A Comparison of Succinctly Represented Finite-state Systems

Romain Brenguier, Stefan Göller, Ocan Sankur

June 2012

Research report LSV-12-13

Laboratoire Spécification & Vérification

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

A Comparison of Succinctly Represented Finite-state Systems

Romain Brenguier[1], Stefan Göller[2], and Ocan Sankur[1]

[1] LSV, CNRS & ENS Cachan, France.
[2] Institut für Informatik, Universität Bremen, Germany

Abstract. We study the succinctness of different classes of succinctly presented finite transition systems with respect to bisimulation equivalence. Our results show that synchronized product of finite automata, hierarchical graphs, and timed automata are pairwise incomparable in this sense. We moreover study the computational complexity of deciding simulation preorder and bisimulation equivalence on these classes.

1 Introduction

In formal verification *model checking* is one of the most successful approaches; it asks to test automatically whether a given system meets a given specification. Unfortunately, model checking tools have to deal with a combinatorial blow up of the state space, commonly known as the *state explosion problem*, that can be seen as one of the biggest challenges in real-world problems. Different sources of explosion arise, for instance the number of *program variables* or *clocks*, the number of *concurrent components*, or the number of different *subroutines*, just to mention few of them. Numerous techniques to tame the state explosion problem have been introduced such as abstraction methods, partial order reduction or counterexample guided abstraction refinement.

Flip side of the coin, when modeling everyday systems that are potentially exponentially big (also called the *flattened system* or just *flat system*), it is desirable to have succinct representations for them. Three fundamental models include (i) *products of flat systems*, (ii) *timed automata* (more precisely the transitions systems evolving from the time-abstract semantics of timed automata), and (iii) *hierarchical systems*, each of them successfully being used to tame the state explosion problem in their dimension (these dimensions are pairwise orthogonal): (i) Products of flat systems allow to succinctly account for the number of concurrently running components, (ii) Timed automata [2] allow to succinctly model the behavior of programs involving program variables or clocks, and finally (iii) hierarchical systems (also known as hierarchical state machines [3] or hierarchical structures [16]) allow to succinctly represent systems that are decomposed from numerous sub-systems. See also [18] for a recent work, where *web services* are modeled as the asynchronous product of flat systems.

An important algorithmic question in this context is whether two given (succinctly presented) systems behave equivalently, or whether one system can be

simulated by another one. For instance, if it turns out that a complex system (implementation) is behaviorally equivalent to a simple system (implementation), the system designer can well replace the complex one by the simple one.

Numerous notions of behavioral equivalences have been proposed by van Glabbeek [21, 22]. Among them, *bisimulation equivalence* is undoubtedly the central one of them in formal verification. For instance beautiful characterizations of the bisimulation-invariant fragments of established logics such as first-order logic and monadic second-order have been proven in terms of modal logic [20] and the modal μ-calculus [8], respectively; we refer to [17] for a further such characterization in terms of CTL*.

Our contributions and related work. In the first part of this paper we study the succinctness with respect to bisimulation equivalence of three established models of succinctly representing finite systems, namely *products of flat systems*, *timed automata*, and *hierarchical systems*. The main contribution of this paper is to pinpoint to the sources of succinctness when comparing any two of the (orthogonally defined) three models of systems, mainly focusing on the different proof ideas for establishing exponential succinctness. We show that each of the three models *can* be exponentially more succinct than any of the other two. Such a rigorous comparison of fundamental models for succinctly representing finite systems has not yet been carried out in the context of formal verification to the best of the authors' knowledge.

In the second part of this paper we study the computational complexity of simulation preorder and bisimulation equivalence checking for products of flat systems, timed systems and hierarchical systems. We provide a general reduction that shows EXPTIME-hardness (and thus completeness) of checking simulation preorder on hierarchical systems and on timed automata. The former is a new result; the latter was already proven in [14] but we believe our proof is more direct. Moreover, our reduction is quite generic, and can be easily applied to a wide range of succinctly presented models.

We also study the problem of deciding simulation preorder and bisimulation equivalence between one of the three above-mentioned succinct systems and a flat system. We show that checking the simulation preorder of a hierarchical system by a flat system is PSPACE-complete. The problem is known to be EXPTIME-complete for (synchronization-free) non-flat systems and timed automata [5].

Via a standard reduction to model checking EF logic we describe a PSPACE algorithm to check bisimilarity of any of the discussed succinctly presented finite systems and a flat system. Essentially since reachability of all of these systems is in PSPACE, it follows that the problem is PSPACE-complete; the upper bound was left open in [5], where it was shown to be PSPACE-hard.

Finally, we study language inclusion between the three above-mentioned succinct models. We show that checking untimed language equivalence (and in fact language universality) is EXPSPACE-hard (and thus EXPSPACE-complete) for hierarchical systems and timed automata. We would like to mention that this problem has been wrongly cited in the literature as being in PSPACE for timed automata [1, 19]. Our results are summarized in Table 1.

Table 1. Complexity results

	Hierarchical	Timed	Prod. of Flat
Simulation	EXPTIME-c	EXPTIME-c [14]	EXPTIME-c [7]
Bisimulation	PSPACE-hard	EXPTIME-c [14]	EXPTIME-c [9]
Bis. with Flat	PSPACE-c	PSPACE-c	PSPACE-c [5]
Sim. by Flat	PSPACE-c	EXPTIME-c [5]	EXPTIME-c [7, 5]
Language Inc.	EXPSPACE-c	EXPSPACE-c	EXPSPACE-c [14]

2 Definitions

A *transition system* (also *flat system* or just *system*) over a finite alphabet Σ is a tuple $\mathcal{T} = (S, (\xrightarrow{\sigma})_{\sigma\in\Sigma})$, where S is an arbitrary set of *states* and each relation $\xrightarrow{\sigma} \subseteq S \times S$, is the set of σ-labeled *transitions*. Its size is $|\mathcal{T}| = |S| + \sum_\sigma |\xrightarrow{\sigma}|$. We say that an *action* $\sigma \in \Sigma$ is *enabled* at state $s \in S$ if there is a transition $s \xrightarrow{\sigma} s'$ for some $s' \in S$. $\mathcal{T}$ is *deterministic* if each $\xrightarrow{\sigma}$ is a partial function. An *initialized transition system* is $(S, s_0, (\xrightarrow{\sigma})_{\sigma\in\Sigma})$, where $s_0 \in S$ is the *initial state*. A *simulation* is a relation $R \subseteq S \times S$, with the following property: for any states $s, t \in S$ with sRt, for any $\sigma \in \Sigma$ and $s' \in S$ such that $s \xrightarrow{\sigma} s'$, there exists $t' \in S$ with $t \xrightarrow{\sigma} t'$ and $s'Rt'$. A simulation is a *bisimulation* whenever it is symmetric. For two states $s, t \in S$, we write $s \sqsubseteq t$ (resp. $s \sim t$) if there exists a simulation (resp. bisimulation) $R \subseteq S \times S$ such that sRt. An initialized transition system $\mathcal{T} = (S, s_0, (\xrightarrow{\sigma})_{\sigma\in\Sigma})$ is simulated by an initialized transition system $\mathcal{T}' = (S', s'_0, (\xrightarrow{\sigma}{}')_{\sigma\in\Sigma})$, if there is a simulation R in the disjoint union of $\mathcal{T}$ and $\mathcal{T}'$ such that $s_0 R s'_0$. We extend notations $\sqsubseteq$ and $\sim$ to initialized transition systems. We also define $\sim_k$, *bisimilarity up-to k steps*: we have $s \sim_k t$ for two states $s, t \in S$ if, and only if the unfolding of $\mathcal{T}$ from s up-to k steps is bisimilar to the unfolding at t up-to k steps. A path of $\mathcal{T}$ is a sequence of states that are connected by transitions. The length of a path of a transition system is the number of transitions it contains. For a path π, π_i denotes the i-th state it visits, and we denote by $\pi_{i\ldots j}$ the subpath of π from π_i to π_j. For any initialized transition system $\mathcal{T}$, we define $L(\mathcal{T})$ as the *language* accepted by $\mathcal{T}$, that is the set of words made of the transition labels in all paths of $\mathcal{T}$ starting at s_0.

A *product of flat systems* is a tuple $\mathcal{S} = (\mathcal{T}_1, \ldots, \mathcal{T}_k)$, where $\mathcal{T}_i = (S_i, (\xrightarrow{\sigma}_i)_{\sigma\in\Sigma})$ is a flat system for each $1 \leq i \leq k$. $\mathcal{S}$ defines a transition system $\mathcal{T}(\mathcal{S}) = (\prod_i S_i, (\xrightarrow{\sigma})_{\sigma\in\Sigma})$, where $(s_i)_i \xrightarrow{\sigma} (t_i)_i$ if, and only if, for all $1 \leq i \leq k$, either $s_i \xrightarrow{\sigma} t_i$ in $\mathcal{T}_i$, or $t_i = s_i$ and σ is not enabled at s_i. An example is given in Fig. 1.

Hierarchical systems are a modeling formalism used to succinctly describe finite systems, by allowing the reuse of subsystems. A hierarchical system is defined by a simple grammar that generates a single transition system, in which each nonterminal defines a system by explicitly introducing states and transitions, and using other nonterminals. The reuse relation is required to be acyclic, so as to ensure that the generated transition system is finite. These were introduced in [15] in the context of VLSI design.

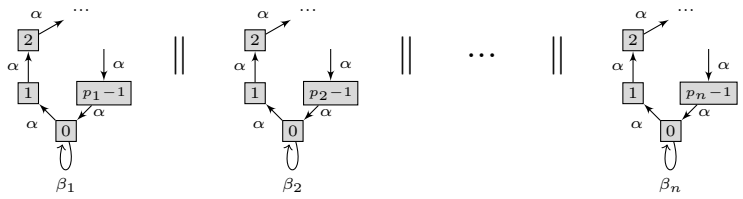

Fig. 1. The system A_n, where $p_1, \ldots, p_n$ are the first n prime numbers, is defined as the product of components F_i made of a α-cycle of length p_i, along where each state corresponds to a value modulo p_i. The self-loop β_i is only available at state 0, which is also the initial state. Then, when the system reads a word α^m, one can read the values $m \mod p_i$ for all $1 \leq i \leq n$, looking at the states of all components.

An *n-pointed system* is a transition system with n selected states, numbered from 1 to n. It is denoted by a pair $(\mathcal{T}, \tau)$, where $\mathcal{T} = (S, (\xrightarrow{\sigma})_{\sigma \in \Sigma})$ is a transition system and $\tau : \{1, \ldots, n\} \to S$ an injection.

Definition 1. *A* hierarchical system *[16] is a tuple $H = (N, I, P)$ where*

1. *N is a finite set of* nonterminals*. Each $B \in N$ has a* rank *denoted by $\mathsf{rank}(B) \in \mathbb{N}$. I is the initial nonterminal with $\mathsf{rank}(I) = 0$.*
2. *P is the set of* productions*, that contains for each $B \in N$ a unique production $B \to (\mathcal{A}, \tau, E)$ where $(\mathcal{A}, \tau)$ is a $\mathsf{rank}(B)$-pointed system with the set of states A, and E is the set of* references *with $E \subseteq \{(B', \sigma) \mid B' \in N, \sigma : \{1, \ldots, \mathsf{rank}(B')\} \to A$ is injective$\}$.*
3. *Define relation $\mathcal{E}_H \subseteq N \times N$ as follows: $(B, C) \in \mathcal{E}_H$ if, and only if for the unique production $B \to (\mathcal{A}, \tau, E)$, E contains some reference of the form (C, σ). We require that $\mathcal{E}_H$ is acyclic.*

Its size is defined as $|H| = \sum_{(B \to (\mathcal{A}, \tau, E)) \in P} |\mathcal{A}| + |E|$. For any production $B \to (\mathcal{A}, \tau, E)$, the states $\tau(i)$ are called *contact states*. Each production produces an n-pointed system, that is, a finite system with n contact states. In fact, a hierarchical system $H = (N, I, P)$ describes a single finite system, obtained by taking, for each production $B \to (\mathcal{A}, \tau, E)$, the disjoint union of the (explicitly given) system $\mathcal{A}$ and those systems defined by nonterminals B' for all references $(B', \sigma) \in E$, and merging the i-th contact state of B' with $\sigma(i)$. Thus, the function σ is used to merge the contact states of the references with the states at the current level. Figure 2 gives an example of a hierarchical system.

