover, the overall duration of the sequence of steps at which a code  $code(C)$  is generated is exactly  $nK$ . Furthermore, whenever an action  $u \in A \cup (A \times Q)$  is generated from a location associated with the  $i$ -th symbol of a TM configuration  $C$ , clock  $x_i$  is reset (and only in this circumstance). This ensures that when the special action  $\lambda_0$ , associated with the  $i$ -th symbol of the next generated TM configuration  $C_{dir}$  with  $dir \in \{L, R\}$ , has to be generated, the following holds: the value of clock  $x_i$  is exactly  $nK - \tau(C(i))$  and (assuming  $i < n$ ) the value of clock  $x_{i+1}$  is exactly  $(n-1)K - \tau(C(i+1))$ . Thus,  $\mathcal{T}_{em}$  will time-deterministically move (by taking a transition whose action is  $\lambda_0$  and whose clock constraint is  $x_0 = 0 \wedge x_i = nK - \tau(C(i)) \wedge x_{i+1} = (n-1)K - \tau(C(i+1))$ ) to a location of the form  $p = (i, C(i-1), C(i), C(i+1), dir, \dots)$ , where the ‘value’  $C(i-1)$  is ‘transmitted’ from the previous location. At this point,  $\mathcal{T}_{em}$  has all the information  $(C(i-1), C(i), C(i+1))$ , and  $dir \in \{L, R\}$  to determine the  $i$ -th symbol of  $C_{dir}$ . Thus, there is exactly one transition from location  $p$  of the form  $(p, u, x_0 = \tau(u), \{x_i\}, p')$ , where  $u = next_{dir}(C(i-1), C(i), C(i+1))$ .

### 3.2 EXPTIME-hardness of Problem 2

In the case of Problem 2, the set  $Act$  of actions is given by

$$Act = Q \cup (Q \times \{0, 1\}) \cup \{L, R\} \cup (\{1, \tilde{1}, \dots, n, \tilde{n}\} \times (A \cup (A \times Q)))$$

where for each  $i \in \{1, \dots, n\}$ ,  $\tilde{i}$  denotes a fresh copy of  $i$ . The meaning of these symbols will be explained later. The code of a TM configuration  $C = C(1), \dots, C(n)$  is now a word of length  $2n$  given by

$$code(C) = (\tilde{1}, C(1)), (1, C(1)), \dots, (\tilde{n}, C(n)), (n, C(n))$$

We construct a nondeterministic FS  $\mathcal{B}_{val}$  over  $Act$  and a *synchronization-free* NFS  $\mathcal{S}_{SF}$  over  $Act$  (whose components are deterministic and with pairwise disjoint alphabets of actions) of sizes polynomial in the size of the fixed TM  $\mathcal{M}$  and input  $\alpha$  such that  $\mathcal{M}$  accepts  $\alpha$  iff  $\llbracket \mathcal{S}_{SF} \rrbracket \preceq \mathcal{B}_{val}$ . Hence, EXPTIME-hardness of Problem 2 follows.

We note that a synchronization-free NFS of size polynomial in the size of  $\mathcal{M}$  and  $\alpha$  cannot faithfully emulate the evolution of  $\mathcal{M}$  over the input  $\alpha$ . In order to cope with this problem, we first define suitable extensions  $Ext\_ET_{C_\alpha}$  and  $Ext\_VT_{q(C_\alpha), 1}$  of the emulation tree  $ET_{C_\alpha}$  (Definition 2) and valuation tree  $VT_{q(C_\alpha), 1}$  (Definition 3), respectively, in such a way that the result of Lemma 2 holds even for these extensions. Then, we show that we can construct  $\mathcal{B}_{val}$  and  $\mathcal{S}_{SF}$  in such a way to ensure that the unwinding of  $\llbracket \mathcal{S}_{SF} \rrbracket$  from its initial state is the extended emulation tree  $Ext\_ET_{C_\alpha}$ , and the unwinding of  $\mathcal{B}_{val}$  from its initial state is the extended valuation tree  $Ext\_VT_{q(C_\alpha), 1}$ .

**Extended emulation trees and Extended valuation trees.**

**Definition 4.** [Extended emulation trees] For each good TM configuration  $C$ , an extended emulation tree of  $\mathcal{M}$  from  $C$  is a (possibly infinite) Act-labeled tree  $Ext\_ET_C$  which extends the emulation tree  $ET_C$  (Definition 2) in such a way that the following conditions are inductively satisfied. There is exactly one partial path from the root which is labeled by some code in  $Codes$  (note that by Definition 2, this path coincides with the partial path from the root to the main node of level 0 of  $ET_C$ ). Moreover, each new edge from the main node of level 0 is labeled by an action in  $Act \setminus (Q \cup (Q \times \{0, 1\}))$  and:

- $C$  is terminal: each new edge from the unique leaf of  $ET_C$  is labeled by an action in  $Act \setminus \{L, R\}$ .
- $C$  is not terminal: each new edge from the choice node  $x_c$  of level 0 of  $ET_C$  is labeled by an action in  $Act \setminus \{L, R\}$ . Moreover, let  $x_L$  and  $x_R$  be the children of  $x_c$  in  $ET_C$  (recall that the subtrees rooted at  $x_L$  and  $x_R$  in  $ET_C$  correspond to  $ET_{succ_L(C)}$  and  $ET_{succ_R(C)}$ , respectively). Then, we require that the subtrees rooted at  $x_L$  and  $x_R$  in  $Ext\_ET_C$  are extended emulation trees of  $\mathcal{M}$  from  $succ_L(C)$  and  $succ_R(C)$ , respectively.

