Complexity bounds for the verification of real-time software

Rohit Chadha[1⋆], Axel Legay[2], Pavithra Prabhakar[3⋆⋆], and Mahesh Viswanathan[3⋆⋆⋆]

[1] LSV, ENS Cachan & CNRS & INRIA, France
[2] Dept. of Computer Science, University of Illinois at Urbana-Champaign, U.S.A
[3] IRISA, campus de Beaulieu, INRIA Rennes, France
chadha@lsv-ens.cachan.fr, alegay@irisa.fr, pprabha2@uiuc.edu, vmahesh@uiuc.edu

Abstract. We present uniform approaches to establish complexity bounds for decision problems such as reachability and simulation, that arise naturally in the verification of timed software systems. We model timed software systems as timed automata augmented with a data store (like a pushdown stack) and show that there is at least an exponential blowup in complexity of verification when compared with untimed systems. Our proof techniques also establish complexity results for boolean programs, which are automata with stores that have additional boolean variables.

1 Introduction

Timed automata [3] are a standard model for formally describing real-time systems. They are automata equipped with real-valued clocks that evolve continuously with time and which can be compared to integers, and reset during discrete transitions. When modelling concurrent real-time software systems, this basic model must be augmented with various data structures to capture different features — a program stack (visible [4] or otherwise) to model recursive procedure calls, a bag or buffer to model undelivered messages in a network, or a higher-order stack to capture safe higher-order functions. In this paper, we study the complexity of classical verification problems for such formal models of real-time software, namely, invariant verification, μ-calculus model checking, and simulation and bisimulation checking.

Our main thesis is that there is *at least* an exponential blowup in complexity for verifying real-time systems when compared with non-real-time systems. More precisely, the problem of verifying a property for automata with an auxiliary data store (like stack, bag or higher-order stack) and clocks is exponentially harder than checking the same property for the automata model without clocks.

⋆ The research was carried out while Rohit Chadha was at University of Illinois at Urbana-Champaign. He was supported by NSF 0429639.
⋆⋆ Supported by NSF 0448178.
⋆⋆⋆ Supported by NSF 0448178.

In general, the increase in complexity could be worse than exponential. For example, (timed) language containment for timed automata is undecidable [3], but is decidable in PSPACE for finite automata (without clocks). However, we also show that for certain properties (specifically, invariant and μ-calculus model checking, simulation and bisimulation checking) the increase in complexity is *exactly* exponential by establishing upper bounds for solving timed games.

This increase in complexity has been implicitly observed through a series of results that established the complexity of verification problems for the basic timed automata model. It has also been explicitly observed in [18] again for timed automata. In this paper we extend this line of work for timed automata with an auxiliary store. However, there is an important difference in the proof techniques used in the earlier papers and the one we use here. While previous papers established lower bounds by coming up with new, non-trivial reductions for timed automata, we obtain our results by using a *uniform* method to lift lower bound proofs for automata without clocks to automata with clocks, *independent* of the verification problem and the auxiliary data store being considered.

More precisely, our main technical result is that if a verification problem for automata without clocks is hard for complexity class C_1 with respect to *poly-log-time* reductions then the same verification problem is hard for an exponentially larger class C_2 with respect to *polynomial time* reductions. In order to prove this, we rely on the following techniques. First, we draw on the ideas previously used in proving the complexity of problems whose input is succinctly represented as a circuit [13, 22, 6, 27] to show that verifying boolean automata is exponentially harder. Boolean automata are automata with auxiliary data stores that have additional boolean variables. Such models arise when a program is abstracted using predicates [14] inferred through a process of counterexample-guided abstraction-refinement [10]. Thus, our observations about boolean automata are of independent interest. Next, we show that automata with clocks can mimic the behavior of automata with boolean variables, and hence establishing the main lemma for timed systems.

While poly-log-time reductions are a stricter class of reductions than polynomial time reductions, we observe that typically reductions satisfy these stronger conditions because they have a highly regular structure that depends only on certain local bits. We establish that this intuition does indeed hold when considering the reductions that establish lower bounds for the invariant and μ-calculus model checking, and simulation and bisimulation checking for finite state systems and pushdown systems. Thus, using our main technical lemma and our observations about reductions used in classical verification problems, we establish new complexity results for timed automata with data stores, and re-establish old results using new, uniform proof techniques.

Before concluding this introduction, we would like to make a couple of points about our new proofs. First, the new proofs are significantly easier to establish, as new reductions are not required. For the new proof, one needs to re-examine classical reductions for automata problem to check that they are poly-log-time, but this requires much less creativity than constructing a new reduction. Second,

and more importantly, from a philosophical standpoint the new proof is appealing since it highlights clearly some reasons why the verification of real-time and embedded systems is harder than that of non-real-time systems.

Our Results: We show the following complexity results.

1. The control state reachability problem for timed automata is PSPACE-complete and for pushdown timed automata is DEXPTIME-complete.
2. The bisimulation and simulation problems between timed automata is DEXPTIME-complete.
3. The bisimulation and simulation problems between two visibly pushdown timed automata is 2-DEXPTIME-complete.
4. The bisimulation and simulation problems between a timed automata and a pushdown timed automata is 2-DEXPTIME-complete.
5. Model checking μ-calculus properties for order n higher order pushdown timed systems is $(n+1)$-DEXPTIME-complete.

The first two results were previously known, but our proofs for them are new. The remaining three results are completely new.

The rest of the paper is structured as follows. We start by discussing related work. In Section 2, we recall background material that we will use. Next, in Section 3, we establish lower bounds and upper bounds for boolean automata with data stores. The complexity of the verification problems for timed automata is studied in Section 4, and finally in Section 5 we present our conclusions.

1.1 Related Work

Verification problems for systems implicitly represented as a parallel composition of many processes, or using boolean variables has been studied since the work of Harel et al., and Rabinovich [16, 23, 24], where the exponential blow-up in complexity was first observed for model checking branching time modal logics. This observation was extended to process algebraic equivalences and to timed automata in [18]. All of these results were established through new reductions for automata without an additional data store. Alur and Dill [3], introduced the model of timed automata, and showed that the reachability problem is PSPACE-complete. Decidability of simulation and bisimulation was shown in [8], while tight lower bounds were established in [18]. Complexity of model checking μ-calculus was shown in [2]. For timed automata A and B, the language containment problem (i.e., whether $L(A) \subseteq L(B)$), was shown to be decidable when B has one clock [21], and undecidable otherwise; for infinite strings the language containment is undecidable even when B has one clock [1]. The model of pushdown timed systems was first studied in [7] where reachability was shown to be decidable; the decidability of binary reachability was demonstrated in [11]. The language containment problem for timed systems with pushdown stacks and visibly pushdown stacks [4], was studied in [12]. For systems A and B, the problem of whether $L(A) \subseteq L(B)$ was shown to be undecidable when both A and B have visibly pushdown stacks. They also conjectured that the problem is decidable when B is a simple timed automata (without stack) with one clock; however, this problem remains open.

2 Preliminaries

Transition Systems. Given a set of *edge labels* Σ_E, a Σ_E*-transition system* (transition system when Σ_E is clear from the context) $\mathbf{S}$ is a tuple $(\mathcal{S}, \longrightarrow, s_0)$ such that $\mathcal{S}$ is a set of *configurations*; Σ_E is a *finite* set of *labels*; $\longrightarrow \subseteq \mathcal{S} \times \Sigma_E \times \mathcal{S}$ is a set of *transitions* and $s_0 \in \mathcal{S}$ is the *initial configuration*. A transition system is said to be *finite* if Σ_E and $\mathcal{S}$ are finite. We often write $s \xrightarrow{a} s'$ instead of $(s, a, s') \in \longrightarrow$. As usual, we can define $s \xrightarrow{w} s'$ for all $w \in {\Sigma_E}^*$. We say that $s \longrightarrow^* s'$ if there exists $w \in \Sigma_E^*$ such that $s \xrightarrow{w} s'$. We assume the reader is familiar with the definitions of reachability, simulation and bisimulation for transition systems and the logic μ-calculus.

Decision problems, succinct and long representations. We assume that inputs to decision problems are encoded as finite words over the alphabet $\Gamma = \{0, 1\}$. A *problem* L over Γ is a subset of Γ^*. Following [6], a *succinct representation* of a word $w \in \Gamma^*$ is a boolean circuit that on input (the binary representation of) i outputs two boolean values, one indicating whether i is less than or equal to the length of w and the other indicating, in that case, the i-th bit of w. Given a problem L, the *succinct representation of* L, denoted $s(L)$, is the set of all boolean circuits which are succinct representations of words in L [6]. The set $long(L)$ is the set of all strings whose length is equal to the number represented by some binary string $1w$ in L [6].