Formally, each nonterminal B, produces a $\mathsf{rank}(B)$-pointed system denoted $\mathsf{eval}_H(B)$ (also written as $\mathsf{eval}(B)$ in the rest) as follows. If the production $B \to (\mathcal{A}, \tau, E)$ satisfies $E = \varnothing$, then $\mathsf{eval}(B)$ is the $\mathsf{rank}(B)$-pointed system $(\mathcal{A}, \tau)$. Otherwise, let $E = \{(B_1, \sigma_1), \ldots, (B_k, \sigma_k)\}$ and consider systems $\mathsf{eval}(B_i) = (\mathcal{A}_i, \tau_i)$ for each i. Let $\mathcal{U}$ denote the disjoint union of all $\mathcal{A}_i$ and $\mathcal{A}$. We let $\mathsf{eval}(B) = (\mathcal{U}/_{\equiv}, \pi_{\equiv} \circ \tau)$, where $\equiv$ is the equivalence relation generated by $\{(\sigma_i(j), \tau_i(j)), 1 \leq i \leq k, 1 \leq j \leq \mathsf{rank}(B_i)\}$, and $\pi_{\equiv}$ is the projection to the equivalence classes. Thus, $\equiv$ merges contact state j of system $\mathcal{A}_i$ with the state $\sigma_i(j)$, for each

$1 \leq i \leq k$. Note that $\mathsf{eval}(B)$ is well-defined since $\mathcal{E}_H$ is acyclic. We define the generated transition system $\mathcal{T}(H)$ of H as $\mathsf{eval}(I)$.

We denote by $\mathsf{unfolding}(H)$ the tree defined as follows. States are labeled by nonterminals, and the root is the initial nonterminal S. The children of each state labeled by nonterminal B are given as follows. If $B \to (\mathcal{A}, \tau, E)$ is the production of nonterminal B, and if $(B_1, \sigma_1), \ldots, (B_k, \sigma_k)$ are the references in E, then B has a child for each $1 \leq i \leq k$, labeled by B_i. Observe that for each state v of $\mathsf{eval}(H)$, there is a unique state in $\mathsf{unfolding}(H)$ labeled by a nonterminal $B \to (\mathcal{A}, \tau, E)$, such that v is an internal state in $\mathcal{A}$, *i.e.* $v \in \mathcal{A} \setminus \mathsf{range}(\tau)$. We denote this state by $\mathsf{unfolding}(H, v)$

For any nonterminal B in H, an *inner path in* B is a path of $\mathsf{eval}(B)$ that does not contain any contact states of $\mathsf{eval}(B)$, except possibly for the first and the last states. An inner path of B is *traversing* if its first and last states are contact states of $\mathsf{eval}(B)$.

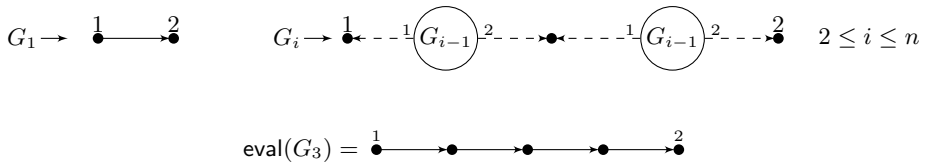

Fig. 2. The figure shows a hierarchical system with nonterminals G_i for $1 \leq i \leq n$. G_1 produces an explicit system with no references, with two contact states (shown by 1 and 2). G_i creates three states, where the leftmost and the rightmost are two contact states, and uses two references to G_{i-1}. The dashed arrows show how to merge the contact states of each copy G_{i-1} with the states of G_i. For instance, the contact state 1 of the leftmost copy of G_{i-1} is merged with contact state 1 of G_i. Then, for $n = 3$, $\mathsf{eval}(G_3)$ is the system depicted on the bottom.

Timed automata are finite automata equipped with a finite set of real-valued clocks. Clocks grow at a constant rate, and are used to enable/disable the transitions of the underlying finite automaton. They can be reset during transitions.

To formally define timed automata, we need the following notations. Given a finite set of clocks $\mathcal{X}$, we call *valuations* the elements of $\mathbb{R}_{\geq 0}^{\mathcal{X}}$. For a subset $R \subseteq \mathcal{X}$ and a valuation v, $v[R \leftarrow 0]$ is the valuation defined by $v[R \leftarrow 0](x) = v(x)$ for $x \in \mathcal{X} \setminus R$ and $v[R \leftarrow 0](x) = 0$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$ and a valuation v, the valuation $v + d$ is defined by $(v + d)(x) = v(x) + d$ for all $x \in \mathcal{X}$. We extend these operations to sets of valuations in the obvious way. We write $\mathbf{0}$ for the valuation that assigns 0 to every clock. An atomic clock constraint is a formula of the form $k \preceq x \preceq' l$ or $k \preceq x - y \preceq' l$ where $x, y \in \mathcal{X}$, $k, l \in \mathbb{Z} \cup \{-\infty, \infty\}$ and $\preceq, \preceq' \in \{<, \leq\}$. *Guards* are conjunctions of atomic clock constraints. The set $\Phi_{\mathcal{X}}$ denotes the guards over clocks $\mathcal{X}$. A valuation v satisfies a guard g, denoted $v \models g$, if all constraints are satisfied when each $x \in \mathcal{X}$ is replaced with $v(x)$.

Definition 2 ([2]). *A* timed automaton $\mathcal{A}$ *is a tuple* $(\mathcal{L}, \Sigma, \mathcal{X}, \ell_0, E)$*, consisting of finite sets* $\mathcal{L}$ *of locations, a finite alphabet* Σ*,* $\mathcal{X}$ *of clocks,* $E \subseteq \mathcal{L} \times \Phi_{\mathcal{X}} \times \Sigma \times 2^{\mathcal{X}} \times \mathcal{L}$ *of edges, and where* $\ell_0 \in \mathcal{L}$ *is the initial location.*

We are interested in the *time-abstract* semantics of timed automata in the following sense. A timed automaton $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{X}, \ell_0, E)$, defines a transition system on the state space $\mathcal{L} \times \mathbb{R}^{\mathcal{X}}_{\geq 0}$, with the initial state $(\ell_0, \mathbf{0})$. There is a transition $(\ell, v) \xrightarrow{\sigma} (\ell', v')$ if, and only if there is $d \geq 0$ and an edge $(\ell, g, \sigma, R, \ell')$ such that $v + d \models g$ and $(v + d)[R \leftarrow 0] = v'$. Although timed automata define infinite transition systems, it is well-known that any timed automaton $\mathcal{A}$ is bisimilar to a computable flat system $\mathcal{T}(\mathcal{A})$, whose size can be exponentially larger than that of $\mathcal{A}$ [2]. See Figure 3 for a timed automaton bisimilar to a large flat system.

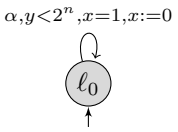

Fig. 3. A timed automaton with one location and two clocks x, y, modelling a "counter" ranging from 0 to 2^n that can only be incremented. At any state (ℓ_0, v) with $v(x) = 0$, $v(y)$ encodes the value of the counter. Taking the self-loop increments the counter. Any run stops after at most 2^n increments.

3 Succinctness

We compare hierarchical systems, products of flat systems, and timed automata with respect to succinctness of models. Our results show that these classes are pairwise incomparable in terms of succinctness: inside each class, there are infinite families of models which are exponentially more succinct than any bisimilar family of models in another class.

3.1 Hierarchical Systems *vs.* Products of Flat Systems

We show that hierarchical systems can be exponentially more succinct than products of flat systems: it is easy to define long finite chains with the former, as in Fig. 2, although this is not possible with the latter.

Theorem 1. *Hierarchical systems can be exponentially more succinct than products of flat systems.*

The other direction, in the next theorem, is more difficult.

Theorem 2. *Products of flat systems can be exponentially more succinct than hierarchical systems.*

This theorem also establishes a non-trivial property of hierarchical systems, giving insight into the differences with the other classes. It shows that any hierarchical graph of size n defining an exponentially large graph contains necessarily two

states that are bisimilar up-to $\Omega(n)$ steps. The proof is based on the observation that this is not the case for products of flat systems.

We consider the system A_n described in Fig. 1. We first give simple properties of A_n, based on the Chinese Remainder Theorem:

Theorem 3 (Chinese Remainder Theorem). *Let $p_1, \dots, p_n$ denote pairwise coprime numbers. For any integers $a_1, \dots, a_n$, there exists a unique $m \in [0, p_1 p_2 \dots p_n - 1]$ such that $m \equiv a_i \mod p_i$ for all $1 \leq i \leq n$.*

Let $p_1, \dots, p_n$ denote the first n prime numbers, and consider A_n given as the product of components $F_1, \dots, F_n$. Observe that A_n has $p_1 p_2 \cdots p_n$ states. By the Chinese Remainder Theorem, all bisimulation classes of A_n are singletons. In fact, consider a state s reached from the initial state by reading α^m. While executing the word α^{p_n} from s, measuring the minimal distance to some state that enables β_i, we can deduce m modulo p_i for each $1 \leq i \leq n$, and this uniquely determines m modulo $p_1 \cdots p_n$. This is formalized in the following lemma. In the rest of the section, we refer to states of A_n by natural numbers from 0 to $p_1 p_2 \cdots p_n - 1$.

Lemma 1. *For any pair of states $0 \leq c < c' < p_1 p_2 \cdots p_n$ of A_n, $c \not\sim_{p_n} c'$. Moreover, $c \sim_{p_n} c'$ if, and only if $c \sim c'$.*

Let us first explain the idea of the proof of Theorem 2. Any (sufficiently large) hierarchical graph of size polynomial in n that is bisimilar to A_n contains two occurrences of a nonterminal since A_n contains an exponential number of states. Similarly, some contact states of a same nonterminal appear several times. We will show moreover that for any hierarchical graph, there is a bisimilar one whose size is polynomially bounded, with the property that for some nonterminal B, the same contact state of two different copies of $\mathsf{eval}(B)$ belong to inner paths of $\mathsf{eval}(B)$ that are bisimilar up-to p_n steps. Thus, both occurrences of the contact state must be bisimilar to the same state of A_n by Lemma 1. However, we also show that these are reachable from the initial states in less than $p_1 p_2 \cdots p_n$ steps, which leads to a contradiction. We now give a formal proof following these ideas.

The size of A_n can be seen to be polynomially bounded in n from below and above since $p_n \sim n \log(n)$ for large n by the Prime Number Theorem. Assume there are hierarchical systems H_n such that $\mathcal{T}(H_n) \sim \mathcal{T}(A_n)$ for all $n \geq 0$, such that $|H_n| \leq f(n)$ for some polynomial f. We first show, in the following lemma, that each H_n can be assumed to satisfy the following Property ($\star$): for all nonterminals B, all traversing paths of $\mathsf{eval}(B)$ have length either 0 or at least $p_n + 2$.

Lemma 2. *For any family $(H_n)_{n \geq 0}$ of pointed hierarchical systems, there exist pointed hierarchical systems $(H'_n)_{n \geq 0}$ such that $|H'_n| \leq p(|H_n|)$ for some polynomial p, $\mathcal{T}(H_n) \sim \mathcal{T}(H'_n)$ and H'_n satisfies Property ($\star$), for all $n \geq 0$.*

The idea of the transformation is the following. We consider each production $B \to (\mathcal{A}, \tau, E)$, in the reverse topological order w.r.t. $\mathcal{E}_{H_n}$. Then, for all productions $C \to (\mathcal{A}', \tau', E')$ with $(C, B) \in \mathcal{E}_{H_n}$, we add to $\mathcal{A}'$ a copy of each traversing

path ρ of $\mathcal{A}$ of size less than $p_n + 2$, and remove all edges labeled by α leaving the first state of ρ. The construction is illustrated in Fig. 4.