Note that for a given (good) TM configuration  $C$ , there can be many extended emulation trees of  $\mathcal{M}$  from  $C$ , and the definition above specifies only some properties that must be satisfied by them. We denote by  $Ext(ET_C)$  the nonempty set of extended emulation trees of  $\mathcal{M}$  from  $C$ . An *extended code* is a word over  $Act$  of the form  $code \cdot u \cdot dir$ , where  $code \in Codes$ ,  $u \in Q \cup (Q \times \{0, 1\})$ , and  $dir \in \{L, R\}$ . Let us consider the infinite Act-labeled valuation trees  $VT_{q,b}$  of Definition 3. A node  $y$  of  $VT_{q,b}$  is called *starting node* iff either  $y$  is the root or  $y$  is the child of some choice node (note that the subtree rooted at a child of a choice node corresponds to some valuation tree  $VT_{q',b'}$ ). We extend  $VT_{q,b}$  as follows, where  $T_{full}$  denotes the Act-labeled infinite tree obtained as the unwinding of the deterministic FS having a unique state and for each  $u \in Act$ , a self-loop labeled by  $u$ .

**Definition 5.** [Extended valuation trees] For all  $q \in Q$  and  $b \in \{0, 1\}$ , the infinite Act-labeled extended valuation tree  $Ext\_VT_{q,b}$  is obtained from  $VT_{q,b}$  as follows. For each node  $y$  of  $VT_{q,b}$ , let  $y_0$  be the first ancestor of  $y$  which is a starting node (note that  $y_0$  may be  $y$ ), and let  $w_{y_0,y}$  be the word over  $Act$  labeling the partial path from  $y_0$  to  $y$ . Note that  $w_{y_0,y}$  has length at most  $2n + 1$  and is the proper prefix of some extended code. Then, for each  $u \in Act$  such that the word  $w_{y_0,y} \cdot u$  is not the prefix of any extended code, we add an edge labeled by  $u$  from  $y$  to the root of a tree isomorphic to  $T_{full}$ .

The following lemma is the variant of Lemma 1 for extended emulation trees and extended valuation trees.

**Lemma 3.** Fix a good TM configuration  $C$ , a state  $q \in Q$ , and  $b \in \{0, 1\}$ . Then, for each extended emulation tree  $Ext\_ET_C \in Ext(ET_C)$ , the root of  $Ext\_ET_C$  is simulated by the root of the extended valuation tree  $Ext\_VT_{q,b}$  iff  $q = q(C)$  and  $b = Val(ET_C)$ .

**Constructions of the FS  $\mathcal{B}_{val}$  and the synchronization-free NFS  $\mathcal{S}_{SF}$ .**

**Lemma 4.** *One can construct a nondeterministic FS  $\mathcal{B}_{val}$  of size polynomial in the sizes of  $\mathcal{M}$  and  $\alpha$  such that the unwinding of  $\mathcal{B}_{val}$  from its initial state is  $Ext\_VT_{q(C_\alpha),1}$ .*

**Lemma 5.** *One can construct a synchronization-free NFS  $\mathcal{S}_{SF}$  (whose components are deterministic and with pairwise disjoint alphabets of actions) of size polynomial in the sizes of  $\mathcal{M}$  and  $\alpha$  such that the unwinding of  $\llbracket \mathcal{S}_{SF} \rrbracket$  from its initial state is in  $Ext(ET_{C_\alpha})$ .*

Below, we prove Lemmata 4 and 5. By Remark 1 and Lemmata 3, 4, and 5, it follows that  $\mathcal{M}$  accepts  $\alpha$  iff  $\llbracket \mathcal{S}_{SF} \rrbracket \preceq \mathcal{B}_{val}$ , where  $\mathcal{B}_{val}$  is the FS of Lemma 4 and  $\mathcal{S}_{SF}$  is the NFS of Lemma 5. Hence, we obtain the desired result.

**Theorem 3.** *Given a FS  $\mathcal{A}$  and a synchronization-free non-flat system  $\mathcal{S}$ , checking whether  $\llbracket \mathcal{S} \rrbracket \preceq \mathcal{A}$  is EXPTIME-hard, even if the components of  $\mathcal{S}$  are assumed to be deterministic and their alphabets are assumed to be pairwise disjoint.*