Indirect access Turing Machines and polylog-time computations. We recall the definition of *indirect access turing machines* [6] used to define complexity classes of low computational power. The Turing machines accepting languages in these classes do not have enough time to read the whole input. Hence *indirect access turing machines* are defined. The machine includes the following elements:

- an input tape;
- a fixed number of work tapes;
- a special tape (henceforth called the pointer tape) to point to a bit of the input, which may be subsequently read in;
- a special tape (henceforth called the symbol tape) on which the symbol just read from the input appears written;
- a "read" state.

The machine is otherwise standard. It reads its input in the following way: the machine can write on the pointer tape the number of position i of the input tape; whenever the "read" state is entered the machine gets (in one computation step) in the symbol tape the contents of the i-th position of the input tape. If the input has length less than i, then the machine does not get anything. The previous content of the symbol tape is overwritten, but the contents of the pointer tape and position of its head remain untouched.

We will denote by LT, the class of languages accepted by deterministic indirect access Turing machines within a computation time bounded by $O(log\ n)$. The class PLT is the class of languages accepted by deterministic indirect access

Turing machines within a computation time bounded by $O((\log n)^k)$ for some natural number k and the class FLT is the class of all functions computable by such machines in $O((\log n)^k)$ time for some natural number k.

Polylog time and polynomial time reductions. Given two problems A and B, A is *polynomial time m-reducible* to B, denoted $A \leq_m^P B$, if and only if there is a polynomial time computable function f such that $w \in A \Leftrightarrow f(w) \in B$ for every string w. *Polylog time m-reducibility* (abbreviated as PLT-reducibility and denoted $\leq_m^{PLT}$) is defined as follows. A is PLT-reducible to B if and only if there is a function f such that i) $w \in A \Leftrightarrow f(w) \in B$ for every w; and ii) the following function φ is computable in polylogarithmic time: for $w \in \{0,1\}^*$ and $i \in N$, $\varphi(w,i)$ is the i-th bit of $f(w)$ if i is less than or equal to the length of $f(w)$ and is undefined otherwise.

We now present two results from [6]. The first result relates PLT-reducibility and polynomial-time m-reducibility[4]. The second result shows that the succinct version of $long(A)$ is at least as hard as A.

Lemma 1. *If* $A \leq_m^{PLT} B$, *then* $s(A) \leq_m^P s(B)$.

Lemma 2. $A \leq_m^P s(long(A))$.

Automata with auxiliary stores. An *automata with auxiliary store* [9] consists of a control and an auxiliary store. Formally, an *auxiliary store* is a tuple $\mathcal{D} = (D, \widetilde{pred}, \widetilde{op}, d_i)$ such that D is a set, elements of which are called *data values*; $\widetilde{op}$ is a finite collection of functions $f : D \to D$; $\widetilde{pred}$ is a finite collection of unary predicates on D; and d_i is an element of D, called the *initial data value*. It is assumed that the identity function $\mathsf{id} \in \widetilde{op}$ and the always true predicate $\mathsf{true} \in \widetilde{pred}$. Pushdown stores, visibly pushdown stores [5], and higher-order pushdown stores [15] can be seen as instances of auxiliary stores.

An automaton is defined over an auxiliary store and a *finite* alphabet (the alphabet is used to annotate the transitions of the automaton). Formally, given an auxiliary store $\mathcal{D} = (D, \widetilde{pred}, \widetilde{op}, d_i)$ and an alphabet Σ_E, a $(\mathcal{D}, \Sigma_E)$-*automaton* $\mathcal{A}$ is a tuple (Q, δ, q_i), where

- Q is a *finite* set of *control states*.
- $\delta \subseteq Q \times \widetilde{pred} \times \Sigma_E \times \widetilde{op} \times Q$ is a *transition relation*.
- q_i is the *initial state* of the automaton $\mathcal{A}$.

The semantics of $\mathcal{A}$ is described in terms of a Σ_E-labeled transition system $(\mathcal{S}, \longrightarrow_\delta, s_i)$. The set of configurations is $\{(q,d) \mid q \in Q \text{ and } d \in \mathcal{D}\}$ and (q_i, d_i) is the initial configuration. The transition relation $\longrightarrow_\delta$ is defined as follows– $(q,d) \xrightarrow{a}_\delta (q',d')$ iff there exists $p \in \widetilde{pred}$ and $g \in \widetilde{op}$ such that $(q,p,a,g,q') \in \delta$, $p(d)$ is true and $g(d) = d'$. We will assume the definition of *isomorphism* between automata.

[4] The result in [6] is only shown for log time m-reducibility, but the extension is straightforward.

For example, a pushdown store on an alphabet Γ in a pushdown automaton can be formalized as an auxiliary store in the following way. The set Γ^* (set of all finite strings over Γ) can be taken as the set of data values with the empty string ϵ as the initial value. The set of predicates $\widetilde{pred}$ can be chosen as $\{\mathsf{empty}\} \cup \{\mathsf{top}_\gamma \mid \gamma \in \Gamma\} \cup \{\mathsf{true}\}$, where $\mathsf{empty} = \{\epsilon\}$, $\mathsf{top}_\gamma = \{w\gamma \mid w \in \Gamma^*\}$ (the top of stack is γ) and $\mathsf{true} = \Gamma^*$ (any stack). The set of functions $\widetilde{op}$ can be defined as $\{id\} \cup \{\mathsf{push}_\gamma \mid \gamma \in \Gamma\} \cup \{\mathsf{pop}_\gamma \mid \gamma \in \Gamma\}$ where push_γ and pop_γ are defined as follows. For all $w \in \Gamma^*$, $\mathsf{push}_\gamma(w) = w\gamma$ and $\mathsf{pop}_\gamma(w) = w_1$ if $w = w_1\gamma$ and w otherwise. In a pushdown system the function pop_γ will be enabled only when the store satisfies top_γ. The function push_γ is enabled when the store satisfies true.

We also consider *visibly pushdown automata* (*VPA*) [5]. A *VPA* is a special kind of pushdown automaton in which every symbol of the input alphabet is designated as a call, return or internal. Every transition labelled by a call pushes a symbol, those labelled by return pop a symbol and transitions labelled by internal symbols do not push or pop any symbols.

The other kind of automata that will be considered are *higher order pushdown systems*. First let us define a higher order store. Given an alphabet Σ, an order 1 store is a stack of elements from Σ, and an order n store for $n > 1$ is a stack of elements from the set of stores of order $n-1$. The different kinds of operation that can be performed on an order n store include push_w where w is a word of the input alphabet, push_l where $l \leq n$, pop_l where $l \leq n$. A stack is written with the top most element to the left. The first order 1 store in an order 2 store ABC, where A, B and C are order 1 stores, is A, and the first order 2 store is ABC itself. push_w pushes w onto the first order 1 stack in the store. push_l pushes a copy of the first element of the first order l store to the first order l store in the store, and pop_l pops the top element of the first order l store in the store. There is another operation *top* which returns the top element of the first order 1 store in the store. A higher order pushdown system of order n is an automaton equipped with an order n store. A transition can be taken only if the element returned by *top* matches the input symbol and the data store is modified according to the operation. A formal description can be found in [15].

Automaton Problem. A k-tuple of automata with auxiliary store has signature $\mathsf{Sig} = ((\mathcal{D}_1, \Sigma_{E1}), \cdots, (\mathcal{D}_k, \Sigma_{Ek}))$ if the i-th automaton in the tuple has auxiliary store $\mathcal{D}_i$ and alphabet Σ_{Ei}. A (k, Sig)-*automaton problem* is a set of k-tuples of automata having signature Sig. For the rest of the paper, we assume that an automaton problem is closed under isomorphism, *i.e.*, if $\mathcal{P}$ is a k-automaton problem, then for any k-tuple $(\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_k)$, $1 \leq i \leq k$, and $\mathcal{A}'_i$ such that $\mathcal{A}_i$ is isomorphic to $\mathcal{A}'_i$, $(\mathcal{A}_1, \mathcal{A}_2, \cdots, A_i, \cdots \mathcal{A}_k) \in \mathcal{P}$ iff $(\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}'_i, \cdots \mathcal{A}_k) \in \mathcal{P}$.

For example, the *simulation* poblem between pushdown automata over the same pushdown store over the same input alphabet Σ_E is the set of pairs $(\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1$ is simulated by $\mathcal{A}_2$.