Proof of Theorem 2. We assume that Property $(\star)$ holds, by Lemma 2. Consider any $n \geq 0$, such that $12f(n)^4 < p_1 p_2 \cdots p_n$, and let us write $H_n = (N, I, P)$. Consider a path π obtained in H_n by reading $\alpha^{p_1 \cdots p_n}$ from the initial state. For all nonterminals $B \in N$, with the unique production $B \to (\mathcal{A}, \tau, E)$, and $i \in \{1, \ldots, \mathsf{range}(\tau)\}$, let us mark by (B, i) in π all states that are equivalent, under relation $\equiv$, to the state $\tau(i)$ of $\mathcal{A}$. A single state in $\mathsf{eval}(H_n)$ can be marked by several pairs (B, i) since the equivalence $\equiv$ merges states. For example, if we were to apply this marking to the graph of Fig. 2, then the leftmost state would be marked by $(G_1, 1), (G_2, 1), \ldots, (G_n, 1)$, since at each production G_i, this state is merged with contact state 1 of the leftmost occurrence of nonterminal G_{i-1}. Note that at least one state among any consecutive $|H_n|$ states must be marked by some (B, i) in π. Otherwise π would not visit any contact states, and therefore would stay inside the same explicit graph, which has size less than $|H_n|$, and some state would appear twice since $p_1 p_2 \cdots p_n > f(n) \geq |H_n|$. Then, a same state of $\mathcal{T}(H_n)$ would be bisimilar to two distinct states of $\mathcal{T}(A_n)$, which contradicts Lemma 1. Since the number of pairs (B, i) is bounded by $|H_n|$ at least $m = \frac{|\pi|}{|H_n|^2}$ states of π are marked by some pair (B, i). Observe that $m = |\pi|/|H_n|^2 = \Omega(2^n/f(n)^2)$. Now, at least half the states marked by (B, i) mark the beginning of traversing paths of $\mathsf{eval}(B)$.

Among these states, assume that there are π_j and $\pi_{j'}$ for $0 \leq j < j' < p_1 \cdots p_n$, such that the traversing paths starting at these states have positive length (therefore, at least $p_n + 2$, thus contain at least p_n inner states). By assumption, these states are bisimilar to the states of A_n corresponding to the numbers j and j' respectively. Consider the inner paths $\pi_{j \ldots j+p_n}$ and $\pi_{j' \ldots j'+p_n}$ of $\mathsf{eval}(B)$. These paths belong to different instances of the production of B, so the visited states are pairwise disjoint. However, states π_j and $\pi_{j'}$, seen as states of $\mathsf{eval}(B)$ are bisimilar, since they correspond to the same contact state of $\mathsf{eval}(B)$. Since all α-labelled transitions from π_j lead to bisimilar states in $\mathsf{eval}(B)$, all internal paths starting at π_j are bisimilar. In particular, π_j and $\pi_{j'}$ are bisimilar up-to p_n steps, since they stay inside $\mathsf{eval}(B)$. So, each β_i is enabled in π_{j+k} iff it is enabled at $\pi_{j'+k}$, for all $1 \leq k \leq p_n$. But then π_j and $\pi_{j'}$ must be bisimilar to the same state of A_n by Lemma 1, and this is a contradiction.

Assume now that there is no more than one state π_j marked by (B, i) with a positive-length traversing path; so there are at least $m/2 - 1$ states corresponding to beginnings of traversing paths of length 0 (consisting of single states). Let $(\alpha_j)_{1 \leq j}$ denote the indices such that π_{α_j} is marked by (B, i) and is a traversing path of length 0. We argue that some state marked by a pair (B', i') with $(B, i) \neq (B', i')$, that is the beginning or the end of a traversing path of positive length must occur in $\pi_{\alpha_1 \ldots \alpha_1 + |H_n|}$. Consider the nonterminal C labeling the state $\mathsf{unfolding}(H_n, \pi_{\alpha_1})$. By definition, π_{α_1} is an inner state of C, so π_{α_1} is part of an inner path of $\mathsf{eval}(C)$. Since the structure defined in the production of C has size less than $|H_n|$, $\pi_{\alpha_1 \ldots \alpha_1 + |H_n|}$ must visit a state labeled by (B', i') that is the beginning or the end of an inner path of positive length: this is either a contact

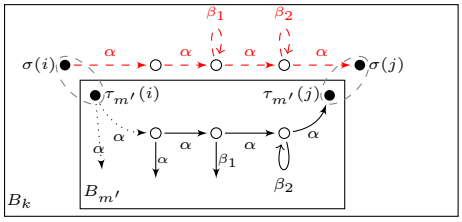

Fig. 4. The construction of Theorem 2 that removes a small traversing path created in a production $B_{m'} \to (\mathcal{A}_{m'}, \tau_{m'}, E_{m'})$. Here, $B_{m'}$ has an internal path of length 4 between $\tau_{m'}(i)$ and $\tau_{m'}(j)$, where internal states are represented by unfilled states. The contact states $\tau_{m'}(i)$ and $\tau_{m'}(j)$ are to be merged with the states $\sigma(i)$ and $\sigma(j)$ created in the production of B_k. The construction removes all edges of $\mathcal{A}_{m'}$ leaving $\tau_{m'}(i)$ (shown by dotted arrows). Then, the red dashed path ρ' is added instead from $\sigma(i)$ and $\sigma(j)$ in $\mathcal{A}_k$.

state of $\mathsf{eval}(C)$ (the end of the inner path containing π_{α_1}), or the beginning of an inner path inside $\mathsf{eval}(B')$, where $(C, B') \in \mathcal{E}_{H_n}$. In fact, if this subpath does not visit contact states of C and if it only contains inner paths of length 0 for other nonterminals B', then it stays inside the explicit graph defined in the production of C. This is again a contradiction with the bisimilarity with A_n since a state then must appear twice. This shows that every chunk of $|H_n|$ starting at some π_{α_i} contains a state marked by some other (B', i'), which is the beginning or the end of some traversing path of positive length of $\mathsf{eval}(B')$. Then, at least $\frac{m/2-1}{2|H_n|^2}$ states are marked by the same (B', i'), and are the beginning of traversing paths of positive length, and we can apply the previous case.

3.2 Timed Automata *vs.* Product of Flat Systems

Theorem 4. *Timed automata can be exponentially more succinct than products of flat systems.*

Proof. The proof immediately follows from Theorem 1 and the fact that the timed automaton of Fig. 3 is time-abstract bisimilar to the system G_n. □

We now show that products of flat systems can be more succinct than timed automata. This result requires new techniques since the nature of state-space explosion of timed automata is different; it is due to the complex relation between its clock values, rather than to its structure. To show this result, we use the well-known notion of *zones*, which are convex sets of the state space with integer corners. We only need the fact that zones are closed under basic operations such as time predecessors and intersection. We refer to [4] for definitions and properties of zones.

The main idea behind the proof is the following: a state in the transition system defined by a timed automaton can have an exponential number of a priori pairwise non-bisimilar successors but we show that the pairwise non-bisimilarity

of an exponential number of successors of a state cannot be detected by looking only one step further in the transition system. This important property is established using geometric properties of regions, and it is inherent to transition systems defined by timed automata. We show, on the other hand, that such a system can be defined by a small product of automata (system A'_n defined below), which yields the following theorem.

Theorem 5. *Products of flat systems can be exponentially more succinct than timed automata.*

For any $n \geq 1$, we define the finite transition system $\mathcal{T}_n$ on the set of states $S_n = \{(c_1, \ldots, c_n) \mid \forall 1 \leq i \leq n, 0 \leq c_i < p_i\}$, where p_i is the $i+2$-th prime number (so that we have $p_i \geq 5$, see below). From any state $(c_1, \ldots, c_n) \in S_n$ and any vector $(b_1, \ldots, b_n) \in \{1, 2\}^n$, there is a transition $(c_1, \ldots, c_n) \xrightarrow{\alpha} (c_1 + b_1 \mod p_1, \ldots, c_n + b_n \mod p_n)$. Moreover, we have a self-loop $(c_1, \ldots, c_n) \xrightarrow{\beta_i} (c_1, \ldots, c_n)$ whenever $c_i \equiv 0 \mod p_i$. $\mathcal{T}_n$ can be defined by adapting the system A_n of Fig. 1, by adding an edge from x to $x + 2$ (modulo p_i) inside each component F_i. Let us call A'_n this product of flat systems. It is clear that A'_n has size $O(n^2 \log(n))$ since $p_n \sim n \log n$.

The following lemma shows that states of $\mathcal{T}_n$ cannot simulate each other.

Lemma 3. *For all states $\boldsymbol{c}, \boldsymbol{c'}$ of $\mathcal{T}_n$, there is no simulation R such that $\boldsymbol{c}\ R\ \boldsymbol{c'}$.*

Proof (of Thm. 5). We consider any timed automaton T_n bisimilar to $\mathcal{T}_n$ (thus, to A'_n). By definition, all states of $\mathcal{T}_n$ have 2^n transitions, all leading to pairwise non-bisimilar states. We show that such a branching is not possible in T_n unless T_n has exponential size.

We consider the state $\boldsymbol{c} = (p_1 - 1, \ldots, p_n - 1)$ of $\mathcal{T}_n$, which is reachable. Let (ℓ, v) be any state of T_n that is bisimilar to $\boldsymbol{c}$. In $\mathcal{T}_n$, $\boldsymbol{c}$ has 2^n α-successors. Moreover, for each $1 \leq i \leq n$, β_i is enabled in exactly half of these successor states. In fact, for any subset $P \subseteq \{\beta_1, \ldots, \beta_n\}$, there is a successor where the set of enabled transition labels is exactly $P \cup \{\alpha\}$. Let $E(\ell)$ denote the number of edges from ℓ. For each successor $\boldsymbol{c'}$ of $\boldsymbol{c}$, pick a transition from (ℓ, v) in T_n, leading to a state bisimilar to $\boldsymbol{c'}$. Then, at least $2^n/E(\ell)$ of these transitions are along some edge $e = (\ell, \phi, \alpha, R, \ell')$. This means that there exist $d_1, \ldots, d_m \geq 0$ with $m = \lfloor 2^n/E(\ell) \rfloor$ such that states $(\ell, v+d_i)$ satisfy the guard ϕ; and the states $(\ell', v'_i) = (\ell', (v + d_i)[R \leftarrow 0])$ are each bisimilar to a successor of $\boldsymbol{c}$. States (ℓ, v'_i) are therefore pairwise non-simulating, by Lemma 3. Let us note here that R cannot be empty, since otherwise $(\ell', v + d_i)$ can simulate $(\ell', v + d_j)$ whenever $d_i \leq d_j$, which contradicts Lemma 3. We are going to show that there must be $\Omega(2^n)$ edges leaving ℓ'.

Valuations $v + d_i$ belong to a line of direction $\mathbf{1}$, that contains v. So the projections $v'_i = (v + d_i)[R \leftarrow 0]$ also belong to a line $\mathcal{D}$. Consider the set $g_1, \ldots, g_m$ of guards of the edges leaving ℓ'. Such a transition can be taken from v'_i if, and only if $v'_i + d \in g_j$ for some delay $d \geq 0$. This condition is equivalent to $v'_i \in \bigwedge_{x \in R}(x = 0) \wedge \mathsf{Pre}(g_i)$, where Pre gives the set of *time-predecessors* of g_i, *i.e.* $\mathsf{Pre}(g_i) = \{v \mid \exists d \geq 0, v + d \models g_i\}$. It is well-known that the right hand side of

the above expression can be expressed by a guard [4]. Therefore, for simplicity, but without loss of generality, let us replace g_i by the right hand side of the above. Thus, we have now a line $\mathcal{D}$ that contains all valuations v'_i, and convex sets defined by the guards. The intersection of each guard with $\mathcal{D}$ is a segment. From now on, we are only interested in valuations and segments that lie in $\mathcal{D}$. Each segment along $\mathcal{D}$ thus can be seen as an interval.