**Proof of Lemma 4.** We need some additional definition. Let  $0 \leq i < 2n$ ,  $last_\perp \in \{\perp\} \cup (\{1, \tilde{1}, \dots, n, \tilde{n}\} \times (A \cup (A \times Q)))$  ( $\perp$  is for undefined), and  $f \in \{yes, no\}$  such that  $last_\perp = \perp$  iff  $i = 0$ . A  $(i, last_\perp, f)$ -word is a proper prefix  $w_p$  of a code in *Codes* such that  $|w_p| = i$ ,  $last_\perp$  is the last symbol of  $w_p$  if  $i > 0$ , and  $f = yes$  iff  $w_p$  contains some occurrence of a symbol in  $\{\tilde{1}, \dots, \tilde{n}\} \times (A \times Q)$ . For each  $u \in Act$ , we consider the predicate  $Prefix(i, last_\perp, f, u)$  which holds iff there exists a  $(i, last_\perp, f)$ -word  $w_p$  such that  $w_p \cdot u$  is the prefix of some code in *Codes*. Note that by definition of *Codes*, the satisfaction of  $Prefix(i, last_\perp, f, u)$  is independent of what representative is chosen in the set of  $(i, last_\perp, f)$ -words, i.e., for all  $(i, last_\perp, f)$ -words  $w_p$  and  $w'_p$ , it holds that  $w_p \cdot u$  is the prefix of some code in *Codes* iff  $w'_p \cdot u$  is the prefix of some code in *Codes*.

The FS  $\mathcal{B}_{val} = \langle Act, P_{val}, p_{val}^0, \Delta_{val} \rangle$  satisfying the statement of Lemma 4 is defined as follows. The set of states is  $P_{val} = \{p_{full}\} \cup P_{cod} \cup P_{main} \cup P_{choice} \cup \{p_\#\}$ , where:

- From state  $p_{full}$  there are self-loops on all actions from *Act*. Thus, the unwinding of  $\mathcal{B}_{val}$  from  $p_{full}$  corresponds to  $T_{full}$  (see Definition 5).
- $P_{cod}$  consists of states of the form  $p_{cod} = (q, b, i, last_\perp, f)$ , where  $(q, b) \in Q \times \{0, 1\}$ ,  $1 \leq i \leq 2n$ ,  $last_\perp \in \{\perp\} \cup (\{1, \tilde{1}, \dots, n, \tilde{n}\} \times (A \cup (A \times Q)))$ , and  $f \in \{yes, no\}$ , where  $last_\perp = \perp$  iff  $i = 1$ . The states in  $P_{cod}$  are used to generate all the codes in *Codes*. Intuitively,  $(q, b)$  is the currently processed pair TM state/ boolean value,  $i$  is the currently processed position of a *code*. Moreover,  $last_\perp$  keeps track of the last generated symbol if  $i > 1$ , and the flag  $f$  is used to keep track whether a symbol of the form  $(\tilde{j}, u)$  with  $u \in A \times Q$  has already been generated in the previous  $i - 1$  steps. From state  $p_{cod}$ ,  $\mathcal{B}_{val}$  generates all the actions  $u \in Act$  in such a way that the following holds. If  $Prefix(i - 1, last_\perp, f, u)$  holds (this means that the word  $w_p \cdot u$  is the prefix of some code in *Codes*, where  $w_p$  is the  $(i, last_\perp, f)$ -word generated in the previous  $i - 1$  steps), then  $\mathcal{C}_{val}$  generates the letter  $u$  and moves to a state in  $P_{cod}$  of the form  $(q, b, i + 1, u, f')$  if  $i < n$  and to the main state  $(q, b)$  (see below) otherwise. If instead  $Prefix(i - 1, last_\perp, f)$  does not hold, then  $\mathcal{B}_{val}$  generates the action  $u$  and moves to the state  $p_{full}$ .
- $P_{main} = Q \times \{0, 1\}$ . States in  $P_{main}$  are associated with the main nodes of  $Ext\_VT_{q(C_\alpha),1}$  (corresponding to the main nodes of  $VT_{q(C_\alpha),1}$ ).

- $P_{choice} = (Q \times \{0, 1\}) \times (Q \times \{0, 1\})$ . States in  $P_{choice}$  are associated with the choice nodes of  $Ext\_VT_{q(C_\alpha), 1}$  (corresponding to the choice nodes of  $VT_{q(C_\alpha), 1}$ ).
- State  $p_\#$  is associated with the nodes of  $Ext\_VT_{q(C_\alpha), 1}$  corresponding to the leaf nodes of  $VT_{q(C_\alpha), 1}$ .

The initial state  $p_{val}^0$  is  $(q(C_\alpha), 1, 1, \perp, no)$  and the transition relation  $\Delta_{val}$  is defined as:

1. *Transitions from  $p_{full}$* : for each  $u \in Act$ , we have the transition  $p_{full} \xrightarrow{u} p_{full}$
2. *Transitions to generate the codes in Codes*: from each state  $(q, b, i, last_\perp, f) \in P_{cod}$  and for each  $u \in Act$ , we have the following transition:
  - *Prefix* $(i - 1, last_\perp, f, u)$  does not hold:  $(q, b, i, last_\perp, f) \xrightarrow{u} p_{full}$
  - *Prefix* $(i - 1, last_\perp, f, u)$  holds and  $i < 2n$ :  $(q, b, i, last_\perp, f) \xrightarrow{u} (q, b, i + 1, u, f')$  where  $f' = yes$  if  $u \in \{\tilde{1}, \dots, \tilde{n}\} \times (A \times Q)$ , and  $f' = f$  otherwise.
  - *Prefix* $(i - 1, last_\perp, f, u)$  holds and  $i = 2n$ :  $(q, b, i, last_\perp, f) \xrightarrow{u} (q, b)$
3. *Transitions from main states  $(q, b) \in Q \times \{0, 1\}$* :
  - $(q, b) \xrightarrow{q} ((q_1, b_1), (q_2, b_2))$  for each  $((q_1, b_1), (q_2, b_2)) \in Choices_{q, b}$ .
  - $(q, b) \xrightarrow{(q, b)} p_\#$
  - $(q, b) \xrightarrow{u} p_{full}$  for all  $u \in Act \setminus (Q \cup (Q \times \{0, 1\}))$ .
4. *Transitions from choice states  $((q_1, b_1), (q_2, b_2)) \in (Q \times \{0, 1\}) \times (Q \times \{0, 1\})$* :
  - $((q_1, b_1), (q_2, b_2)) \xrightarrow{L} (q_1, b_1, 1, \perp, no)$
  - $((q_1, b_1), (q_2, b_2)) \xrightarrow{R} (q_2, b_2, 1, \perp, no)$
  - $((q_1, b_1), (q_2, b_2)) \xrightarrow{u} p_{full}$  for each  $u \in Act \setminus \{L, R\}$ .
5. *Transitions from state  $p_\#$* :  $p_\# \xrightarrow{u} p_{full}$  for each  $u \in Act \setminus \{L, R\}$ .

Correctness of construction easily follows.

**Proof of Lemma 5.** The synchronization-free NFS  $\mathcal{S}_{SF}$  satisfying the statement of Lemma 5 is given by  $\mathcal{S}_{SF} = \langle Cell_1, \dots, Cell_n, Control \rangle_{SF}$ . Intuitively,  $Cell_j$  ( $1 \leq j \leq n$ ) keeps track by its finite control of the  $j$ -th symbol of a TM configuration, and it can generate only the actions of the form  $(j, u)$  (where  $u \in A \cup (A \times Q)$ ). Note that the action  $(j, u)$  corresponds to the 2nd symbol in the  $j$ -th pair  $(\tilde{j}, u), (j, u)$  of the code of some TM configuration. *Control* is instead used to model the control unit of  $\mathcal{M}$ , and it can generate only the actions in  $Act \setminus (\{1, \dots, n\} \times (A \cup (A \times Q)))$ . After having ‘correctly’ generated the code of a TM configuration  $C$ ,  $Cell_j$  is in state  $C(j)$  and *Control* is in a state which keeps track of the position  $i$  of the tape head of  $C$  together with the  $i$ -th symbol of  $C$ . Assume that  $C$  is not terminal. In order to generate for each  $dir \in \{L, R\}$ , the  $j$ -th pair  $(\tilde{j}, u), (j, u)$  ( $1 \leq j \leq n$ ) of the code of the  $dir$ -successor  $succ_{dir}(C)$  of  $C$ , the  $\mathcal{S}_{SF}$ -components behave as follows. Assume that  $j \neq i$  and  $j$  is *not* the position of the tape head in  $succ_{dir}(C)$  (the other cases are similar). Since *Control* keeps track of the pair  $(i, C(i))$  and the current position  $j$ , it can check (by using the transition function  $\delta$  of the TM  $\mathcal{M}$ ) whether this condition is satisfied or not. Note that in this case, the  $j$ -th symbol of  $succ_{dir}(C)$  coincides with the  $j$ -th symbol of  $C$ , i.e.  $u = C(j)$ , and, additionally  $u \in A$ . Then, *Control* guesses a pair  $(\tilde{j}, u')$  with  $u' \in A$  and generates it. Component  $Cell_j$ , which is in state  $C(j)$ , will be able to generate in the next step the matching

pair  $(j, u')$  iff  $u' = C(j)$ . In this way, the  $\mathcal{S}_{SF}$ -components ensure that for each choice  $dir \in \{L, R\}$ , exactly one code in  $Codes$  will be generated, and this code is precisely the encoding of  $succ_{dir}(C)$ . The crucial point is that even if other words of length  $2n$  will be generated (due to all the possible interleaving of the individual and asynchronous computational steps of the single components), exactly one of these words of length  $2n$  will be a code in  $Codes$ . The  $\mathcal{S}_{SF}$ -components are formally defined below.

$Cell_j = \langle \{j\} \times (A \cup (A \times Q)), \{p_j^0\} \cup A \cup (A \times Q), p_j^0, \Delta_j \rangle$ , where  $\Delta_j$  is defined as:

1.  $p_j^0 \xrightarrow{(j, C_\alpha(j))} C_\alpha(j)$
2.  $a \xrightarrow{(j, a)} a$  and  $a \xrightarrow{(j, a, q)} (a, q)$  for all  $a \in A$  and  $q \in Q$
3.  $(a, q) \xrightarrow{(j, a')} a'$  for all  $a, a' \in A$  and  $q \in Q$

The first transition is used to generate the 2nd symbol of the  $j$ -th pair of the code of the initial TM configuration  $C_\alpha$ . Transitions of type 2 (resp., 3) are used to generate the 2nd symbol of the  $j$ -th pair of the code of the *next* TM configuration when the tape head is at position  $i \neq j$  (resp.,  $i = j$ ). Note that the source state of transitions of type 2–3 represents the  $j$ -th symbol of the current TM configuration.