Encoding of an automaton. We encode the automaton over an auxiliary store $\mathcal{D}$ and an alphabet Σ_E as a binary string. Let n_1, n_2, n_3 and n_4 be the least integers

such that $|Q| \leq 2^{n_1}$, $|\Sigma_E| \leq 2^{n_2}$, $|\widetilde{pred}| \leq 2^{n_3}$ and $|\widetilde{op}| \leq 2^{n_4}$, respectively. We will assume some enumeration of the elements in Q (the initial state will always be numbered 0), Σ_E, $\widetilde{pred}$ and $\widetilde{op}$. The automaton is then encoded as a binary string w of length 2^n, where $n = n_2 + n_3 + n_4 + 2n_1$ as follows. We basically encode the transition function. Note that any position of the encoding can be represented by a binary number of length $n_2 + n_3 + n_4 + 2n_1$. The first n_2 bits index the edge symbol of a transition, the next n_3 bits index the predicate of the transition, the next n_4 bits index the operation and the next n_1 bits index the "current control state" and the final n_1 bits index the "next control state".

Before giving the formal encoding, let us fix some notation. Given a binary string $w \in \Gamma^*$, let $|w|$ denote the length of w and w_i denote its i-th symbol (counting from left to right) of w. Given a binary string w, $\hat{w}$ represents the natural number whose binary expansion is w.

Let w be the encoding of the automaton. Then $w_i = 1$ if and only if $i = \hat{s}$ and $s = xvyuz$ where $|x| = n_2$, $|v| = n_3$, $|y| = n_4$, $|u| = n_1$ and $|z| = n_1$, and $(q_1, p_1, e_1, o_1, q_2) \in \delta$, where q_1 is the $\hat{u}$-th symbol of $|Q|$, p_1 is the $\hat{v}$-th symbol of $\widetilde{pred}$, e_1 is the $\hat{x}$-th symbol of Σ_E, o_1 is the $\hat{y}$-th symbol of $\widetilde{op}$ and q_2 is the $\hat{z}$-th symbol of Q. $w_i = 0$, otherwise.

We now describe how a tuple $(\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_n)$ of automata is encoded. Let $x_1, \cdots, x_n$ be the encoding of the automata $\mathcal{A}_1, \cdots, \mathcal{A}_n$, respectively as explained above. Let k be the length of the maximum of the lengths of x_is. We pad 0s to the end of the x_is if required so that their length is k. Let these new strings be $x'_1, \cdots, x'_n$. Let y_i be a string of length k with j_i 1s followed by 0s, where j_i is the number of bits required to index the states of $\mathcal{A}_i$ (which was n_1 in the encoding of the individual automaton). The encoding of the automata tuple would be $y_1 x'_1 y_2 x'_2 \cdots y_n x'_n$.

3 Boolean Automata with Stores

In this section, we establish complexity bounds for the verification of boolean automata, which we later use (in Section 4) to prove complexity bounds on the verification of real-time software. Boolean automata with stores are automata with auxiliary stores that are equipped with additional boolean variables that influence the enabling condition for transitions. We show that solving an automaton problem when the inputs are given as boolean automata is at least as hard as solving the same when the input automata are represented succinctly using circuits. This observation allows us to lift lower bound proofs for automata uniformly to those for boolean automata. Next we show that solving any problem on boolean automata is at most exponentially worse than solving the same problem for automata without boolean variables. We conclude this section by using these observations to establish the exact complexity for a variety of verification problems for boolean automata.

Definitions and notations. Let $Var = \{x_1, \cdots, x_n\}$ be a finite set of boolean variables. A *valuation* of Var is a function $v : Var \to \{0, 1\}$. The set of boolean formulas over Var, denoted $BFor(Var)$ is defined inductively as: $\varphi := \top \mid x \mid \neg\varphi \mid \varphi \vee$

$\varphi \mid \varphi \wedge \varphi$, where $x \in \mathit{Var}$. Given $\varphi \in \mathit{BFor}(\mathit{Var})$, and a valuation v of Var, $(\varphi)_v$ is defined inductively as $(\top)_v = 1$, $(\neg\varphi)_v = 1 - (\varphi)_v$, $(\varphi_1 \vee \varphi_2)_v = \max((\varphi_1)_v, (\varphi_2)_v)$, and $(\varphi_1 \wedge \varphi_2)_v = \min((\varphi_1)_v, (\varphi_2)_v)$. Next we define the set of reset expressions over Var, denoted $\mathit{BReset}(\mathit{Var})$, as the expressions of the form $x := y$, $x := 0$, $x := 1$, or $\mathit{nondet}(x)$, where $x, y \in \mathit{Var}$. Let us fix $\mathit{Var} = \{x_1, \cdots, x_n\}$ with the ordering $x_1, \cdots, x_n$ for the rest of this section. We define $\mathit{TReset}(\mathit{Var})$ to be n-tuples over $\mathit{BReset}(\mathit{Var})$ where the i-th component of the tuple is of the form $x_i := x_j$, $x_i = 0$, $x_i = 1$ or $\mathit{nondet}(x_i)$. Given $\eta \in \mathit{TReset}(\mathit{Var})$ we denote by η_i, the i-th component of η. $(\eta)_v$ gives the valuations resulting from the application of η to v and is defined as follows. $(\eta)_v$ is the set of valuations v' such that for each i, $v'(x_i)$ is $v(x_j)$ if η_i is $x_i := x_j$, is 0 if η_i is $x_i := 0$, is 1 if η_i is $x_i := 1$, is either 0 or 1 if $\eta_i = \mathit{nondet}(x_i)$. Given a valuation v, $\bar{v}$ will denote the tuple $(v(x_1), \cdots, v(x_n))$.

We now define a boolean automaton as an automaton augmented with a finite set of boolean variables on which the transitions could depend. We have the following definition.

Boolean Automaton. Let $\mathcal{D}$ be a store and Σ_E be an alphabet. A $(\mathcal{D}, \Sigma_E)$-boolean automaton $\mathcal{B}$ is a tuple $(Q, \mathit{Var}, \delta, q_i, v_i)$, where

- Q is a finite set of *control states*.
- Var is a finite set of *control variables*.
- $\delta \subseteq Q \times \mathit{BFor}(\mathit{Var}) \times \widetilde{\mathit{pred}} \times \Sigma_E \times \widetilde{\mathit{op}} \times \mathit{TReset}(\mathit{Var}) \times Q$ is a finite set of *transitions*.
- q_i is the initial state of the automaton.
- v_i is the initial valuation of the variables.

The *semantics* of $\mathcal{B}$ is the same as that of the $(\mathcal{D}, \Sigma_E)$-automaton $[\![\mathcal{B}]\!] = (Q', \delta', q_i')$, where $Q' = Q \times \{0,1\}^n$; $q_i' = (q_i, \bar{v}_i)$; and $((q, \bar{v}), p, e, o, (q', \bar{v}')) \in \delta'$ iff there exists g and r such that $(q, g, p, e, o, r, q') \in \delta$, and $(g)_v = 1$ and $v' \in (r)_v$.

In order to avoid clutter, we do not give the binary encoding of a boolean automaton, but it follows the same lines as the encoding of an automaton. The signature of a k-tuple of Boolean automata with auxiliary stores is defined as for the case of automata. A k-boolean automaton problem is a set of k-tuples which have the same signature. The boolean version of a k-automaton problem $\mathcal{P}$, which we denote $b(\mathcal{P})$, is defined as $b(\mathcal{P}) = \{(B_1, B_2, \ldots B_k) \mid ([\![B_1]\!], [\![B_2]\!], \ldots [\![B_k]\!]) \in \mathcal{P}\}$.

3.1 Lower bounds for boolean automata

We now show that the boolean version of an automaton problem is at least exponentially harder than the automaton problem itself. We first show that the boolean version of an automaton problem is at least as hard as its succinct version. This result will allow us to lift lower bound proofs uniformly. In order to carry out these steps, we need the technical definition of a *two-step automaton* which follows next.

Two-step automaton. Informally, a two-step of an automaton $\mathcal{A}$ is a collection of automata, where an automaton in this collection is obtained by replacing each transition of $\mathcal{A}$ by two consecutive transitions having the same label as the original transition. In addition there are one or more transitions out of every state to some dead states. Formally, given a $(\mathcal{D}, \Sigma_E)$-automaton $\mathcal{A} = (Q, \delta, \Sigma_E)$, an automaton $\mathcal{A}' = (Q', \delta', \Sigma'_E)$ is in *two-step*$(\mathcal{A})$, if there exists a set Y (disjoint from Q and δ) such that the following conditions hold. The states of Q' are $Q \cup \delta \cup Y$. For every transition $x = (q_1, p, e, o, q_2)$ in $\mathcal{A}$, there are transitions $(q_1, \mathsf{true}, e, id, x)$ and (x, p, e, o, q_2) in $\mathcal{A}'$. For every $q_1 \in Q$ and e, there are transitions $(q_1, \mathsf{true}, e, id, q_2)$ in $\mathcal{A}'$, for one or more $q_2 \in Y$. Y represents the dead states. There are no other transitions.