Now, we will show that only a small number of bisimulation classes can be distinguished inside $\mathcal{D}$, looking only at the immediate enablement of m guards. For a set of real intervals $\mathcal{I} = \{I_1, \ldots, I_n\}$, we denote by $\chi_{\mathcal{I}}$ the equivalence relation among real numbers given by $(x, y) \in \chi_{\mathcal{I}}$ if, and only if $x \in I \Leftrightarrow y \in I$, for all $I \in \mathcal{I}$. When $\mathcal{I}$ is finite, this relation is finite too. For instance, if $\mathcal{I} = \{[a, b]\}$, then $\chi_{\mathcal{I}}$ has index 2. We denote by $|\chi_{\mathcal{I}}|$ the index of $\chi_{\mathcal{I}}$.

Lemma 4. *Let $\mathcal{I}$ be a finite set of real intervals, and let J be a real interval. Then $|\chi_{\mathcal{I}\cup\{J\}}| \leq |\chi_{\mathcal{I}}| + 2$.*

Now, using m guards, one can only define $2m$ subsets of $\mathcal{D}$ which are pairwise distinguished with respect to the satisfaction of all guards g_i. By the previous lemma, there are at most $2m$ equivalence classes defined by $\chi_{\{g_1,\ldots,g_{E(\ell')}\}}$. On the other hand, any pair of states v'_i and v'_j can be distinguished by the satisfaction of some guard g_k, since this is the case for the 2^n successors of $\boldsymbol{c} = (p_1-1, \ldots, p_n-1)$ inside $\mathcal{T}_n$. It follows that $2m \geq 2^n/E(\ell)$. Therefore, $|T_n| \geq m = \Omega(2^n)$. □

3.3 Timed Automata *vs.* Hierarchical Systems

Theorem 6. *Timed automata can be exponentially more succinct than hierarchical systems, and vice versa.*

The proof of the first direction is similar to that of Theorem 2: we give a timed automaton that describes a system similar to A_n. The other direction uses the techniques of Theorem 5.

4 Complexity of Preorder Checking

4.1 Hardness of Simulation

The main result of this section is that deciding simulation between two hierarchical systems is EXPTIME-complete. Our proof is based on a simple reduction from countdown games [10]. Our reduction is quite generic, and it can be applied to any class with a set of simple properties (discussed at the end of the section). As an example, we apply the reduction to timed automata. Note the EXPTIME-hardness of checking simulation for timed automata was already proved in [14] by a reduction based on Turing machines; we obtain here a simpler proof.

Theorem 7. *Checking simulation between two hierarchical systems (resp. two timed automata) is* EXPTIME*-complete.*

Our reduction is based on *countdown games* [10], defined as follows. A countdown game $\mathcal{C}$ is played on a weighted graph (S, T), whose edges are labeled with positive integer weights encoded in binary. A *move* of the game from configuration $(s, c) \in S \times \mathbb{N}$ is determined jointly by both players, as follows. First, Eve chooses a number $d \leq c$ such that $(s, d, s') \in T$ for some state s'. Then Adam chooses a state $s' \in S$ such that $(s, d, s') \in T$. The resulting configuration is $(s', c-d)$. The game stops when Eve has no available moves: configuration (s, c) is *winning* for Eve if $c = 0$. Given a countdown game, one can build an equivalent turn-based graph game of exponential size with a reachability objective. We note that given a countdown game and an initial configuration, the existence of a winning strategy for Eve is EXPTIME -complete [10].

We first reduce the problem of determining the winner in countdown games to the simulation problem on finite automata, that may have exponential size. We then show how these automata can be described in polynomial size by hierarchical systems and timed automata. This proves the EXPTIME-hardness (thus, completeness) of the simulation problem on these classes.

Consider a countdown game $\mathcal{C} = (S, T)$ with initial state $q \in S$ and initial value c. Let Σ denote the set of constants used in $\mathcal{C}$. We define two finite automata on the alphabet $\Gamma = \Sigma \cup \{e, \alpha, \beta\}$. The first one, called $\text{Counter}_{\mathcal{C}}(c)$, is a directed path of length c with some additional states, defined in Fig. 5. The bottom left state is the initial state. Intuitively, this is used to count down from c when simulating the countdown game.

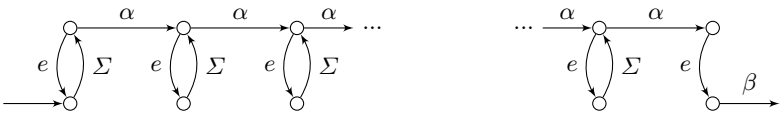

Fig. 5. System $\text{Counter}_{\mathcal{C}}(c)$.

The second automaton is called $\text{Control}_{\mathcal{C}}$, and has the same structure as the game $\mathcal{C} = (S, T)$, except that each transition labeled by $k \in \Sigma$ is replaced by a module $\text{Chain}_{\mathcal{C}}(k)$, which is roughly a directed path of length $k + 1$ labeled by α^k. In addition, in every state, an edge leads to a sink state by any symbol of $\Gamma \setminus \{\alpha\}$ from all but the last state, and by any symbol in $\Gamma \setminus \{e\}$ from the last state. Sink states have self-loops on all symbols. The module is given in Fig. 6.

Now, automaton $\text{Control}_{\mathcal{C}}$ is defined by replacing each transition labeled by k in the game $\mathcal{C}$, with an instance of module $\text{Chain}_{\mathcal{C}}(k)$, as shown in Fig. 7. Moreover, from each state s_i, there is an edge going to a sink state, labeled by all labels in $\Gamma \setminus (\Sigma(s_i) \cup \{\beta\})$, where $\Sigma(s_i)$ denotes the set of labels of the edges leaving s_i in game $\mathcal{C}$. This ensures that a path in $\text{Control}_{\mathcal{C}}$ encodes a correct simulation of the game. The initial state of $\text{Control}_{\mathcal{C}}$ is the initial state of $\mathcal{C}$.

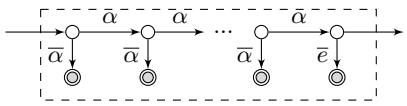

Fig. 6. Module $\mathrm{Chain}_{\mathcal{C}}(k)$. Here, $\overline{x}$ denotes the complement of the set $\{x\}$. All gray states at the bottom in the figure are sink states with (omitted) self-loops on all symbols.

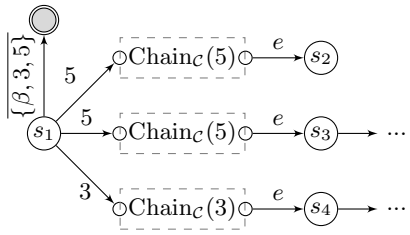

Fig. 7. A part of $\mathrm{Control}_{\mathcal{C}}$ for a countdown game $\mathcal{C}$ with states s_1, s_2, s_3, s_4 and edges $(s_1, 5, s_2), (s_1, 5, s_3), (s_1, 3, s_4)$.

Proposition 1. *For any countdown game $\mathcal{C} = (S, T)$ with initial state s_1 and initial value c, $\mathrm{Counter}_{\mathcal{C}}(c) \sqsubseteq \mathrm{Control}_{\mathcal{C}}$ if, and only if Eve does not have a winning strategy in $\mathcal{C}$ from configuration (s_1, c).*

We now explain how this reduction can be applied to hierarchical systems and timed automata, in polynomial time. For hierarchical systems, in order to succinctly represent $\mathrm{Counter}_{\mathcal{C}}(c)$ and modules $\mathrm{Chain}_{\mathcal{C}}(k)$, we use the trick of Fig. 2. For instance, in order to define $\mathrm{Chain}_{\mathcal{C}}(k)$, one can generate all systems $G_1, G_2, \ldots, G_{\lceil \log(k) \rceil}$ and combine these according to the binary representation of k. For timed automata, a pair of clocks can be simply used to count upto k, as in Fig. 3. Thus, modules $\mathrm{Counter}_{\mathcal{C}}(c)$ and $\mathrm{Chain}_{\mathcal{C}}(k)$ can be defined in polynomial space in these classes, which yields a polynomial-time reduction.

4.2 Simulation and Bisimulation with a Flat System

We show that checking whether a hierarchical graph is simulated by a finite automaton is PSPACE-complete. The PSPACE-membership follows from the fact that simulation by a finite automaton can reduced to μ-calculus model-checking (see e.g. [12]), which is in turn in PSPACE [6]. The corresponding lower bound can in fact be deduced from results from [11] and [12] by using lengthy definitions, however, we decided to give a direct reduction from quantified Boolean satisfiability problem.

Theorem 8. *Checking whether a flat system simulates a hierarchical system is* PSPACE*-complete.*

Second, we show that the problems of checking bisimilarity between a timed automaton and a flat system, and between a hierarchical system and a flat system are PSPACE-complete. In fact, one can reduce bisimilarity with a flat system to model checking CTL's fragment EF (where formulas are represented as DAGs) in polynomial time [12]. This yields a polynomial space algorithm for this problem since EF model-checking is easily seen to be in PSPACE for products of flat systems, timed automata and hierarchical systems since reachability for all these systems is in PSPACE.

Theorem 9. *Checking bisimilarity between a timed automaton (resp. hierarchical system, product of flat systems) and a flat system is* PSPACE*-complete.*

The PSPACE-hardness for product of finite automata was already proved in [5]. For timed automata, it follows from PSPACE-hardness of control state reachability that checking *any* relation between time-abstract language equivalence and time-abstract bisimulation between a timed automaton and a finite automaton is PSPACE-hard. For hierarchical systems, we observe that the reduction of [12] that shows the PSPACE-hardness of checking bisimulation between a pushdown automaton and a finite automaton can be adapted to hierarchical systems.

4.3 Language Inclusion and Universality

Given any timed automaton $\mathcal{A}$, one can effectively construct an exponential-size finite automaton, called the *region automaton* that is time-abstract bisimilar to $\mathcal{A}$ [2]. Then, using region automata, one can decide the inclusion between the untimed languages of two timed automata in exponential space. The exact complexity of these problems had not been characterized, and the problem was wrongly cited in the literature as being PSPACE in [1, 19]. In this section, we prove that untimed language universality and inclusion are actually EXPSPACE-complete. The result holds already for two clocks. For one clock, the problem is PSPACE-complete. Language inclusion can also be decided in exponential space for hierarchical systems, since the system generated has at most exponential size. We adapt the proof to hierarchical systems, and obtain the same complexity results.

Theorem 10. *Checking untimed language universality is* EXPSPACE*-complete for timed automata with two clocks, and* PSPACE*-complete with one clock. Language universality is* EXPSPACE*-complete for hierarchical systems.*

To prove this, we consider the acceptance problem on exponential-space Turing machines, and show how to compute timed automata (resp. hierarchical systems) that accept all words but those encoding correct accepting executions.

5 Conclusion

In this paper, we compared products of automata, timed automata and hierarchical systems, which are used to succinctly describe finite-state systems. We showed that each of them contains models that are exponentially more succinct than the others, formalizing the intuition that the nature of the state space explosion is different in each formalism. Several variants of these systems were not considered in this paper. For instance, silent transitions improve succinctness in general: the main argument in the proof of Theorem 5 does not hold for timed automata with silent transitions. One could also study different synchronization semantics for products of automata. We also studied the computational complexity of several preorder and equivalence relations. The complexity of bisimilarity between hierarchical systems remains open.