$Control = \langle Act \setminus (\{1, \dots, n\} \times (A \cup (A \times Q))), P, 1, \Delta \rangle$  is defined as follows. The set of states is given by  $P = \{p_{fin}\} \cup P_{init} \cup P_{conf} \cup P_{main} \cup P_{choice}$ , where:

- $p_{fin}$  is a state with no outgoing transitions.
- $P_{init} = \{1, \dots, n\}$ . States in  $P_{init}$  are used to generate the code of  $C_\alpha$ , and 1 is the initial state.
- $P_{conf}$  consists of states of the form  $p_{conf} = (j, (i, a, q), a_\perp, dir)$ , where  $1 \leq j, i \leq n$ ,  $a \in A$ ,  $q \in Q \setminus \{q_{acc}, q_{rej}\}$ ,  $a_\perp \in A \cup \{\perp\}$  ( $\perp$  is for undefined), and  $dir \in \{L, R\}$ . Intuitively,  $i$  is the position of the tape head for the current non-terminal TM configuration  $C$  and  $(a, q) = C(i)$ . Let  $\delta(q, a) = \langle (q_L, a_L, \theta_L), (q_R, a_R, \theta_R) \rangle$ . Then, from state  $p_{conf}$ ,  $Control$  guesses and generates an action of the form  $(\tilde{j}, u)$  with the constraint that  $u = a_{dir}$  if  $j = i$ ,  $u = (a', q_{dir})$  for some  $a' \in A$  if  $j = i + \theta_{dir}$ , and  $u \in A$  if  $j \notin \{i, i + \theta_{dir}\}$  (note that  $a_{dir}$  is the  $i$ -th symbol of  $succ_{dir}(C)$  and  $i + \theta_{dir}$  is the position of the tape head in  $succ_{dir}(C)$ ). If the guess is correct (i.e.,  $u$  is the  $j$ -th symbol of  $succ_{dir}(C)$ ), then  $Cell_j$  is able to generate the matching action  $(j, u)$  in the next step. In this case,  $a_\perp$  is the content of the  $succ_{dir}(C)$ -cell pointed by the tape head if the position of this cell is smaller than  $j$ , and  $a_\perp = \perp$  otherwise.
- $P_{main}$  consists of states of the form  $p_{main} = (main, (i, a, q))$ , where  $1 \leq i \leq n$ ,  $a \in A$ , and  $q \in Q$ . Intuitively,  $i$  represents the position of the tape head for the new generated TM configuration  $C$  and  $(a, q) = C(i)$ . From state  $p_{main}$ ,  $Control$  moves to the choice state  $(choice, (i, a, q))$  (see below) by generating the action  $q$  if  $q \notin \{q_{acc}, q_{rej}\}$ , and to the state  $p_{fin}$  by generating the action  $(q, Val(q))$  otherwise.
- $P_{choice}$  consists of states of the form  $(choice, (i, a, q))$ , where  $(i, a, q)$  has the same meaning as above. From these states,  $Control$  generates the two actions  $L$  and  $R$ .

The transition relation  $\Delta$  of  $Control$  is defined as follows.

1. *Initialization (transitions to generate the code of  $C_\alpha$ ):*

- $1 \xrightarrow{(\tilde{1}, C_\alpha(1))} 2, \dots, n \xrightarrow{(\tilde{n}, C_\alpha(n))} (\text{main}, (1, \alpha(0), q_0))$
- 2. *Transitions to generate the next TM configuration:* from each state  $(j, (i, a, q), a_\perp, \text{dir}) \in P_{\text{conf}}$  with  $\delta(q, a) = \langle (q_L, a_L, \theta_L), (q_R, a_R, \theta_R) \rangle$  and  $1 \leq i + \theta_{\text{dir}} \leq n$ , and for each  $a' \in A$  such that  $a' = a_{\text{dir}}$  if  $j = i$ , we have the following transitions:
  - $j \neq i + \theta_{\text{dir}}, j < n$ :  $(j, (i, a, q), a_\perp, \text{dir}) \xrightarrow{(\tilde{j}, a')} (j+1, (i, a, q), a_\perp, \text{dir})$
  - $j \neq i + \theta_{\text{dir}}, j = n, a_\perp \neq \perp$ :  $(n, (i, a, q), a_\perp, \text{dir}) \xrightarrow{(\tilde{n}, a')} (\text{main}, (i + \theta_{\text{dir}}, a_\perp, q_{\text{dir}}))$
  - $j = i + \theta_{\text{dir}}, j < n$ :  $(j, (i, a, q), a_\perp, \text{dir}) \xrightarrow{(\tilde{j}, a', q_{\text{dir}})} (j+1, (i, a, q), a', \text{dir})$
  - $j = i + \theta_{\text{dir}} = n$ :  $(n, (i, a, q), a_\perp, \text{dir}) \xrightarrow{(\tilde{n}, a', q_{\text{dir}})} (\text{main}, (n, a', q_{\text{dir}}))$
- 3. *Transitions from main states*  $(\text{main}, (i, a, q)) \in P_{\text{main}}$ :
  - $q \notin \{q_{\text{acc}}, q_{\text{rej}}\}$ :  $(\text{main}, (i, a, q)) \xrightarrow{q} (\text{choice}, (i, a, q))$
  - $q \in \{q_{\text{acc}}, q_{\text{rej}}\}$ :  $(\text{main}, (i, a, q)) \xrightarrow{(q, \text{Val}(q))} p_{\text{fin}}$
- 4. *Transitions from choice states*  $(\text{choice}, (i, a, q)) \in P_{\text{choice}}$ :
  - $(\text{choice}, (i, a, q)) \xrightarrow{L} (1, (i, a, q), \perp, L)$  and  $(\text{choice}, (i, a, q)) \xrightarrow{R} (1, (i, a, q), \perp, R)$