For a k-automaton problem $\mathcal{P}$, we define *two-step*$(\mathcal{P})$ to be $\{(\mathcal{A}'_1, \mathcal{A}'_2, \ldots, \mathcal{A}'_k) |$ $\mathcal{A}'_i \in$ *two-step*$(\mathcal{A}_i)$ and $(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k) \in \mathcal{P}\}$. Given an automaton problem $\mathcal{P}$, we say that $\mathcal{P}$ is *two-step expansion invariant* if *two-step*$(\mathcal{P}) \subseteq \mathcal{P}$. For example, consider an automaton problem $\mathcal{P}$ which corresponds to pairs of automata which are bisimilar. Then $\mathcal{P}$ is two-step expansion invariant, since whenever $\mathcal{A}$ and $\mathcal{B}$ are bisimilar and $\mathcal{A}' \in$ *two-step*$(\mathcal{A})$ and $\mathcal{B}' \in$ *two-step*$(\mathcal{B})$, $\mathcal{A}'$ and $\mathcal{B}'$ are bisimilar.

The following important lemma states that the succinct version of an automaton problem can be reduced to the boolean version of its two-step expansion.

Lemma 3. *Let $\mathcal{P}$ be an automaton problem, then $s(\mathcal{P}) \leq^P_m b($two-step$(\mathcal{P}))$.*

Proof (Sketch.) We show the reduction for a 1-automaton problem, the extension to the k-automaton problem is direct. Let $\mathcal{P}$ be an automaton problem. Let $\mathcal{A} \in \mathcal{P}$ be an automaton and C be its succinct representation. We construct in time polynomial in $|C|$, the boolean automaton $\mathcal{B}$ such that $[\![\mathcal{B}]\!] \in$ *two-step*$(\mathcal{A})$, i.e., $\mathcal{B} \in b($*two-step*$(\mathcal{A}))$.

The circuit C computes the encoding of the automaton $\mathcal{A}$. The first half of the encoding consists of n 1s followed by zero or more 0s where n is the number of bits used to represent the states in $\mathcal{A}$. The second half encodes the transition relation. Hence in time polynomial in the size of $|C|$, we can compute the value of n and also fix the most significant bit of the input of C to 1 to obtain the circuit C_δ which encodes only the transition relation. From now on by C we mean C_δ.

Let us name the inputs of C by variables in sets X, P, O, E and Y (will assume the sets are ordered) such that the inputs corresponding to variables X are used to index the current state; similarly the inputs labelled by P, O, E and Y are used to index the predicates, operations, edge labels and next states of a tuple (q_1, p, o, e, q_2), respectively. A valuation v to the variables corresponding to the tuple (q_1, p, o, e, q_2) when input to C evaluates to 1 if and only if (q_1, p, o, e, q_2) is a transition of $\mathcal{A}$. (Note the number of variables in X and Y are the same.)

The idea is to encode the states of $\mathcal{A}$ using boolean variables and use the circuit to somehow verify the transition relation. Since the boolean automaton can have boolean formulas as guards and not circuits, we verify the transition relation by converting the circuit $C(X, P, O, E, Y)$ to a boolean guard $\varphi(X, P, O, E, Y, I)$ such that for a transition $t = (q_1, p, o, e, q_2)$, $C(q_1, p, o, e, q_2) =$ 1 iff $\varphi(q_1, p, o, e, q_2, I)$ is satisfiable, when I is a new set of variables.

However to check the satisfiability of $\varphi(X, P, O, E, Y, I)$ we need to guess the values of the variables in I. Hence the boolean automaton $\mathcal{B}$ we construct has two states, namely, the *current* and the *guess* state. There are six sets of variables X, P, O, E, Y and I. In the current state only the variables of X are non-zero and they correspond to an encoding of a state of the automaton $\mathcal{A}$. From the current state there is a transition to the guess state in which the variables in X remain intact but the variables in P, O, E, Y and I are all non-deterministically set to 0 or 1. The transition is labelled by the label encoded in E, the predicate is a boolean formula which checks that the edge label of the transition is same as that encoded by the variables in E, and the operation is the identity operation *id*. The values of the variables in P, O, E and Y are used to encode the predicate, operation, edge label and the next state, whereas the variables in I correspond to the intermediate variables which arise in the conversion from the circuit to the boolean guard. From the guess state there is a transition to the current state only if the variables satisfy the guard φ. Then the values of the variables in Y are copied to the corresponding variables of X and all variables other than those in X are set to 0. The edge label, predicate, operation of this transition are those encoded in E, P and O respectively.

This construction takes time polynomial in $|C|$ since the number of states is 2, the number of variables is less than the size of $|C|$ (it is just all the input variables and one intermediate variable for each gate in the circuit), and boolean formula is polynomial in the size of $|C|$ (in fact linear) and can be computed in time polynomial in $|C|$. Hence the boolean automaton constructed is polynomial in the size of $|C|$ and the reduction takes polynomial time.

It is easy to see that $[\![\mathcal{B}]\!]$ is in *two-step*$(\mathcal{A})$. Different valuations in the current state correspond to different states of $\mathcal{A}$. Every transition of $\mathcal{A}$ is mimicked in $[\![\mathcal{B}]\!]$ by two consecutive transition, the first one going into the guess state and the other from the guess state to the current state. There are some transitions into the guess state which cannot be verified in the sense that the values of the variables do not satisfy the guard on the transition to the current state, these will occur as transitions from the current state to dead states (which are accommodated in the definition of *two-step*). □

The next theorem establishes the fact that the boolean version of an automaton problem is at least exponentially harder than the automaton problem.

Theorem 1. *Let C_1 and C_2 be arbitrary complexity classes such that for every problem $\mathcal{P}_1$ in C_1, long$(\mathcal{P}_1)$ is in C_2. Then for every automaton problem $\mathcal{P}_2$ which is two-step expansion invariant, if $\mathcal{P}_2$ is hard for C_2 under PLT-reducibility, then $b(\mathcal{P}_2)$ is hard for C_1 under polynomial time m-reducibility.*

Proof Let $\mathcal{P}_1 \in C_1$, we need to show that $\mathcal{P}_1 \leq_m^P b(\mathcal{P}_2)$. Since *long*$(\mathcal{P}_1) \in C_2$, and *long*$(\mathcal{P}_1) \leq_m^{PLT} \mathcal{P}_2$, we have from Lemma 1, that $s($*long*$(\mathcal{P}_1)) \leq_m^P s(\mathcal{P}_2)$. But $\mathcal{P}_1 \leq_m^P s($*long*$(\mathcal{P}_1))$, from Lemma 2. Hence $\mathcal{P}_1 \leq_m^P s(\mathcal{P}_2)$. Now from Lemma 3, we have $s(\mathcal{P}_2) \leq_m^P b($*two-step*$(\mathcal{P}_2))$. But since $\mathcal{P}_2$ is expansion invariant we have $s(\mathcal{P}_2) \leq_m^P b(\mathcal{P}_2)$. Hence $\mathcal{P}_1 \leq_m^P b(\mathcal{P}_2)$. Therefore $b(\mathcal{P}_2)$ is hard for C_1 under m-reducibility. □

Note that if C_1 is an exponentially larger class than C_2, then they satisfy the condition in the above theorem. Hence if an automaton problem is hard for C_2, then its boolean version is at least exponentially harder.

3.2 Upper bounds for boolean automata

We can also show that solving the boolean version of an automaton problem is at most exponentially harder than the automaton problem itself.

Proposition 1. *Let $\mathcal{P}$ be a k-automaton problem. If $t(n) \geq n$ and $\mathcal{P} \in \mathsf{DTIME}(t(n))$ (or $\mathcal{P} \in \mathsf{NTIME}(t(n))$) then the boolean automaton problem $b(\mathcal{P}) \in \mathsf{DTIME}(t(2^{O(n)}))$ (or $b(\mathcal{P}) \in \mathsf{NTIME}(t(2^{O(n)}))$, respectively). If $s(n) \geq \log(n)$ and $\mathcal{P} \in \mathsf{DSPACE}(s(n))$ (or $\mathcal{P} \in \mathsf{NSPACE}(s(n))$) then $b(\mathcal{P}) \in \mathsf{DSPACE}(s(2^{O(n)}))$ (or $b(\mathcal{P}) \in \mathsf{NSPACE}(s(2^{O(n)}))$, respectively).*

As an example of the application of this proposition, problem of deciding whether two boolean automata (with no store) are trace equivalent is easily seen to be in EXPSPACE as the trace equivalence problem between finite automata is in PSPACE.