References

1. Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, NY, USA, 2007.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
3. Rajeev Alur and Mihalis Yannakakis. Model checking of hierarchical state machines. *ACM Trans. Program. Lang. Syst.*, 23(3):273–303, 2001.
4. Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, p. 87–124, 2003.
5. Laura Bozzelli, Axel Legay, and Sophie Pinchinat. Hardness of preorder checking for basic formalisms. In *LPAR (Dakar)*, p. 119–135, 2010.
6. Stefan Göller and Markus Lohrey. Fixpoint logics over hierarchical structures. *Theory Comput. Syst.*, 48(1):93–131, 2011.
7. David Harel, Orna Kupferman, and Moshe Y. Vardi. On the complexity of verifying concurrent transition systems. *Inf. Comput.*, 173:143–161, March 2002.
8. David Janin and Igor Walukiewicz. On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic. In *Proc. of CONCUR*, LNCS 1119, p. 263–277. Springer, 1996.
9. Lalita Jategaonkar and Albert R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154(1):107–143, 1996.
10. Marcin Jurdziński, François Laroussinie, and Jeremy Sproston. Model checking probabilistic timed automata with one or two clocks. In TACAS'07, LNCS 4424, p. 170–184. Springer, March 2007.
11. Antonín Kučera and Richard Mayr. Why is simulation harder than bisimulation? In *CONCUR*, LNCS 2421, p. 594–610. Springer, 2002.
12. Antonín Kučera and Richard Mayr. On the complexity of checking semantic equivalences between pushdown processes and finite-state processes. *Information and Computation*, 208(7):772–796, 2010.
13. François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking timed automata with one or two clocks. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, LNCS 3170, p. 387–401. Springer, August 2004.
14. François Laroussinie and Philippe Schnoebelen. The state explosion problem from trace to bisimulation equivalence. In FOSSACS'07, p. 192–207. Springer, 2000.
15. Thomas Lengauer and Egon Wanke. Efficient solution of connectivity problems on hierarchically defined graphs. *SIAM J. Comput.*, 17(6):1063–1080, 1988.
16. Markus Lohrey. Model-checking hierarchical structures. *J. Comput. Syst. Sci.*, 78(2):461–490, 2012.
17. Faron Moller and Alexander Moshe Rabinovich. Counting on CTL*: on the expressive power of monadic path logic. *Inf. Comput.*, 184(1):147–159, 2003.
18. Anca Muscholl and Igor Walukiewicz. A lower bound on web services composition. *Logical Methods in Computer Science*, 4(2), 2008.
19. Jiri Srba. Comparing the expressiveness of timed automata and timed extensions of Petri nets. In FORMATS'08, LNCS 5215, p. 15–32. Springer, 2008.
20. Johan van Benthem. *Modal Correspondence Theory*. PhD thesis, University of Amsterdam, 1976.
21. Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR*, LNCS 458, p. 278–297. Springer, 1990.

22. Rob J. van Glabbeek. The linear time - branching time spectrum ii. In *CONCUR*, LNCS 715, p. 66–81. Springer, 1993.

A Proofs of Section 3

Hierarchical systems vs. Product of flat systems A product of flat systems of polynomial size cannot describe the system G_n given in Fig. 2. The proof is an application of the pumping lemma: since each component of the product has polynomial size, if there is a path of length 2^n in the product, there must be one in one of the components. This however implies that a longer path is also possible in the product.

Theorem 1. *Hierarchical systems can be exponentially more succinct than products of flat systems.*

Proof. For any $n \geq 1$, we define a system H_n describing a chain with 2^{n-1} transitions, where all transitions are labeled by α. H_n can be defined by a hierarchical system of size polynomial in n. In fact, one can define a nonterminal G_m, inductively, by joining two copies of G_{m-1}, for all $1 \leq m \leq n$, and G_1 is defined as an explicit system. These can be defined similarly to the example of Fig. 2. One only needs to use relation R_α to label transitions. Thus, the system $\mathsf{eval}(G_n)$ has size linear in n. The initial state is the leftmost state.

Assume that there is a family of synchronized product of finite automata $(A_n)_{n\geq 0}$ bisimilar to $(H_n)_{n\geq 0}$, where A_n has size polynomial in n. Consider n such that $|A_n| < 2^{n-1}$ and let us write A_n for the product $A_n = F_1 \parallel \ldots \parallel F_k$. Consider the word $w = \alpha^{2^{n-1}}$. Notice that after reading w from the initial state of H_n, action α is not enabled. Consider any path from the initial state of A_n obtained by reading w. One of the components F_i has a path reading w from its initial state. This property follows from the definition of the synchronous product: any component in which the action α is enabled has to take a corresponding transitions at each step where α is read, and if α is not enabled in some component it will never be enabled again . But since $|F_i| < 2^{n-1}$, this path contains a cycle made of α-labeled transitions. Then, for instance $\alpha^{2^{n-1}+1}$ can be read in A_n, but this is not possible in H_n, which is a contradiction.

□

Lemma 2. *For any family $(H_n)_{n\geq 0}$ of pointed hierarchical systems, there exist pointed hierarchical systems $(H'_n)_{n\geq 0}$ such that $|H'_n| \leq p(|H_n|)$ for some polynomial p, $\mathcal{T}(H_n) \sim \mathcal{T}(H'_n)$ and H'_n satisfies Property $(\star)$, for all $n \geq 0$.*

Proof. An important observation that we will use throughout the proof is that since A_n is deterministic, for any word $w = \alpha^m$, if there are several paths labeled by w in $\mathsf{eval}(H_n)$ starting from the initial state, then the target states of all these paths are bisimilar. We use this observation for instance to discard some transitions with the same label that leave the same state. Removing such transitions always yields a bisimilar system, as long as we leave at least one

outgoing transition per label. Observe also that in H_n, there are no self-loops labeled by α on states, reachable from the initial state, since this would contradict bisimilarity with A_n, by Chinese remainder theorem. For any $n \geq 0$, we construct a new hierarchical system H'_n from H_n, which satisfies the property that for all nonterminals B, all traversing paths in $\mathsf{eval}(B)$ have length 0 or at least $p_n + 2$. We topologically sort the nonterminals according to the preorder $\mathcal{E}_{H_n}$; let us write $B_1 B_2 \dots B_m$, where each B_i only contains references among $B_{i+1} \dots B_m$. We treat each nonterminal $B_{m'}$ in the reverse order, from m down to 1. When we treat $B_{m'}$, we assume that all systems $\mathsf{eval}(B_{m''})$ with $m' < m''$ only have traversing paths of length 0 or at least $p_n + 2$.

This is trivial for B_m, which does not have references. For $1 \leq m' < m$, let us write $B_{m'} \to (\mathcal{A}_{m'}, \tau_{m'}, E_{m'})$, and for all $1 \leq i \neq j \leq \mathsf{range}(\tau_{m'})$, consider ρ the shortest directed path from $\tau_{m'}(i)$ to $\tau_{m'}(j)$ labeled entirely by α's. By induction, if $|\rho| < p_n$, then ρ only visits states defined in $\mathcal{A}_{m'}$ and it does not visit any contact state of references in $E_{m'}$; since if it entered a contact state of a reference in $E_{m'}$, then it would come out of this system only after visiting at least $p_n + 2$ states. Assume $0 < |\rho| < p_n$. We define ρ' as a fresh copy of ρ, where we also add a self-loop labeled by β_j at each state ρ'_i, whenever there is a transition labeled by β_j leaving the state ρ_i. We then remove all transitions leaving the state $\tau_{m'}(i)$ in $\mathcal{A}_{m'}$. Notice that system $\mathsf{eval}(B_{m'})$ may still have transitions leaving $\tau_{m'}(i)$, since these can be introduced in references. However, by induction hypothesis, such transitions correspond to traversing paths of positive length. Next, we consider all nonterminals B_k with $k < m'$, in which $B_{m'}$ appears (*i.e.* $(B_k, B_{m'}) \in \mathcal{E}_D$). Let us write $B_k \to (\mathcal{A}_k, \tau_k, E_k)$. For any reference $(B_{m'}, \sigma)$ in E_k, $\sigma(i)$ and $\sigma(j)$ denotes states of $\mathcal{A}_k$ that are merged with the states $\tau(i)$ and $\tau(j)$ of $\mathsf{eval}(B_{m'})$. We add a fresh copy of ρ' in $\mathcal{A}_k$, starting at $\sigma(i)$ and ending in $\sigma(j)$. As observed above, by the fact that all α-successors (resp. β_i-successors) at each state lead to bisimilar states, this operation preserves the initial system up-to bisimilarity. The construction is illustrated in Fig. 4.

Thus, when defining H'_n, we add $O(p_n)$ states inside each production B', for all pairs (B, i, j). So we have $|H'_n| = O(|H_n|^3 p_n) = O(f(|A_n|)^4)$. Since $|H'_n| = \Omega(2^n)$ by the above proof, $|H_n|$ cannot be bounded by a polynomial. □

Timed Automata vs. Product of flat systems

Lemma 3. *For all states $\boldsymbol{c}, \boldsymbol{c'}$ of $\mathcal{T}_n$, there is no simulation R such that $\boldsymbol{c}\ R\ \boldsymbol{c'}$.*

Proof. We distinguish several cases. Let us write $\boldsymbol{c} = (c_1, \dots, c_n)$ and $\boldsymbol{c'} = (c'_1, \dots, c'_n)$.

1. First assume that $c_i \geq c'_i + 2$ for some i. Then we consider the shortest path labeled by α's, ending with a state where β_i is enabled. Formally, we consider any path $s_0 s_1 \dots s_m$ from $s_0 = \boldsymbol{c}$, such that the i-th component of s_i is increased by 2 at each step, except possibly for the last one, and the i-th component of s_m is 0. Since $c_i \geq c'_i + 2$, any path $s'_0 s'_1 \dots s'_m$ from $s'_0 = \boldsymbol{c'}$, satisfies $(s_{m-1})_i \geq (s'_{m-1})_i + 2$. Thus, β_i is not enabled in s'_m.

2. Assume that $c_i = c'_i + 1$ for some i.
   - If $p_i - c_i$ is even, then we consider $\frac{p_i - c_i}{2}$ α-transitions from $\boldsymbol{c}$ along which the i-th component is increased by 2 at each step. This proves that $\boldsymbol{c'}$ cannot simulate $\boldsymbol{c}$ since immediately after this path, β_i is enabled at $\boldsymbol{c}$ but not from the states along any path of same length from $\boldsymbol{c'}$.
   - If $p_i - c_i$ is odd, we consider $\lceil \frac{p_i - c_i}{2} \rceil$ α-transitions from $\boldsymbol{c}$, along which the i-th component is increased by 2. This yields to a state s_0 where the i-th component is 1. Let s'_0 denote any state reached from $\boldsymbol{c'}$ while simulating this path. If $(s'_0)_i = 0$, then we extend s_0 by $\frac{p_i - 1}{2}$ α-transitions along which the i-th component is increased by 2 at each step. Then β_i is enabled at the last state, but this cannot be the case for paths from s'_0, of same length. If $(s'_0)_i = p_i - 1$, then consider only one α-transition from s_0, leading to s_1 with $(s_1)_i - (s_0)_i = 2$. For any state s'_1 that is a successor of s'_0, we have $(s'_1)_i \in \{0, 1\}$ and $(s_1)_i \geq (s'_1)_i + 2$. So we are back at the previous case.
3. Assume now $c_i < c'_i$ for some $i \in \{1, \ldots, n\}$. Let us first distinguish two special cases. If $c_i = 0$ and $c'_i = p_i - 1$, then we increment c_i by 2, which leads us to the previous case ($c_i > c'_i$). If $c_i = 0$ and $c'_i = p_i - 2$, then we repeat two such transitions, and we get $0 \leq c'_i < c_i = 4$. Since $p_i \geq 5$, we are back at the previous case. Now, in the general case, that is when $c_i \geq 1$, we consider a sequence of α-transitions from $\boldsymbol{c}$, where all components are increased by 1, until a β_i-transition is possible for the first time. Along this path, from $\boldsymbol{c'}$, any path will necessarily reach a state s' with $s'_i \in \{0, 1\}$. Since in the rest of the path from $\boldsymbol{c}$, the i-th component is always greater than 2, we are back at the previous case.