Now, we prove that the construction is correct. Let  $C$  be a *non-terminal good* TM configuration and  $\text{dir} \in \{L, R\}$ . The *starting*  $(C, \text{dir})$ -state is the state of  $S_{SF}$  in which component  $\text{Cell}_j$  is in state  $C(j)$  and *Control* is in the state  $(1, (i, a, q), \perp, \text{dir}) \in P_{\text{conf}}$ , where  $i$  is the position of the tape head in  $C$  and  $(a, q) = C(i)$ . Moreover, for each  $\text{dir} \in \{L, R\}$ , the starting  $(\perp, \text{dir})$ -state is the initial state of  $S_{SF}$ . By construction, the following result easily follows.

**Lemma 6.** *Fix a starting  $(C_\perp, \text{dir})$ -state  $p_{\text{start}}$ , and let  $C = C_\alpha$  if  $C_\perp = \perp$ , and  $C = \text{succ}_{\text{dir}}(C_\perp)$  otherwise. Then, there is a unique path  $\pi$  of  $S_{SF}$  starting from  $p_{\text{start}}$  labeled by a code  $\in \text{Codes}$ . Moreover,  $\text{code} = \text{code}(C)$  and there is exactly one transition  $p \xrightarrow{u} p'$  from the last state  $p$  of  $\pi$  labeled by an action  $u \in Q \cup (Q \times \{0, 1\})$ . Furthermore, the action  $u$  and state  $p'$  satisfies the following:*

- $C$  is terminal:  $u = (q(C), \text{Val}(q(C)))$  and there is no transition outgoing from  $p'$  labeled by an action in  $\{L, R\}$ .
- $C$  is not terminal:  $u = q(C)$  and there are exactly two transitions from state  $p'$  labeled by actions in  $\{L, R\}$ . Moreover, one, labeled by  $L$ , leads to the starting  $(C, L)$ -state, and the other one, labeled by  $R$ , leads to the starting  $(C, R)$ -state.

By Definition 4 and Lemma 6, we obtain the desired result.

**Corollary 1 (Correctness).** *The unwinding of  $S_{SF}$  from its initial state is in  $\text{Ext}(ET_{C_\alpha})$ .*

Note that the components of  $S_{SF}$  are deterministic and with pairwise disjoint alphabets of actions. Moreover, the size of  $S_{SF}$  is polynomial in the sizes of the TM  $\mathcal{M}$  and input  $\alpha$ . Thus, by the above corollary, Lemma 5 follows.

## 4 Additional hardness results

We can show that the deterministic versions of Problems 1 and 2 are PSPACE-hard by polynomial-time reductions from the word problem for *linearly-bounded deterministic Turing Machines*. The proposed constructions can be seen as a simplification of those illustrated in the previous section.

**Theorem 4.** *The deterministic versions of Problems 1 and 2 are PSPACE-hard, and for Problems 2, PSPACE-hardness holds even if the components of the synchronization-free non-flat system are assumed to be deterministic and with pairwise disjoint alphabets.*

The rest of this section is devoted to the proof of the following theorem.

**Theorem 5.** *For any relation  $\leq$  on LTS lying between trace containment and bisimulation equivalence, checking whether  $\llbracket S \rrbracket \leq \mathcal{A}$  for a given synchronous NFS  $S$  and a FS  $\mathcal{A}$  is PSPACE-hard even if  $\mathcal{A}$  and the  $S$ -components are assumed to be deterministic.*

**Proof of Theorem 5.** By a polynomial-time reduction from the acceptance problem for *non-halting linearly-bounded deterministic Turing Machines* (TM). Fix such a TM machine  $\mathcal{M} = \langle A, Q, q_0, \delta, \{q_{acc}\} \rangle$ , where  $A, Q, q_0, q_{acc}$  (with  $q_0 \neq q_{acc}$ ) are as for alternating Turing Machines, and  $\delta: Q \times A \rightarrow (Q \times A \times \{+1, -1\})$  is the transition function, where  $+1$  (resp.,  $-1$ ) denotes a right (resp., left) tape head move. Fix an input  $\alpha \in A^*$  and let  $n = |\alpha|$ . Since  $\mathcal{M}$  is linearly bounded, we can assume that a TM configuration (of  $\mathcal{M}$  over  $\alpha$ ) is a word  $C = \beta_1, (a, q), \beta_2 \in A^* \cdot (A \times Q) \cdot A^*$  of length exactly  $n$ .  $\mathcal{M}$  accepts  $\alpha$  iff the unique (infinite) computation of  $\mathcal{M}$  over  $\alpha$  visits an accepting configuration. W.l.o.g. we assume that the alphabet  $A$  contains a special symbol, say  $\#$ , such that if the computation of  $\mathcal{M}$  over  $\alpha$  visits an accepting configuration  $C_{acc}$ , then  $C_{acc}$  is  $\#$ -homogeneous, i.e. the content of each cell of  $C_{acc}$  is the special symbol  $\#$ .