3.3 Results

We demonstrate that Theorem 1 and Proposition 1 can be used to show that for a variety of automata problems, there is exactly an exponential blowup in complexity when we consider inputs that are boolean automata. For the rest of this section, by pushdown boolean automata we shall mean boolean automata with a pushdown stack as the auxiliary store and by boolean automata we shall mean a boolean automata with no store.[5]

We can extend the results on bisimulation and simulation between finite state machines [25], bisimulation and simulation between visibly pushdown automata [26], bisimulation and simulation between finite state systems and pushdown automata [19, 17] and model-checking μ-calculus properties for higher order pushdown automata [20] to obtain the following result.

Theorem 2.

1. *The problem of control state reachability in boolean automata is* PSPACE*-complete.*
2. *The problem of bisimulation and simulation between boolean automata is* DEXPTIME*-complete.*
3. *The problem of bisimulation and simulation between two boolean VPAs is* 2-DEXPTIME*-complete.*
4. *The problem of bisimulation and simulation between boolean automata and pushdown boolean automata is* 2-DEXPTIME*-complete.*
5. *Model checking μ-calculus properties for order n higher order pushdown boolean automata is* $(n+1)$-DEXPTIME*-complete.*

[5] No store is modeled by taking the set of data values to be a singleton.

Proof (*U*pper bounds) First consider the problem of control state reachability in boolean automata. The problem of control state reachability for automata is easily seen to be PTIME-equivalent to the problem $\mathcal{P}$ of deciding given an edge label a, whether there is a trace from the initial state in which action a occurs. (Note that $\mathcal{P}$ can be expressed as an automaton problem in our framework.) Since the problem $\mathcal{P}$ is easily seen to be in NLOGSPACE for automata without store, the control state reachability for automata is in NLOGSPACE. Hence we get that control state reachability problem for boolean automata is in PSPACE.

The other upperbounds are obtained by direct application of Proposition 1 to the problem of bisimulation and simulation between automata which is PTIME-complete [25], the problem of bisimulation and simulation between two *VPA*s which is DEXPTIME-complete [26], the problem of bisimulation and simulation between automata and pushdown boolean automata which is DEXPTIME-complete [20], and model checking μ-calculus properties for order n higher order pushdown automata which is n-DEXPTIME-complete.

(*L*ower bounds) It is easy to see that the problem of reachability in directed graphs is logtime [6] reducible to $\mathcal{P}$. Thus, the succinct version of graph reachability is polynomial time reducible to the problem $b(\textit{two-step}(\mathcal{P}))$ (see Lemma 3). Now, the succinct version of graph reachability is PSPACE-hard [6]. Also $\mathcal{P}$ is two-step expansion invariant. Therefore $b(\mathcal{P})$ is PSPACE-hard which implies that the control state reachability problem for boolean automata is PSPACE-hard.

Next consider the problem of deciding whether a boolean automaton is simulated by a boolean pushdown automaton. Now, the problem of deciding whether a finite state system is simulated by a pushdown automaton is DEXPTIME-complete [17]. The DEXPTIME-hardness of the problem is shown by a reduction from the problem of acceptance by a linear bounded alternating automaton. The following theorem whose proof is sketched in the Appendix 6.1 states that the reduction is in fact done in polylog time.

Theorem 3. *The problem of simulation of a FS by a PDA is* DEXPTIME*-hard with respect to polylog time reductions.*

Since the problem of simulation is two-step expansion invariant, using Theorem 1, we get that the problem of simulation of boolean automata by boolean pushdown automata is 2-DEXPTIME hard. The other results can be obtained similarly using Theorem 1 on the lower bounds of the automata versions of the corresponding problems. □

The first two items have been established in [18, 24], albeit by different methods. The last three items are new. The last item also implies that the μ-calculus satisfiability of boolean automata is DEXPTIME-complete and that of boolean pushdown automata is 2-DEXPTIME-complete since these correspond to order 0 and 1 higher order pushdown boolean systems respectively.

[6] Log-time reductions are a special kind of *PLT* reductions, where the reduction takes place in $O(\log n)$ time, as opposed to poly-log-time.

4 Timed Automata

In this section we define timed automata with auxiliary stores and prove lower and upper bounds for problems on them. Let C be a finite set of symbols, henceforth called *clocks*. The set Φ_C of *clock constraints* over C is defined by $\phi ::= \mathsf{true} \,|\, x \sim k \,|\, x \sim y \,|\, \neg\phi \,|\, \phi \wedge \phi \,|\, \phi \vee \phi$, where $k \in N$ stands for any non-negative integer and $\sim \in \{=, <, >, \leq, \geq\}$ is a comparison operator. A *valuation* for C is a function from the set of clocks to the set of positive reals, *i.e.*, $v : C \to \mathbb{R}_{\geq 0}$. Let *ValSet*$(C)$ be the set of all valuations of C. We say that v satisfies ϕ, denoted $v \models \phi$, if ϕ is true when the variables of ϕ are replaced by their values, i.e., x is replaced by $v(x)$. We denote by $v+t$ the function mapping x to $v(x)+t$, and by $v[X \to 0]$ the function which maps x to 0 if $x \in X$ and maps x to $v(x)$ otherwise.

Timed Automaton. Let $\mathcal{D}$ be an auxiliary store and Σ_E be an alphabet. A *$(\mathcal{D},\Sigma_E)$-timed automaton* is a tuple (Q, C, δ, q_i), where:

- Q is a finite set of *control states*,
- C is a finite set of clocks,
- $\delta \subseteq Q \times \Phi_C \times \widetilde{pred} \times \Sigma_E \times \widetilde{op} \times 2^C \times Q$ is a finite set of transitions, and
- $q_i \in Q$ is the *initial state*.

The semantics of a $(\mathcal{D}, \Sigma_E)$-timed automaton $\mathcal{T} = (Q, C, \delta, q_i)$ is described in terms of a $\Sigma_E \times \mathbb{R}_{\geq 0}$-labeled transition system $(\mathcal{S}, \longrightarrow_\delta, s_i)$, where $\mathcal{S} = Q \times D \times$ *ValSet*(C); $s_i = (q_i, d_i, v[C \to 0])$; and $(q_1, d_1, v_1) \xrightarrow{(a,t)}_\delta (q_2, d_2, v_2)$ iff there exists a transition $(q_1, \phi, p, a, o, C_1, q_2) \in \delta$ such that $v_1 + t \models \phi$, $(v_1 + t)[C_1 \to 0] = v_2$, $p(d_1)$ is true, and $o(d_1) = d_2$.

4.1 Lower bounds for timed automata with store

Our goal is to show that solving a timed automaton problem with store is at least as hard as solving a corresponding boolean automaton problem with store. First, we will construct a timed automaton *Timed*$_k(\mathcal{B})$ for a boolean automaton $\mathcal{B}$ which has the following property. The construction of *Timed*$_k(\mathcal{B})$ is along the lines of [18], and is omitted here for lack of space.

Lemma 4. *Let $\mathcal{B}_1$ be a $(\mathcal{D}_1, \Sigma_{E1})$-boolean automata and $\mathcal{B}_2$ be a $(\mathcal{D}_2, \Sigma_{E2})$-boolean automata, and k be the maximum of the number of variables in $\mathcal{B}_1$ and $\mathcal{B}_2$. Then Timed$_k(\mathcal{B}_1)$ and Timed$_k(\mathcal{B}_2)$ are bisimilar iff $\mathcal{B}_1$ and $\mathcal{B}_2$ are bisimilar. Also Timed$_k(\mathcal{B}_1)$ is simulated by Timed$_k(\mathcal{B}_2)$ iff B_1 is simulated by B_2.*

Proof The construction of *Timed*$_k(B)$ is along the lines of [18]. Given a $(\mathcal{D}, \Sigma_E)$-boolean automaton $\mathcal{B} = (Q, Var, \delta, q_i)$ with $|Var| = n$, we define the $(\mathcal{D}, \Sigma_E)$-timed automaton *Timed*$_k(\mathcal{B}) = (Q', C, \delta', q_i')$ for $k \geq n$ as follows. Q' is $(Q \times (\delta \cup (\delta \times [2k]) \cup \{guess\})) \cup \{q_i'\}$. $C = X \cup Y \cup Z \cup W \cup \{step\}$, where $X = \bigcup_{i=1}^n x_i$, $Y = \bigcup_{i=1}^n y_i$, $Z = \bigcup_{i=1}^n z_i$ and $W = \bigcup_{i=1}^n w_i$. The clocks in X and Y will be used to store the corresponding values of the boolean variables in the following

way. If variable v_i has value 0, then the corresponding clock values of x_i and y_i will be such that $x_i = y_i$, and if v_i has value 1, then $x_i < y_i$.