□

Lemma 4. *Let $\mathcal{I}$ be a finite set of real intervals, and let J be a real interval. Then $|\chi_{\mathcal{I} \cup \{J\}}| \leq |\chi_{\mathcal{I}}| + 2$.*

Proof. One can show, by induction, that the equivalence classes of $\chi_{\mathcal{I}}$ are unions of (disjoint) intervals. One can write $I_1 \cup \ldots \cup I_n = \mathbb{R}_{\geq 0}$, where each I_i is a disjoint interval, such that each equivalence class of $\chi_{\mathcal{I}}$ is the union of a subset of I_i's. When we refine this relation by adding $\mathcal{J}$, there are three possibilities for each of the intervals I_i: either $I_i \subseteq J$, or $I_i \subseteq J^c$, or $I_i \not\subseteq J$ and $I_i \cap J \neq \varnothing$. In the first two cases, I_i is entirely included in an equivalence class of $\chi_{\mathcal{I} \cup \{J\}}$, since the elements of I_i cannot be distinguished by J. In the latter case, I_i is split into two $I_i \cap J$ and $I_i \cap J^c$. But since I_i's are disjoint, this case can happen for at most two intervals I_i and I_j (containing the endpoints of J). Then, the number of equivalence classes increases by 2. □

Timed Automata vs. Hierarchical systems

Theorem 6. *Timed automata can be exponentially more succinct than hierarchical systems, and vice versa.*

We prove the two directions separately.

Timed automata can be more succinct than hierarchical graphs.

Proof. We construct a timed automaton that describes a system obtained by a slight modification of A_n of Fig. 1. The proof is then similar to that of Theorem 2. Given prime numbers $p_1, \ldots, p_n$, we define a timed automaton T_n with clocks $x_1, \ldots, x_n$ and u. The clocks will have integer values immediately after each transition. Runs of T_n will count from 0 to $p_1 \ldots p_n$, and each clock x_i will record the value modulo p_i. A single incrementation of the value will be done in n α-steps. At step i of an incrementation, T_n is able to take a self-loop labeled by β_i, if the current value is 0 modulo p_i. The timed automaton T_n is given in the following figure. The length of the path since the initial state can be

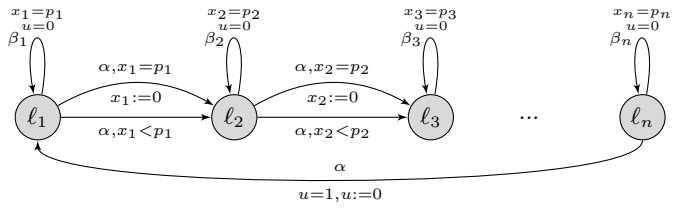

Fig. 8. Timed automaton T_n of Theorem 6

deduced looking $p_n n{+}2$ transitions ahead, by Chinese remainder theorem. Timed automaton T_n has size $O(p_1 + \ldots + p_n)$. Using the same trick as in Theorem 2, any family of hierarchical systems bisimilar to the family of timed automata $(T_n)_{n \geq 1}$ can be assumed to have all traversing paths of length either 0 or at least $p_n n + 2$. Then, by the same arguments as in Theorem 2, we prove that any such family of hierarchical systems must have exponential size. □

Hierarchical graphs can be more succinct than timed automata.

Proof. For any $n \geq 1$, we define system G'_n whose transitions are labeled in the alphabet $\{\alpha, 0, 1\} \cup \{\beta_1, \ldots, \beta_n\}$, as follows. G'_n contains a binary tree of height n, whose transitions are directed bottom-up. Each transition leaving a left child of the parent state is labeled by 0, and others by 1. We create the state s_{β_i} for each label β_i. Each state s_{β_i} has a self-loop labeled by β_i. From each leaf f of the tree, we add an transition labeled by α to state s_{β_i} if, and only if, the i-th transition in the unique path from f to the root is labeled by 1. Otherwise, we add a transition labeled by α from f to a blocking state $\perp$. We add a transition labeled by α from the root to each leaf. Notice that all leaves are pairwise non-simulating since they are distinguished by the paths leading to the root.

Consider now a timed automaton T_n that is time-abstract bisimilar to G'_n. Let E denote the number of edges in T_n. From the initial state (ℓ_0, ν_0), there are 2^n successors that are bisimilar to the leaves of G'_n. Then, there is an edge

$(\ell_0, \phi, \alpha, R, \ell)$ and $d_1, \ldots, d_{\lfloor 2^n/E \rfloor} \geq 0$ such that $\nu_0 + d_i \models \phi$ and if we write $\nu'_i = (\nu_0 + d_i)[R \leftarrow 0]$, then each (ℓ, ν'_i) is bisimilar to a distinct leaf of G'_n. Here, R cannot be empty since otherwise (ℓ, ν'_i) can simulate (ℓ, ν'_j) whenever $d_i \leq d_j$. Thus, all ν'_i lie in a line $\mathcal{D}$ (the line directed by vector $\mathbf{1}$ going through ν_0, projected by resetting clocks R) and their time-successors do not intersect. Consider any pair of consecutive edges from ℓ with labels α and β_i, say $e = (\ell, g_\alpha, \alpha, R_\alpha, \ell')$ and $e' = (\ell', g_i, \beta_i, R_i, \ell'')$. The set of valuations at location ℓ that can follow these edges is given by the following formula:

$$G_{e,e'} = \mathsf{Pre}\big(g_\alpha \cap \mathsf{Unreset}_{R_\alpha}(\mathsf{Pre}(g_i))\big).$$

Here $\mathsf{Unreset}_{R_\alpha}(X)$ denotes the set of all states that end in X by resetting the clocks R_α. Then, a valuation ν'_i can fire the sequence $\alpha\beta_i$ following these edges, if, and only if $\nu'_i \in G_{e,e'} \cap \mathcal{D}$, which is a convex subset of $\mathcal{D}$, thus a segment. Now, each state (ℓ, ν'_i) has a different subset of sequences $\alpha\beta_i$ for $1 \leq i \leq n$, that it can fire. And we have at most $|E|^2$ sequences of edges labeled by $\alpha\beta_i$, for any i, leaving ℓ. By Lemma 4, we have $2E^2 \geq 2^n/E$. This implies that $E = 2^{\Omega(n)}$, thus $|T_n| = 2^{\Omega(n)}$.

We now explain how G'_n can be defined by a hierarchical system of polynomial size. The tree is constructed recursively top-down. We define nonterminals $A_1, \ldots, A_n$, where A_i has rank $i + 2$, each defining one level of the tree. Each production A_i creates the state s_{β_i}, and A_1 also creates states root and $\bot$. A_1 creates the two child states for root with corresponding edges labeled by 0 and 1 respectively. A_1 has two references of nonterminal A_2, defining the left and the right subtrees. The left subtree is defined by a copy of A_2 with its contact states merged with the triple $(\mathsf{root}, \bot, \bot)$, while the right subtree is defined similarly but by connecting the contact states to $(\mathsf{root}, \bot, s_{\beta_1})$. The idea behind the choice of the contact states is the following. For any nonterminal A_i for $1 \leq i \leq n-1$, the first contact state is always root, and the second one is $\bot$. For any $1 \leq j \leq i-1$, the contact state $j + 2$ is given s_{β_j}, if in the path from the current state to root, the transition of depth j is labeled by 1, and otherwise it is $\bot$. Then, each leaf state is created inside the nonterminal A_n, which has contact states $(\mathsf{root}, \bot, t_1, \ldots, t_n)$, for some $t_i \in \{s_{\beta_i}, \bot\}$ for each $1 \leq i \leq n$. Nonterminal A_n creates a transitions labeled by α from the leaf state to each t_i. □

B Proofs of Section 4

B.1 Hardness of Simulation

Proposition 1. *For any countdown game $\mathcal{C} = (S, T)$ with initial state s_1 and initial value c, $\mathrm{Counter}_{\mathcal{C}}(c) \sqsubseteq \mathrm{Control}_{\mathcal{C}}$ if, and only if Eve does not have a winning strategy in $\mathcal{C}$ from configuration (s_1, c).*

Proof. In the simulation game between $\mathrm{Counter}_{\mathcal{C}}(c)$ and $\mathrm{Control}_{\mathcal{C}}$, Eve will be the spoiler, thus choosing the successors in $\mathrm{Counter}_{\mathcal{C}}(c)$, while Adam will be the duplicator, choosing matching transitions in $\mathrm{Control}_{\mathcal{C}}$. Assume that Eve has a

winning strategy in $\mathcal{C}$. At a state (s_i, c), if the strategy is to pick $k \in \Sigma(s_i)$, then the spoiler will read the word $k\alpha^k e$ in $\text{Counter}_{\mathcal{C}}(c)$. Then, the duplicator has no choice but to pick a successor of q with label k, and by executing the same word, it will end in some s_j such that $(s_i, k, s_j) \in T$. Note that, if the spoiler does not pick a label inside $\Sigma(s_i)$, or if it reads a word kw where $\alpha^k e$ is not a prefix of w, then the replicator will be able to go to a sink state and win the simulation game, by construction of our modules. However, if Eve plays correctly the simulation game, then the simulation will arrive at some state s_i, and counter value 0 (which is the last state of $\text{Counter}_{\mathcal{C}}(c)$). At this point, $\text{Counter}_{\mathcal{C}}(c)$ is able to read β, but this is not possible at states s_i. Thus, simulation fails. On the other hand, if Eve does not have a winning strategy in $\mathcal{C}$, then spoiler will either not respect the above encoding of the transitions of the game (and the duplicator will win by moving to a sink state), or it will necessarily read β while $\text{Control}_{\mathcal{C}}$ is inside a module $\text{Chain}_{\mathcal{C}}(k)$. But in the latter case, $\text{Control}_{\mathcal{C}}$ is able to go to a sink state by reading β, so simulation holds. □

B.2 Simulation and Bisimulation with a Flat System

Theorem 8. *Checking whether a flat system simulates a hierarchical system is* PSPACE*-complete.*

Proof. The PSPACE algorithm is described in the core of the paper. To show hardness, we present a reduction from quantified satisfiability problem QSAT. For any fully quantified formula in prenex normal form $\phi = \forall x_1.\ \exists x_2.\ \ldots \exists x_n.\ C_1 \wedge \cdots \wedge C_k$, with $C_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$, we construct a hierarchical system $G(\phi)$ and an automaton $A(\phi)$ such that the automaton simulates the hierarchical system if, and only if the formula is valid.

We first define the reduction by a (large) flat system, then explain how this can be succinctly defined by hierarchical graph.

To any quantifier-free formula ϕ and valuation $v\colon \{x_1; \ldots; x_n\} \mapsto \{\mathsf{true}, \mathsf{false}\}$, we associate a system $G_v(\phi)$, on alphabet $\Sigma = \{\mathsf{true}, \mathsf{false}, \alpha, \alpha_1, \alpha_2, \alpha_3\}$ as follows. We have an initial state t_v, and a state $c_{v,i}$ for each clause C_i, for $1 \leq i \leq k$. There is a transition labeled by α from the initial state t_v of $G_v(\phi)$ to each $c_{v,i}$. We then have transitions for each literal, with different labels, that leads to either $\bot$ or $\top$ depending on the valuation of the literal: if $v(\ell_{i,j}) = \mathsf{true}$ then $c_{v,i} \xrightarrow{\alpha_j} \top$, and otherwise $c_{v,i} \xrightarrow{\alpha_j} \bot$.

We first show that t_v is simulated by the initial state s_0 of the automaton $A(\phi)$, described in Fig. 9, if and only if, v evaluates the formula ϕ to true. If s_0 simulates t_v, then for all $c_{v,i}$ there is a s_j such that s_j simulates $c_{v,i}$, as $s_j \xrightarrow{\alpha_j} \top$ and $\top$ does not simulate $\bot$, this means that $c_{v,i} \xrightarrow{\alpha_j} \top$ and therefore $v(\ell_{i,j}) = \mathsf{true}$. Hence $v(\phi) = \mathsf{true}$. If $v(\phi) = \mathsf{true}$, then for each clause C_i, a literal $\ell_{i,j}$ is true for v. We show that s_j simulates $c_{v,i}$. If $k \neq j$, then $s_j \xrightarrow{\alpha_k} \bot$ which simulates any state. In the other case, $c_{v,i} \xrightarrow{\alpha_j} \top$ which is simulated by any state. Hence t_v is simulated by s_0.