**Preliminary step: encoding of acceptance.** Let  $Act = (\{1, \dots, n\} \times (A \cup (A \times Q))) \cup \{\flat\}$ , where  $\flat$  is a special action. For each TM configuration  $C$ ,  $code(C)$  is the word over  $Act \setminus \{\flat\}$  given by  $(1, C(1)), \dots, (n, C(n))$ . Let  $Codes$  be the finite set of these codes and  $T_{Codes}$  be the deterministic tree encoding of  $Codes$  (as defined in Section 3).

**Definition 6 (Valuation tree).** *The valuation tree  $T_{val}$  is the infinite  $Act \setminus \{\flat\}$ -labeled tree obtained as the limit of the sequence of finite trees  $(T_{Codes}^k)_{k \in \mathbb{N}}$ , where:  $T_{Codes}^0 = T_{Codes}$  and  $T_{Codes}^{k+1}$  results from rooting a fresh copy of  $T_{Codes}$  at each leaf of  $T_{Codes}^k$ .*

Note that  $T_{val}$  is a deterministic and all its maximal paths from the root are infinite and labeled by concatenations of codes of TM configurations. The *special path* of  $T_{val}$  is the unique maximal path from the root whose sequence of labels  $code(C_1) \cdot code(C_2) \dots$  is such that  $C_\alpha, C_1, C_2, \dots$  is the computation of  $\mathcal{M}$  over  $\alpha$ .

**Definition 7 (Emulation tree).** *The deterministic  $Act$ -labeled emulation tree  $T_{em}$  is defined as follows. Let  $\pi$  be the special path of  $T_{val}$  and let  $code(C_1) \cdot code(C_2) \dots$  be its sequence of labels. For each  $i \geq 1$  such that  $C_i$  is an accepting  $\#$ -homogeneous configuration, let  $x_i \xrightarrow{u_i} y_i$  be the edge along  $\pi$  associated with the last symbol of  $code(C_i)$ . Then,  $T_{em}$  is obtained from  $T_{val}$  by adding for each of these edges  $x_i \xrightarrow{u_i} y_i$  (if any), a new edge labeled by the special action  $\flat$  from  $y_i$  to a new leaf node.*

**Fact:** if  $\mathcal{M}$  does not accept  $\alpha$ , then  $T_{em} = T_{val}$ . Otherwise,  $Tr(T_{em}) \not\subseteq Tr(T_{val})$ .

**Final step.** Theorem 5 directly follows from the fact above and the following two Lemmata. The proof of Lemma 7 is trivial and we omit it.

**Lemma 7.** *One can construct a deterministic FS  $\mathcal{D}_{val}$  over Act of size polynomial in the sizes of  $\mathcal{M}$  and  $\alpha$  such that the unwinding of  $\mathcal{D}_{val}$  from its initial state is  $T_{val}$ .*

**Lemma 8.** *One can construct a synchronous NFS  $\mathcal{S}_{em}$  over Act (whose components are deterministic) of size polynomial in the sizes of  $\mathcal{M}$  and  $\alpha$  such that the unwinding of  $\llbracket \mathcal{S}_{em} \rrbracket$  from its initial state is  $T_{em}$ .*