Initially, the automaton has a transition from q'_i to $(q_i, guess)$ with constraint $step = 1$. During this transitions the clocks in X_i and Y_i are reset so that they represent the values of the initial valuation v_i of $\mathcal{B}$. Unless specified the constraint and predicate is taken to be true, the operator is taken to be id and the set of clocks to be reset is taken to be $\emptyset$.

From every state $(q, guess)$, there is a transition to every (q, tr), where $tr = (q, g, p, e, o, r, q')$, with constraint $step = 1 \wedge guard$ where $guard$ corresponds to the boolean guard g, and $step$ is reset to 0.

Next, the reset r of the transition tr of the boolean automaton is simulated by $2k$ steps of the timed automaton. (This is the reason for storing the transition tr of the boolean automaton to be simulated in the state.) Let the valuation of the boolean variables be changed from v to v'. At the beginning the clocks in X and Y store v as explained above. Note that if we do not reset these clocks then their values will always correspond to v. In the first n steps, the values of the clocks in Z and W are reset to values which correspond to v'. From a state (q, tr), there are one or two transitions to $(q, tr, 1)$ having constraint $step = 1$, and similarly there are transitions from $(q, tr, i-1)$ to (q, tr, i) with constraint $step = i$, for each $i \in \{2, \cdots, 2k\}$. The transition from $(q, tr, i-1)$ to (q, tr, i) resets the clocks z_i and w_i if required so that they correspond to the value $v'(x_i)$. (Note that the constraints may use the values of x_j and y_j). For example if there is an assignment $v_i := v_j$ in r, then there will be two transitions which contain $step = i$, one of which checks if $x_j < y_j$ and resets z_i, and the other checks if $x_j = y_j$ and resets both z_i and w_i. Similarly other resets of the boolean variables can be taken care of. In the time steps $n < i \leq k$, no clocks are reset. In the next n steps, that is, $k < i \leq k + n$, the clocks in X_i and Y_j are reset to "copy" the values of Z_i and W_i in the following sense. If $z_i < w_i$ then x_i is reset to ensure that $x_i < y_i$, and if $z_i = w_i$, then both x_i and y_i are reset to ensure $x_i = y_i$. Again no clocks are reset in the steps $k+n < i \leq 2k$. Finally there is a transition from $(q, tr, 2k)$ to $(q', guess)$, with predicate p, operation o, and the clocks in $Z \cup W \cup \{step\}$ are reset. All the above transitions are labelled by e. □

The above lemma allows us to conclude the following results from the lower bound results of Theorem 2:

Theorem 4.

1. *The problem of control state reachability in timed automata is* PSPACE*-hard.*
2. *The problem of bisimulation and simulation between timed automata is* DEXPTIME*-hard.*
3. *The problem of simulation and bisimulation between two timed VPA is* 2-DEXPTIME*-hard.*
4. *The problem of bisimulation and simulation between timed automata and pushdown timed automata is* 2-DEXPTIME*-hard.*
5. *Model checking timed μ-calculus properties for order n higher order pushdown timed systems is* $(n+1)$-DEXPTIME*-hard.*

We note that the first result above has also been established in [3], while the second result was established in [18]. The last three results are however new and (to the best of our knowledge) do not appear in literature. As a byproduct we obtain that model-checking timed μ-calculus formulas for timed systems and pushdown timed systems is DEXPTIME-hard and 2-DEXPTIME-hard by instantiating n to 0 and 1 respectively in the last item.

4.2 Upper bounds for timed automata with store

We now show that the lower bounds for decision problems obtained in Section 4.1 are tight. As these decision problems are mainly concerned with simulation and bisimulation, they can be converted to decision problems on *game graphs* by standard techniques. The game graphs that will arise for timed automata will be infinitely branching and we shall appeal to the region construction [3] in order to deal with the "infinite-branching." We start by recalling the definition of regions and game graphs.

Regions. Regions were introduced in [3] in order to show that reachability in timed systems is decidable. Given a finite set of clocks $\mathcal{C}$ and a natural number n_{max}, we can define an equivalence class on the set of real-valuations $\mathit{ValSet}(\mathcal{C})$ as follows. For a real number r, let $\lfloor r \rfloor$ denote the integral value of r and $\mathit{frac}(r)$ the fractional value of r. We say that for valuations $v_1, v_2 \in \mathit{ValSet}(\mathcal{C})$, v_1 is equivalent to v_2 (denoted as $v_1 \equiv v_2$) iff for all $c, c_1, c_2 \in \mathcal{C}$:

1. $v_1(c) > n_{\mathsf{max}}$ iff $v_2(c) > n_{\mathsf{max}}$;
2. if $v_1(c), v_2(c) \leq n_{\mathsf{max}}$, then $\lfloor v_1(c) \rfloor = \lfloor v_2(c) \rfloor$;
3. if $v_1(c), v_2(c) \leq n_{\mathsf{max}}$, then $\mathit{frac}(v_1(c)) = 0$ iff $\mathit{frac}(v_2(c)) = 0$; and
4. if $v_1(c_1), v_1(c_2), v_2(c_1), v_2(c_2) \leq n_{\mathsf{max}}$, then $\mathit{frac}(v_1(c_1)) \leq \mathit{frac}(v_1(c_2))$ iff $\mathit{frac}(v_2(c_1)) \leq \mathit{frac}(v_2(c_2))$.

The equivalence relation $\equiv$ is of finite index and the set of equivalence classes under $\equiv$ shall henceforth be denoted as $\mathsf{Reg}(n_{\mathsf{max}}, \mathcal{C})$. The following proposition states some well known facts about the region construction [3].

Proposition 2. *Given two valuations v_1 and v_2 such that $\mathsf{Reg}\ (n_{\mathsf{max}}, v_1) = \mathsf{Reg}(n_{\mathsf{max}}, v_2)$, we have the following:*

1. *For each $t_1 \in \mathbb{R}$ there is a $t_2 \in \mathbb{R}$ such that $\mathsf{Reg}(n_{\mathsf{max}}, v_1 + t_1) = \mathsf{Reg}(n_{\mathsf{max}}, v_2 + t_2)$.*
2. *For any $\mathcal{C}_0 \subseteq C$ we have $\mathsf{Reg}(n_{\mathsf{max}}, v_1[\mathcal{C}_0 \to 0]) = \mathsf{Reg}(n_{\mathsf{max}}, v_2[\mathcal{C}_0 \to 0])$.*
3. *For any clock constraint ϕ defined over $\mathcal{C}$ such that the maximum integer appearing in $\mathcal{C}$ is less than n_{max}, we have $v_1 \models \phi$ iff $v_2 \models \phi$.*

Game graphs. A game graph is a graph $\mathsf{G} = (V_P \cup V_O, E)$ such that $V_P \cap V_O = \emptyset$ and $E \subseteq (V_P \times V_O) \cup (V_O \times V_P)$. The nodes in the set V_P are called *proponent nodes* and the nodes in the set V_O are called *opponent nodes*. A binary relation $R \subseteq (V_P \times V_P) \cup (V_O \times V_O)$ is a *game bisimulation* if for every $(v_1, v_2) \in R$ the following two conditions hold:

1. For every $v_1' \in V_P \cup V_O$ such that $(v_1, v_1') \in E$, there is a $v_2' \in V_P \cup V_O$ such that $(v_1', v_2') \in R$ and $(v_2, v_2') \in E$.
2. For every $v_2' \in V_P \cup V_O$ such that $(v_2, v_2') \in E$, there is a $v_1' \in V_P \cup V_O$ such that $(v_1', v_2') \in R$ and $(v_1, v_1') \in E$.

The set of *plays* and *strategies* are defined in the standard way. It is well-known that simulation and bisimulation between transition systems can be stated as reachability games on appropriate game graphs.