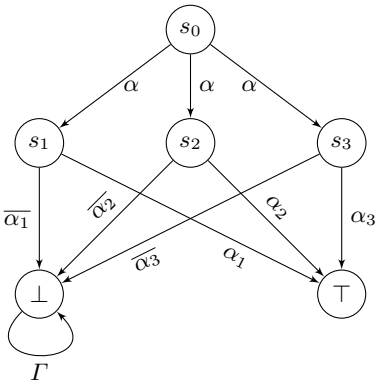

Fig. 9. Automaton $A(\phi)$, for a quantifier-free formula

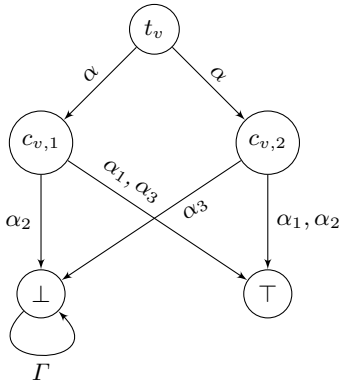

Fig. 10. Example of $G_v(\phi)$, for formula $\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$ and valuation $v : x_1 \mapsto \mathsf{true}; x_2 \mapsto \mathsf{false}; x_3 \mapsto \mathsf{false}$

Now if x_1 and x_2 are free variables in ϕ, and v is a partial valuation, we construct the system $G_v(\forall x_1.\ \exists x_2.\ \phi)$ as described in Fig. 11, and the corresponding automaton $A(\forall x_1.\ \exists x_2.\ \phi)$. We show that s_0 simulates t_v, if and only if, $\forall x_1.\ \exists x_2.\ \phi$ is valid under the valuation v. If s_0 simulates t_v, for all t_a with $a \in \{\mathsf{true}, \mathsf{false}\}$, there is a s_b such that s_b simulates t_a. As $t_a \xrightarrow{b} G_{v[x_1 \mapsto a; x_2 \mapsto b]}$, $A(\phi)$ simulates $G_{v[x_1 \mapsto a; x_2 \mapsto b]}$. Hence $\forall x_1.\ \exists x_2.\ \phi$ is made valid by v. If $\forall x_1.\ \exists x_2.\ \phi$ is made valid by v, then for t_a select s_b such that $v[x_1 \mapsto a; x_2 \mapsto b]$ evaluates ϕ to true. We show that s_b simulates t_a, if $c \neq b$ then $s_b \xrightarrow{c} \bot$ which simulates anything, and $s_b \xrightarrow{a} G_{v[x_1 \mapsto a; x_2 \mapsto b]}$ which is simulated by $A(\phi)$. Therefore s_0 simulates t_v.

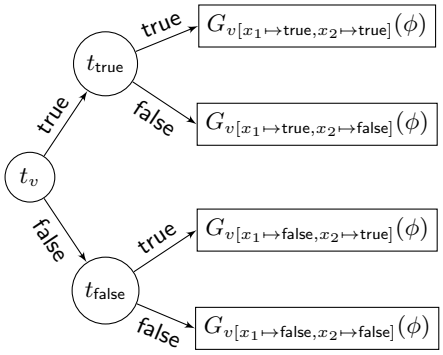

Fig. 11. $G_v(\forall x_1.\ \exists x_2.\ \phi)$

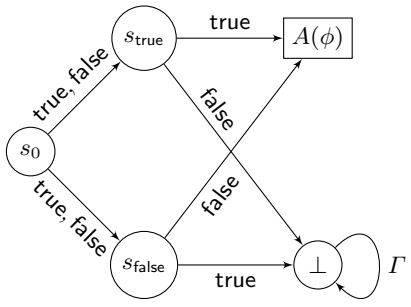

Fig. 12. Automaton $A(\forall x_1.\ \exists x_2.\ \phi)$

We now describe how to generate $G(\phi)$ by a hierarchical system of size polynomial in $|\phi|$. We have a nonterminal A_i for each variable x_i, $1 \leq i \leq n$, and another nonterminal A_{n+1}. The idea of the nonterminal A_i is to define a root state from which if true is read, then one enters into nonterminal A_{i+1} whose contact state $2i$ points to $\top$ and $2i+1$ points to $\bot$, and if false is read, then we enter into another copy of A_{i+1} where the contact states are connected the other way around. Thus choosing a path from the state 1 of A_1 defines a valuation of the variables of ϕ. To get the valuation of a variable x_i, one just have to go to the state pointed by the contact state $2i$, and to $2i+1$ for the valuation of $\neg x_i$. Then, given contact states $1, 2, \ldots, 2n+1$, we define A_{n+1} that produces graph $G_v(\phi)$, where v is determined by the contact states. An example of the nonterminal A_{n+1} is represented in Fig. 14. □

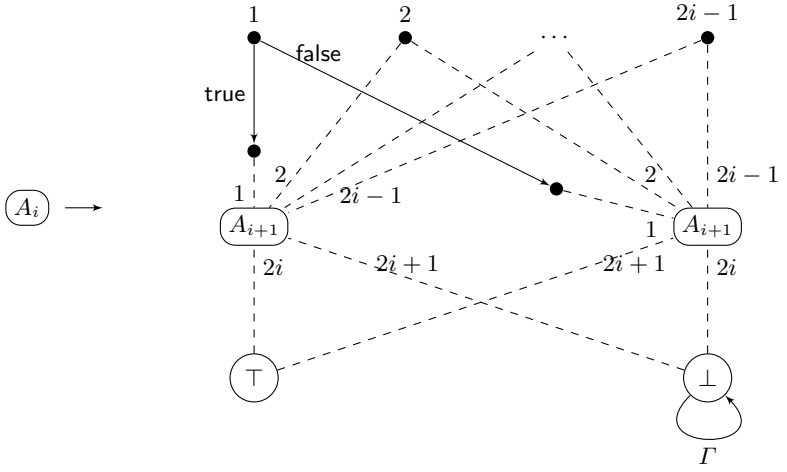

Fig. 13. Production rule A_i for i in $[1, n]$

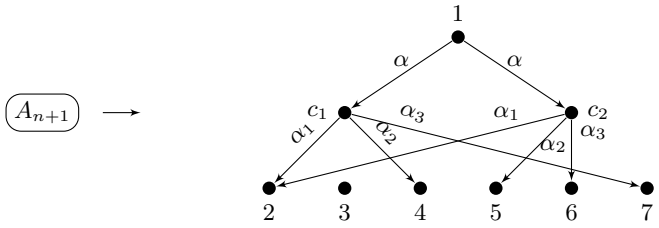

Fig. 14. Example of rule A_{n+1} for the formula $\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$

Lemma 5. *Checking any relation between time-abstract language equivalence and time-abstract bisimulation between a timed automaton and a finite automaton is* PSPACE*-hard.*

Proof. Consider any timed automaton T. We replace all edge labels by α in the initial states, and add self-loops (with no guard) with label α at all states. We also add self-loops labelled by β at all final states. Now, let F denote the flat system consisting of one state, and a self-loop on α.

Now, if action β is available (a final state is reachable) in T, then F and T are not related by any relation between untimed language inclusion and bisimilarity. Otherwise, T is equivalent to F for any such relation. Since reachability is PSPACE-hard for timed automata, we obtain the result. □

Lemma 6. *Checking bisimulation between a hierarchical system and a flat system is* PSPACE*-complete.*

The PSPACE algorithm was described in the core of the paper. We show here the PSPACE-hardness.

A *pushdown automaton (PDA)* is a tuple $\mathcal{A} = (Q, q_0, \Gamma, \Sigma, \delta)$, where Q is a finite set of control states, $q_0 \in Q$ is an initial control state, Γ is a finite *stack alphabet*, Σ is a finite input alphabet, and δ is a finite set of *transition rules*, each either of the following forms:

- $p \xrightarrow{a} q$, where $p, q \in Q$ and $a \in \Sigma$,
- $p \xrightarrow{a} qA$, where $p, q \in Q$, $a \in \Sigma$ and $A \in \Gamma$,
- $pA \xrightarrow{a} q$, where $p, q \in Q$, $a \in \Sigma$, and $A \in \Gamma$.

The induced (pointed) transition system is $\mathcal{T}(\mathcal{A}) = (Q\Gamma^*, q_0, \{\xrightarrow{a} \mid a \in \Sigma\})$, where for all $p, q \in Q$, for all $u, v \in \Gamma^*$, and for all $a \in \Sigma$ we have $pu \xrightarrow{a} qv$ if and only if

- $u = v$ and $p \xrightarrow{a} q \in \delta$,
- $\exists A \in \Gamma : v = Au$ and $p \xrightarrow{a} qA \in \delta$, or
- $\exists A \in \Gamma : u = Av$ and $pA \xrightarrow{a} q \in \delta$.

We call $\mathcal{A}$ *level-finite* if Q can be partitioned into (pairwise disjoint sets) $Q_0, Q_1, \ldots, Q_n$ and moreover the following conditions hold:

1. $q_0 \in Q_0$,
2. if $p \xrightarrow{a} q \in \delta$, then $p, q \in Q_i$ for some $1 \leq i \leq n$,
3. if $p \xrightarrow{a} qA \in \delta$, then $p \in Q_i$ and $q \in Q_{i+1}$ for some $1 \leq i < n$, and
4. if $pA \xrightarrow{a} q \in \delta$, then $p \in Q_{i+1}$ and $q \in Q_i$ for some $1 \leq i < n$.

Note that the transition system of each level-finite PDA is *finite*.

Theorem 11 (see proof of Theorem 16 in [12]). *The following problem is* PSPACE*-hard:*

INPUT: A level-finite PDA $\mathcal{A}$ and a pointed flat system $\mathcal{T}$.

QUESTION: $\mathcal{T}(\mathcal{A}) \sim \mathcal{T}$?

For proving PSPACE-hardness of bisimilarity between hierarchical systems and flat systems, we show that every level-finite PDA $\mathcal{A}$ one can compute an at most polynomially bigger hierarchical system H such that $\mathcal{T}(H)$ is isomorphic to $\mathcal{T}(\mathcal{A})$.

Lemma 7. *The following is computable in polynomial time*
INPUT: A level-finite PDA $\mathcal{A}$.
OUTPUT: A hierarchical system H such that $\mathcal{T}(\mathcal{A})$ is isomorphic to $\mathcal{T}(H)$.