*Sketched proof.*  $\mathcal{S}_{em}$  is a synchronous composition of  $n+1$  components  $Cell_1, \dots, Cell_n$ , and  $Control$ . Intuitively,  $Cell_j$  ( $1 \leq j \leq n$ ) keeps track by its finite control of the  $j$ -th symbol of a TM configuration, and its alphabet is  $\{b\} \cup (\{j\} \times (A \cup (A \times Q)))$ .  $Control$  is instead used to model the control unit of  $\mathcal{M}$ , and its alphabet is Act (hence,  $Control$  participates in each transition of  $\mathcal{S}_{em}$ ). After having ‘correctly’ generated a TM configuration  $C$  (this, intuitively, means that  $\mathcal{S}_{em}$  is emulating the computation of  $\mathcal{M}$  over  $\alpha$ , i.e., the computational path from the initial state to the current state of  $\mathcal{S}_{em}$  corresponds to a prefix of the special path of  $T_{val}$ ),  $\mathcal{S}_{em}$  is in a *starting good* state  $s_{good}$  such that component  $Cell_j$  is in state  $C(j)$  and  $Control$  is in a good local state which keeps track of the position  $k$  of the tape head of  $C$  together with the  $k$ -th symbol of  $C$ .<sup>2</sup> From state  $s_{good}$ ,  $\mathcal{S}_{em}$  generates in  $n$  steps by ‘computational nondeterminism’ all the possible codes in  $Codes$  as follows. At the  $j$ -th step,  $Control$  guesses an action of the form  $(j, u)$  and generates it by binary synchronization with  $Cell_j$ . Note that since  $Cell_j$  is in state  $C(j)$  and  $Control$  keeps track of  $k$  and  $C(k)$ , either  $Cell_j$  or  $Control$  is able to detect whether for the communication action  $(j, u)$ ,  $u$  is *not* the  $j$ -th symbol of the successor  $succ(C)$  of  $C$ . If it is the case, and  $Cell_j$  (resp.,  $Control$ ) has detected it, then  $Cell_j$  (resp.,  $Control$ ) will move to the local bad state  $bad$  (resp., to a local bad state  $p_{bad, j'}$  associated with the step  $j' = (j+1) \bmod n$ ). From state  $bad$ ,  $Cell_j$  remaining in  $bad$  may generate all the actions of its alphabet except  $b$ . From the bad state  $p_{bad, j'}$ ,  $Control$  can generate only the actions of the form  $(j', u)$  and move to a local bad state associated with the step  $(j'+1) \bmod n$ . If instead for the generated action  $(j, u)$ ,  $u$  is the  $j$ -th symbol of  $succ(C)$ , then  $Control$  will move to a good local state associated with the step  $j' = (j+1) \bmod n$  and  $Cell_j$  will move to the good local state  $[succ(C)](j)$ . Thus, after having generated the code of  $succ(C)$ ,  $\mathcal{S}_{em}$  will be in the starting good state  $s'_{good}$  associated with  $succ(C)$ . From  $s'_{good}$ ,  $\mathcal{S}_{em}$  can generate the special action  $b$  (by synchronization among all its components)<sup>3</sup> iff  $succ(C)$  is an accepting #-homogeneous configuration. If the action  $b$  is generated, then the target state has no outgoing transition.  $\square$

## 5 Conclusions

As future research, there is an interesting question left open: the exact complexity of bisimulation checking between a flat system and a non-flat system. Our contribution (Theorem 5) shows that the problem is PSPACE-hard even for synchronous composition without hiding. Note that the problem is in EXPTIME. We believe that filling this

<sup>2</sup> Initially,  $\mathcal{S}_{em}$  is in the starting good state associated with the initial TM configuration

<sup>3</sup> note that  $b$  is in the alphabet of each component

gap is a very difficult question. Simple settings are however tractable: Muscholl and Walukiewicz [13] have recently shown that bisimulation checking can be solved in NLOGSPACE when there is no synchronization and both the flat system and the non-flat system components are deterministic. It would be interesting to investigate the non-deterministic framework.

## References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. J.L. Balcázar, J. Gabarró, and M. Santha. Deciding bisimilarity is p-complete. *Formal Asp. Comput.*, 4(6A):638–648, 1992.
3. A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
4. O. Kupferman, D. Harel and M.Y. Vardi. On the complexity of verifying concurrent transition systems. In *Proc. 8th CONCUR*, LNCS 1243, pages 258–272. Springer-Verlag, 1997.
5. J.F. Groote and F. Moller. Verification of parallel systems via decomposition. In *Proc. 3rd CONCUR*, LNCS 630, pages 62–76. Springer-Verlag, 1992.
6. M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. 36th FOCS*, pages 453–462. IEEE Computer Society, 1995.
7. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1984.
8. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
9. L. Jategaonkar and A.R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154(1):107–143, 1996.
10. A. Kučera and P. Jančar. Equivalence-checking on infinite-state systems: Techniques and results. *Theory and Practice of Logic Programming*, 6(3):227–264, 2006.
11. F. Laroussinie and Ph. Schnoebelen. The state explosion problem from trace to bisimulation equivalence. In *Proc. 3rd FOSSACS*, LNCS 1784, pages 192–207, 2000.
12. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
13. A. Muscholl and I. Walukiewicz. A lower bound on web services composition. In *Proc. 10th FOSSACS*, LNCS 4423, pages 274–286, 2007.
14. A.M. Rabinovich. Complexity of equivalence problems for concurrent systems of finite agents. *Information and Computation*, 139(2):111–129, 1997.
15. Z. Sawa and P. Jančar. Behavioural equivalences on finite-state systems are PTIME-hard. *Computing and Informatics*, 24(5), 2005.
16. Z. Sawa and P. Jančar. Hardness of equivalence checking for composed finite-state systems. *Acta Informatica*, 46(3):169–191, 2009.
17. S.K. Shukla, H.B. Hunt III, D.J. Rosenkrantz, and R.E. Stearns. On the complexity of relational problems for finite state processes. In *Proc. 23rd ICALP*, LNCS 1099, pages 466–477. Springer-Verlag, 1996.
18. S. Tasiran, R. Alur, R.P. Kurshan, and R.K. Brayton. Verifying abstractions of timed systems. In *Proc. 7th CONCUR*, LNCS 1119, pages 546–562. Springer, 1996.
19. K. Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In *Proc. 4th CAV*, LNCS 663, pages 302–315, 1992.
20. A. Valmari and A. Kervinen. Alphabet-based synchronisation is exponentially cheaper. In *Proc. 13th CONCUR*, LNCS 2421, pages 161–176. Springer-Verlag, 2002.
21. R.J. van Glabbeek. The linear time-branching time spectrum. In *Proc. 1st CONCUR*, LNCS 458, pages 278–297. Springer, 1990.