Consider, for example, the problem of simulation of a timed transition system $\mathcal{G}_0$ by $\mathcal{G}_1$ (by a timed transition system we mean a transition system arising out of a timed automaton). A proponent node will correspond to a pair of configurations of $\mathcal{G}_0$ and $\mathcal{G}_1$. Since every move of $\mathcal{G}_0$ needs to be simulated by $\mathcal{G}_1$, the proponent moves are those of $\mathcal{G}_0$. Consider the proponent node (C_1, C_2) where C_1 is the configuration of $\mathcal{G}_0$ and C_2 the configuration of $\mathcal{G}_1$. Suppose $\mathcal{G}_0$ takes a transition (a, t) and moves to C_1'. Then, for $\mathcal{G}_0$ to be simulated by $\mathcal{G}_1$, $\mathcal{G}_1$ has to take a transition (a, t) from C_2 . Therefore, the proponent move corresponding to $\mathcal{G}_0$ transitioning to C_1' leads us to the opponent node (C_1', C_2, a, t). Now if $\mathcal{G}_2$ can take a (a, t) transition from C_2 to C_2' then we move from the opponent node (C_1', C_2, a, t) to the proponent node (C_1', C_2'). It is easy to see that $\mathcal{G}_0$ is simulated by $\mathcal{G}_1$, iff proponent does not have a strategy to reach an opponent node from which there is no transition. The case of bisimulation is similar except that the proponent must have moves corresponding to both $\mathcal{G}_0$ and $\mathcal{G}_1$, and a proponent move corresponding to $\mathcal{G}_0$ must be answered by a move of $\mathcal{G}_1$ (and vice-versa). We formalize these game graphs as *timed game graphs.*

Timed Game Graph. As already described above, simulation and bisimulation between timed automata can be cast as reachability games on game graphs. Given a $(\mathcal{D}_0, \Sigma_E)$-timed automaton $\mathcal{G}_0 = (Q_0, \mathcal{C}_0, \delta_0, \hat{q}_0)$ and a $(\mathcal{D}_1, \Sigma_E)$-timed automaton $\mathcal{G}_1 = (Q_1, \mathcal{C}_1, \delta_1, \hat{q}_1)$, let $(\mathsf{Conf}_0, \longrightarrow_{\delta_0}, s_{00})$ and $(\mathsf{Conf}_1, \longrightarrow_{\delta_1}, s_{01})$ be the timed transition systems associated with them. We assume that $\mathcal{C}_0 \cap \mathcal{C}_1 = \emptyset$ (we can always rename clocks). Let $\mathsf{Players}$ be a non-empty subset of $\{0, 1\}$ and $\mathsf{Moves} = \Sigma_E \times \mathbb{R}$. The *timed game graph* corresponding to $\mathcal{G}_0$, $\mathcal{G}_1$ and $\mathsf{Players}$ is given by the game graph $\mathsf{G} = (V_P \cup V_O, E)$ where:

1. $V_P = \{(\mathsf{Players}, Conf_0, Conf_1) \mid Conf_i \in \mathsf{Conf}_i \text{ for } i = 0, 1\}$.
2. Let $V_O^0 = (P \times \mathsf{Conf}_0 \times \mathsf{Conf}_1 \times \mathsf{Moves})$ where $P = \{1 - i \mid i \in \mathsf{Players}\}$.
3. For $(v, w) \in V_P \times V_O$, $(v, w) \in E$ iff $v = (\mathsf{Players}, Conf_0, Conf_1)$, $w = (P, Conf_0', Conf_1', (a, t))$ and there exists $i \in \mathsf{Players}$ such that $Conf_i \xrightarrow{(a,t)}_{\delta_0} Conf_i'$, $P = \{1 - i\}$ and $Conf_{1-i}' = Conf_{1-i}$.
   For $(w, v) \in V_O \times V_P$, $(w, v) \in E$ iff $w = (i, Conf_0, Conf_1, (a, t))$ and $v = (\mathsf{Players}, Conf_0', Conf_1')$, where $Conf_i \xrightarrow{(a,t)}_{\delta_0} Conf_i'$ and $Conf_{1-i}' = Conf_{1-i}$.

So the question of simulation can be cast as a question on the game graph G. Note that G is potentially *infinite-branching* and it is not immediately obvious as to how to solve the game problem. We appeal to the region construction to eliminate the infinite branching as follows.

The idea behind our construction is to use regions on clocks of both the systems (a similar strategy has been used in [8] to show that bisimulation between two timed systems without store is decidable). Let n_{max} be some integer such that n_{max} is greater than any integer occurring in the clock constraints of δ_0 and δ_1. Given $v_0 \in \mathit{ValSet}(\mathcal{C}_0)$ and $v_1 \in \mathit{ValSet}(\mathcal{C}_1)$, we use $\mathsf{Reg}(n_{\mathsf{max}}, v_0, v_1)$ for the region $\mathsf{Reg}(n_{\mathsf{max}}, v)$(the equivalence class of v), where $v \in \mathit{ValSet}(\mathcal{C}_0 \cup \mathcal{C}_1)$ is the valuation such that $v(c) = v_0(c)$ for $c \in \mathcal{C}_0$ and $v(c) = v_1(c)$ for $c \in \mathcal{C}_1$. Let $Conf_0 \in \mathsf{Conf}_0$ and $Conf_1 \in \mathsf{Conf}_1$ be configurations such that $Conf_0 = (q_0, d_0, v_0)$ and $Conf_1 = (q_1, d_1, v_1)$. For a proponent node $v = (\mathsf{Players}, Conf_0, Conf_1)$, let $\mathcal{H}(v) = (q_0, d_0, q_1, d_1, \mathsf{Reg}(n_{\mathsf{max}}, v_1, v_2))$. For an opponent node $w = (i, Conf_0, Conf_1, (a, t))$, let $\mathcal{H}(w) = (i, a, q_0, d_0, q_1, d_1, \mathsf{Reg}(n_{\mathsf{max}}, v'_0, v'_1))$, where $v'_i = v_i + t$ and $v'_{1-i} = v_{1-i}$. We have the following result.

Theorem 5. *The relation $R = \{(u_1, u_2) \mid \mathcal{H}(u_1) = \mathcal{H}(u_2)\}$ is a game bisimulation on the timed game graph.*

Proof(Sketch.) Clearly, $R \subseteq (V_P \times V_P) \cup (V_O \times V_O)$. We show that R is a simulation (that R^{-1} is also a simulation can be shown similarly). In other words, for any $(u_1, u_2) \in R$ and $(u_1, u'_1) \in E$, there is a u'_2 such that $(u_2, u'_2) \in E$ and $(u'_1, u'_2) \in R$. There are two cases– either $u_1, u_2 \in V_P$ or $u_1, u_2 \in V_O$. We consider the case where $u_1, u_2 \in V_P$ (the other case is similar).

Let $u_1 = (Conf_0, Conf_1)$ and $u_2 = (Conf_3, Conf_4)$. There are two possibilities for u'_1: either $u'_1 = (1, Conf'_0, Conf_1, (a, t))$ in which case $0 \in \mathsf{Players}$ and $Conf_0 \xrightarrow{(a,t)}_{\delta_0} Conf'_0$ or $u'_1 = (0, Conf_0, Conf'_1, (a, t))$ in which case $1 \in \mathsf{Players}$ and $Conf_1 \xrightarrow{(a,t)}_{\delta_1} Conf'_1$. We consider the case $u'_1 = (1, Conf'_0, Conf_1, (a, t))$ (the other case is similar).

Let $Conf_0 = (q_0, d_0, v_0)$, $Conf_1 = (q_1, d_1, v_1)$ and $Conf'_0 = (q'_0, d'_0, v'_0)$. We have that $Conf_0 \xrightarrow{(a,t)}_{\delta_0} Conf'_0$. As $Conf_0 \xrightarrow{(a,t)}_{\delta_0} Conf'_0$ there must be a transition $(q_0, \phi, p, a, g, \mathcal{C}, q'_0) \in \delta_0$ such that $p(d_0)$ is true $g(d_0) = d'_0$, $v_0 + t \models \phi$ and $(v_0 + t)[\mathcal{C} \to 0] = v'_0$.

Now, as $\mathcal{H}(u_1) = \mathcal{H}(u_2)$, it must be the case that $Conf_3 = (q_0, d_0, v_3)$ and $Conf_4(q_1, d_1, v_4)$ where v_3, v_4 are such that $\mathsf{Reg}(n_{\mathsf{max}}, v_0, v_1) = \mathsf{Reg}(n_{\mathsf{max}}, v_3, v_4)$. Now, we have by Proposition 2 that there is a t' such that $\mathsf{Reg}(n_{\mathsf{max}}, v_0 + t, v_1 + t) = \mathsf{Reg}(n_{\mathsf{max}}, v_3 + t', v_4 + t')$. This implies that $v_3 + t' \models \phi$. Hence, we have that $Conf_3 \xrightarrow{(a,t')}_{\delta_0} Conf'_3$ where $Conf'_3 = (q'_0, d'_0, v'_3)$ with $v'_3 = (v_3 + t'))[\mathcal{C} \to 0]$. We have that $(u_2, u'_2) \in E$ where $u'_2 = (1, Conf'_3, Conf_4, (a, t'))$.