Proof. Assume $\mathcal{A} = (Q, q_0, \Gamma, \Sigma, \delta)$ and let $Q = Q_1 \cup Q_2 \cdots \cup Q_n$ be the partition of Q. Moreover let $\ell_i = |Q_i|$ and $Q_i = \{q_i(1), \ldots, q_i(\ell_i)\}$ for each $1 \leq i \leq n$. We put $H = (N, I, P)$, where

1. $N = \{I\} \cup \{\langle i, X\rangle \mid 1 \leq n, p \in Q_{i-1}, X \in \Gamma\}$,
2. $\mathsf{rank}(\langle i, X\rangle) = \ell_i$ for each $\langle i, X\rangle \in N$ and $\mathsf{rank}(I) = 0$.
3. $\langle i, X\rangle \to (\mathcal{A}, \tau, E)$, where for the ℓ_i-pointed transition system $\mathcal{A} = (A, \{\xrightarrow{a} \mid a \in \Sigma\}, \tau)$ we have
   - $A = \begin{cases} Q_i \cup Q_{i+1} & \text{if } 1 \leq i < n \\ Q_i & \text{if } i = n \end{cases}$,
   - the binary relation $\xrightarrow{a}$ is the smallest relation that contains
     - $q_i(j) \xrightarrow{a} q_i(j')$ if $q_i(j) \xrightarrow{a} q_i(j') \in \delta$,
     - $q_i(j) \xrightarrow{a} q_{i+1}(j')$ if $q_i(j) \xrightarrow{a} q_{i+1}(j')X \in \delta$ (if $1 \leq i < n$), and
     - $q_{i+1}(j) \xrightarrow{a} q_i(j)$ if $q_{i+1}(j)X \xrightarrow{a} q_i(j') \in \delta$ (if $1 \leq i < n$)
   - $\tau(j) = q_i(j)$ for each $1 \leq j \leq \ell_i$
   - in case $1 \leq i < n$ we define E to consist of all $(\langle i+1, X\rangle, \sigma)$ (with $X \in \Gamma$), where $\sigma(j) = q_{i+1}(j)$ for each $1 \leq j \leq \ell_{i+1}$ and in case $i = n$ we put $E = \varnothing$.
4. $I \to (\mathcal{A}, \tau, E)$, where $\mathcal{A}$ is the (0-pointed) transition system $\mathcal{A} = (A, \tau, E)$ with
   - $A = Q_0 \cup Q_1$
   - the binary relation $\xrightarrow{a}$ is the smallest relation that contains
     - $q_0(j) \xrightarrow{a} q_0(j')$ if $q_0(j) \xrightarrow{a} q_0(j') \in \delta$,
     - $q_0(j) \xrightarrow{a} q_1(j')$ if $q_0(j) \xrightarrow{a} q_1(j')X \in \delta$ and
     - $q_1(j) \xrightarrow{a} q_0(j)$ if $q_1(j)X \xrightarrow{a} q_1(j') \in \delta$.
   - we define E to consist of all $(\langle 1, X\rangle, \sigma)$ (with $X \in \Gamma$), where $\sigma(j) = q_1(j)$ for each $1 \leq j \leq \ell_1$.

It is straightforward to see that the state $\mathcal{T}(H)$ and $\mathcal{T}(\mathcal{A})$ are isomorphic. □

Lemma 6 then follows from Theorem 11 and Lemma 7.

B.3 Language Inclusion and Universality

Theorem 10. *Checking untimed language universality is* EXPSPACE*-complete for timed automata with two clocks, and* PSPACE*-complete with one clock. Language universality is* EXPSPACE*-complete for hierarchical systems.*

The proof is by reduction from the acceptance of an exponential space Turing machine. We give the detailed proof for timed automata, then explain how it can be adapted to hierarchical systems.

We denote a Turing machine a tuple $M = \langle \Sigma, Q, q_0, q_f, \delta \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $q_f \notin Q$ is the accepting state, and $\delta \colon Q \times \Sigma \mapsto (Q \cup \{q_f\}) \times \Sigma \times \{\leftarrow, \rightarrow\}$ a transition function. Σ contains a special symbol $\sqcup$ (the blank symbol). δ specifies for the current state $q \in Q$ and symbol $a \in \Sigma$, a triple $\delta(q, a) = (p, a', D)$ where $p \in (Q \cup \{q_f\})$ is the next state, $a' \in \Sigma$ is the symbol to be overwritten over a and $D \in \{\leftarrow, \rightarrow\}$ is the direction in which the cursor will move. A *configuration* of M is a triple (q, w, u), where $q \in (Q \cup \{q_f\})$ is the current state, $w \in \Sigma^*$ is the string on the left of the cursor including the symbol that is currently being read, and $u \in \Sigma^*$ is the string on the right of the cursor. We say that (q, w, u) *yields* configuration (q', w', u') *in one step*, denoted $(q, w, u) \xrightarrow{M} (q', w', u')$ if the latter is obtained by executing one transition in M. The *initial configuration* is (q_0, ϵ, x), where x is a word in $(\Sigma \setminus \{\sqcup\})^*$, called the input word, and ϵ is the empty word. The Turing machine M *accepts* x, if the initial configuration of M with this input, yields a configuration with state q_f.

Let M be an exponential space Turing machine, and $x \in \Sigma^n$ be an input word, w.l.o.g. we can assume that M uses at most 2^n tape cells when started on the word x. We encode a configuration (q, w, u) of M by the word $w \cdot q \cdot u \cdot \sqcup^{2^n - |w| - |u|} \in \Delta$, where Δ is the language $\bigcup_{0 \leq k \leq 2^n} \left(\Sigma^k \cdot Q \cdot \Sigma^{2^n - k}\right)$. An execution of M is encoded by a concatenation of words representing configurations, separated by a special symbol $\$ \notin \Sigma$. Thus, the encoding of executions belong to the language $(\$ \cdot \Delta)^* \cdot \$$.

Given a machine M and a word $x \in \Sigma^n$, we will construct a timed automaton $\mathcal{A}$ over the alphabet $\Gamma = \Sigma \cup Q \cup \{q_f, \$\}$ that is universal if and only if the word x is not accepted by M. The timed automaton $\mathcal{A}$ will recognize all words that do not represent correct executions of M on x, and those words that represent correct executions that stop at some state other than q_f.

If a word in Γ^* does not encode a correct execution of the machine M on input x, then one of the following must hold: (1) it is not a concatenation of configurations; (2) it does not start in the initial configuration; (3) there exists a configuration that does not yield the next one in one step. Notice that any other word in Γ^* encodes a correct finite execution, although it may not be accepting. The automaton $\mathcal{A}$ is made of several parts, each of them recognizing one kind of error. The first part recognizes words that are not sequences of configurations.

Lemma 8. *For any Turing machine M, the timed automaton $\mathcal{B}$ of Fig. 15 recognizes the untimed language $\Gamma^* \setminus (\Delta \cdot \$)^*$.*

We can also easily detect words that do not start with the initial configuration.

Lemma 9. *For any Turing machine M and word w, one can compute a timed automaton $\mathcal{C}(w)$ with two clocks, of polynomial size, that recognizes the untimed language $\Gamma^* \setminus (q_0 w \sqcup^{2^n - n - 1} \cdot \Gamma^*)$.*

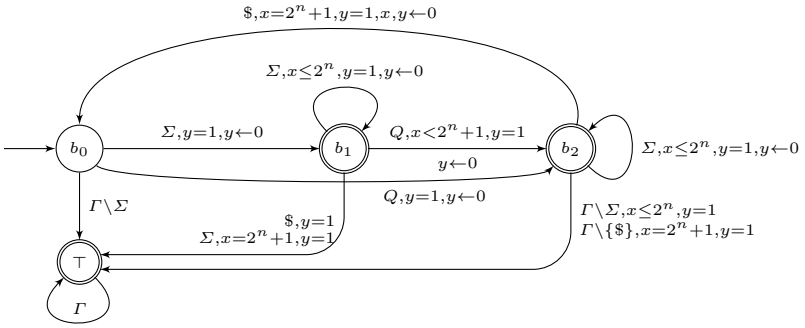

Fig. 15. Timed automaton $\mathcal{B}$.

We then need a timed automaton that recognizes transition errors. These timed automata will check whether some cell in the tape is updated incorrectly in the following configuration, that is, exactly $2^n + 1$ symbols further in the word. To check such inconsistencies, we use the timed automaton $D(a, m)$ defined in Fig. 16, which accepts, for $a \in \Sigma$ and $m \geq 1$ any word whose m-th symbol is not a. More precisely, $D(a, m)$ accepts $(\Gamma \setminus \{q_f\})^m \cdot \Gamma \setminus \{a\} \cdot \Gamma^*$.

In the lemma below, the timed automaton $E(\cdot)$ detects whether the current cell content is updated incorrectly during a transition, while $F(\cdot)$ detects whether an inactive cell of the tape is changed. Both timed automata can be defined easily using module $D(a, m)$.

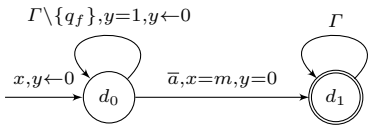

Fig. 16. Module $D(a, m)$

Lemma 10. *Consider any transition* $\delta(q, a) = (q', a', d)$ *where* $d \in \{\leftarrow, \rightarrow\}$.
If $d = \leftarrow$*, then there exists a timed automaton* $E(a, q, b)$ *with two clocks that recognizes the untimed language* $a \cdot q \cdot b \cdot (\Gamma \setminus \{q_f\})^{2^n-2} \cdot (\Gamma^* \setminus (q' \cdot a' \cdot b \cdot \Gamma^*))$.
If $d = \rightarrow$ *then* $E(a, q, b)$ *recognizes* $a \cdot q \cdot b \cdot (\Gamma \setminus \{q_f\})^{2^n-2} \cdot (\Gamma^* \setminus (a' \cdot b \cdot q' \cdot \Gamma^*))$.
There exists a timed automaton $F(a)$ *that recognizes* $(\Sigma \cup \{\$\}) \cdot a \cdot (\Sigma \cup \{\$\}) \cdot (\Gamma \setminus \{q_f\})^{2^n} \cdot \Gamma \setminus \{a\} \cdot \Gamma^*$. *All timed automata have polynomial size.*

Now, the above timed automata can be combined to detect any error in words encoding executions of an exponential space Turing machine on some word w. The idea is to have a self-loop on the initial state i, that reads any symbol but q_f: Either an error is found and the word is accepted, or the automaton stays in i, and it can only accept if symbol q_f does not appear in the word. The overall construction is illustrated in Fig. 17. Note that if the Turing machine M does not accept the word x, then any word in Γ^* that contains q_f necessarily contains an encoding error. Hence, the timed automaton in Fig. 17 accepts Γ^* if, and

only if input x is not accepted by M. Since each part is independent, the timed automaton can be defined with only two clocks.

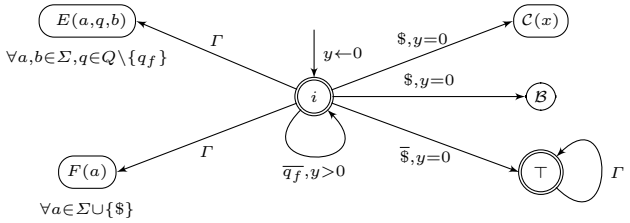

Fig. 17. Timed automaton that accepts incorrect encodings and (correct) encodings of non-accepting executions.

Proof for Hierarchical Systems The same reduction can be applied for hierarchical systems. Consider for instance the module $D(a, m)$, which basically defines a chain of length m, where the first $m-1$ edges are labelled by all symbols in $\Gamma\setminus\{q_f\}$, and the m-th edge is labelled by $\overline{a}$. This can be defined by a polynomial-size hierarchical system as in Fig. 2. Similarly, it is easy to define a hierarchical system bisimilar to $\mathcal{C}(w)$. Last, a hierarchical system defining a system bisimilar to $\mathcal{B}$ is given in Fig. 18.

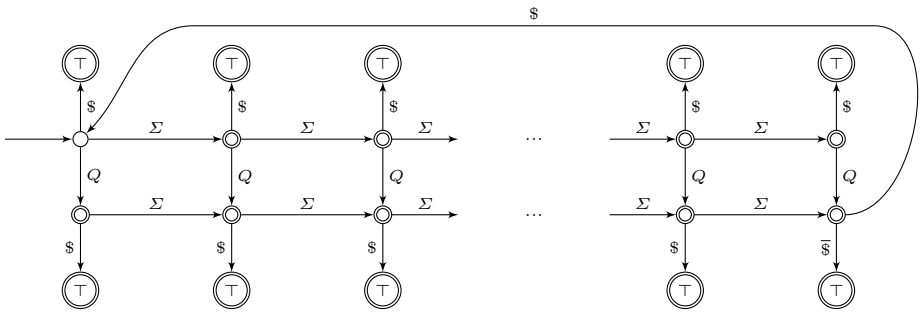

Fig. 18. A finite automaton bisimilar to timed automaton $\mathcal{B}$. Note that there are $2^n + 1$ nodes at each row. All states are accepting except for the initial state. The states labelled by $\top$ also have self-loops reading any symbol in Γ. This automaton is definable by a hierarchical system of size polynomial in n by adapting the example of Fig. 2.

For timed automata with one clock, language inclusion is PSPACE-hard since it is already for flat systems. The polynomial space algorithm follows from the

fact that the region automaton has only polynomial size for timed automata with one clock [13].