Note that thanks to Proposition 2, $\mathsf{Reg}(n_{\mathsf{max}}, (v_0 + t, v_1 + t)[\mathcal{C} \to 0]) = \mathsf{Reg}(n_{\mathsf{max}}, (v_3 + t', v_4 + t')[\mathcal{C} \to 0])$. But $\mathsf{Reg}(n_{\mathsf{max}}, (v_0 + t, v_1 + t)[\mathcal{C} \to 0]) = \mathsf{Reg}(n_{\mathsf{max}}, v'_0, v_1 + t)$ and $\mathsf{Reg}(n_{\mathsf{max}}, (v_3 + t', v_4 + t')[\mathcal{C} \to 0]) = \mathsf{Reg}(n_{\mathsf{max}}, v'_3, v_4 + t')$. We get that $\mathcal{H}(u'_1) = \mathcal{H}(u'_2)$. Therefore, we get that $(u'_1, u'_2) \in R$ also. □

Hence, while solving the problems of simulation and bisimulation between timed automata, one can appeal to Theorem 5 and reduce the timed game problem to one without time by constructing the $\mathcal{H}$-bisimulation quotient. Then a winning strategy for the proponent in the timed game graph is obtained by "mimicking" the strategy in the bisimulation quotient. For example, simulation between

timed automata and pushdown timed automata can be converted to a game on pushdown graph by constructing the $\mathcal{H}$-bisimulation of the graph G. Note that the description of the resulting pushdown game however is exponential in size of the input as one needs to construct the regions on the clocks. Further, since the reachability game can be solved in PTIME for finite game graphs and DEXPTIME for pushdown games, we obtain the following results.

Theorem 6.

1. *The control state reachability problem of timed automata is in* PSPACE.
2. *The bisimulation and simulation problems between two timed automata is in* DEXPTIME.
3. *The problem of simulation and bisimulation between two timed VPA is in* 2-DEXPTIME.
4. *The bisimulation and simulation problems between a timed automaton and a pushdown timed automaton is in* 2-DEXPTIME.
5. *Model checking timed μ-calculus properties for order n higher order pushdown timed systems is* $(n+1)$-DEXPTIME-*complete.*

The first two results are known [3, 18, 2] and the last three are new.

5 Conclusions

We established the exact complexity of the problems of reachability, simulation, bisimulation, and μ-calculus model checking for timed automata, timed pushdown automata, and timed higher order pushdown automata. Our proof relied on ideas from succinct representations to uniformly lift lower bound proofs for finite automata, pushdown automata, and higher order pushdown automata to the corresponding timed versions. As an intermediate step we established complexity bounds on the verification of boolean automata (without stacks, with stacks, and with higher order stacks), which are also important models that arise in verification. Thus we re-established some previously known results for timed automata using new proof techniques, and proved many new results about timed pushdown automata and timed higher order pushdown automata.

References

1. P.A. Abdulla, J. Deneux, J. Ouaknine, and J. Worrell. Decidability and complexity results for timed automata via channel machines. In *International Colloquium on Automata, Languages and Programming*, pages 1089–1101, 2005.
2. L. Aceto and F. Laroussinie. Is you model checker on time? In *International Symposium on the Mathematical Foundations of Computer Science*, pages 125–136, 1999.
3. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
4. R. Alur and P. Madhusudan. Visibly Pushdown Automata. In *ACM Symposium on Theory of Computation*, pages 202–211, 2004.

5. Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *STOC*, pages 202–211. ACM, 2004.
6. J.L. Balcázar, A. Lozano, and J. Torán. The complexity of algorithmic problems on succinct instances. *Computer Science*, pages 351–377, 1992.
7. A. Bouajjani, R. Echahed, and R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *International Conference on Hybrid Systems: Computation and Control*, pages 64–85, 1994.
8. K. Cerans. Decidability of bisimulation equivalence for parallel timer processes. In *International Conference on Computer-Aided Verification*, pages 302–315, 1993.
9. R. Chadha and M. Viswanathan. Decidability results for well-structured transition systems with auxiliary storage. In *18th. International Conference on concurrency theory*, volume 4703, pages 136–150, 2007.
10. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided Abstraction-refinement. In *International Conference on Computer-Aided Verification*, pages 154–169, 2000.
11. Z. Dang. Pushdown timed automata: A binary reachability characterization and safety verification. *Theoretical Computer Science*, 302:93–121, 2003.
12. M. Emmi and R. Majumdar. Decision Problems for the Verification of Real-time Software. In *International Conference on Hybrid Systems: Computation and Control*, pages 200–211, 2006.
13. H. Galperin and A. Wigderson. Succinct representations of graphs. *Information and Computation*, 56:183–198, 1983.
14. S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *International Conference on Computer-Aided Verification*, pages 72–83, 1997.
15. Matthew Hague and C.-H. Luke Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. In Helmut Seidl, editor, *FoSSaCS*, volume 4423 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 2007.
16. D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. In *International Conference on Concurrency Theory*, pages 258–272, 1997.
17. A. Kucera. On simulation-checking with sequential systems. In *Asian Computing Science Conference*, pages 133–148, 2000.
18. F. Laroussinie and P. Schnoebelen. The State Explosion Problem from Trace to Bisimulation Equivalence. In *3rd International Conference on Foundation of software Science and Computation Structures*, pages 192–207, 2000.
19. R. Mayr. On the complexity of bisimulation problems for pushdown automata. In *IFIP TCS*, pages 474–488, 2000.
20. C.-H. Luke Ong. On model checking trees generated by higher order recursion schemes. In *IEEE Symposium on Logic in Computer Science*, pages 81–90, 2006.
21. J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *IEEE Symposium on Logic in Computer Science*, pages 54–63, 2004.
22. C. Papadimitriou and M. Yannakakis. A note on succinct representation of graphs. *Information and Computation*, 71:181–185, 1986.
23. A. Rabinovich. Complexity of equivalence problems for concurrent finite agents. *Information and Computation*, 139(2):111–129, 1997.
24. A. Rabinovich. Symbolic model checking for μ-calculus requires exponential time. *Theoretical Computer Science*, 243(2):467–475, 2000.

25. Z. Sawa and P. Jancar. P-Hardness of Equivalence Testing on Finite-State Processes. In *Conference on the Current Trends in Theory and Practice of Informatics*, pages 326–335, 2001.
26. J. Srba. Visibly pushdown automata: From language equivalence to simulation and bisimulation. In *International Workshop on Computer Science Logic*, pages 89–103, 2006.
27. H. Veith. Succinct representation, leaf languages, and projection reductions. In *IEEE Conference on Computational Complexity*, pages 118–126, 1996.

6 Appendix

6.1 Proof of Theorem 3

Proof(*P*roof Sketch.) The problem of "acceptance by alternating linear bounded automata" (LBA acceptance) is EXPTIME-hard with respect to poly-logtime-reductions. We observe that the reduction from LBA acceptance to simulation between PDA processes and deterministic FS processes given in [17] can be done in poly-logtime. Given an alternating TM M and a word w, they construct a finite state machine A and a PDA B such that P simulates F iff M does not accept w. The idea behind the simulation is the following. The PDA wants to prove that M does not accept w and F wants to prove otherwise. So the PDA stores in its stack the current configuration. If the configuration consists of an universal state of M, then F gets to choose the next transition, and PDA executes it by guessing the next configuration. F then might ask P to verify for it that the next configuration was valid. F does that by dictating P to pop the stack and nondeterministically asking it to verify some triple. (Note that the PDA does not gain by cheating because its aim is to show that M accepts w). If the PDA is in an existential state then it gets to choose the next transition. If P ever reaches an accepting configuration, then it has no transitions out of the state, hence will not be able to simulate the moves of F. If P reaches a non-accepting configuration, then it simulates F thereafter.

To show that the reduction is logtime we need to spit out the bits of F and P in poly-logtime. This is possible because to compute a particular bit of F or P, say whether there is a transition between two states with some label and some operation on the pushdown store, we need only local information. For example, to verify that a triple of the next configuration is consistent with previous, the PDA needs to only do local computation like reading the three bits into its control state and then maintaining a counter which will enable it to reach the triple of the previous configuration. It is easy to see in this case that whether there is transition between two states can be computed in poly-log time. (Verifying that the counter value was incremented might take $(\log(n))$ time where n is the length of $|w|$).

Hence there is a poly-logtime reduction from LBA acceptance to simulation between of PDA and FS. Therefore the problem is EXPTIME-hard with respect to poly-logtime reductions. □