Exact Incremental Analysis of Timed Automata with an SMT-Solver

Bahareh Badban and Martin Lange*

School of Elect. Eng. and Computer Science, University of Kassel, Germany

Abstract. Timed automata as acceptors of languages of finite timed words form a very useful framework for the verification of safety properties of real-time systems. Many of the classical automata-theoretic decision problems are undecidable for timed automata, for instance the inclusion or the universality problem. In this paper we consider restrictions of these problems: universality for deterministic timed automata and inclusion of a nondeterministic one by a deterministic one. We then advocate the use of SMT solvers for the exact incremental analysis of timed automata via these problems. We stratify these problems by considering domains of timed words of bounded length only and show that each bounded instance is in (co-)NP. We present some experimental data obtained from a prototypical implementation measuring the practical feasibility of the approach to timed automata via SMT solvers.

1 Introduction

Timed automata as introduced in [3] are one of the most well-established models for the specification and verification of systems in which events can happen arbitrarily close in time [14].

The real numbers present an adequate model for continuous time; systems in which such timing aspects need to be modeled are therefore also called real-time systems. Execution traces of real-time systems can be modeled by timed words—sequences of events which are attached to the time at which they occur. Timed automata then act as acceptors of languages of timed words for instance, and various verification problems on real-time systems can be phrased as classic automata-theoretic problems. For instance the classic problem of determining whether an implementation satisfies a specification can be modeled as a language inclusion problem on timed automata.

Real-time aspects introduce additional complexity. The state space of a timed automaton is in fact infinitely large due to the infinitely many moments in time that the execution along a timed word presents. While emptiness for finite automata is NLOGSPACE-complete, and universality (is $L(\mathcal{A}) = \Sigma^*$?) and inclusion (is $L(\mathcal{A}) \subseteq L(\mathcal{B})$?) are PSPACE-complete [3]; emptiness for timed automata

* The European Research Council has provided financial support under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 259267.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 177–192, 2011.

is already PSPACE-complete and universality and inclusion even are undecidable [3]. This has always limited the analysis of such automata to a great extent. To circumvent this problem extensive research has been done, in which certain limitations are imposed on the structure of the automata, for instance by restricting the resources that a timed automaton has access to.

In this paper we consider the problems of language universality and inclusion for timed automata. In order to obtain effective algorithmic solutions we restrict the universality problem to deterministic timed automata, and in the inclusion problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ we require that $\mathcal{B}$ is deterministic. In Sect. 2 we recall very simple constructions that embed the former as well as the emptiness problem for nondeterministic timed automata into the latter. Because of this we can restrict our attention to this inclusion problem.

In Sect. 3 we then develop an incremental approach to these three problems using an idea that has been developed for bounded model checking: by limiting the space of possible witnesses or counterexamples to the emptiness, universality or inclusion problem one can obtain a computationally easier problem. Formally, we define bounded variants of these three problems which are obtained by adding a further input parameter k. The bounded emptiness problem then asks whether or not there is a word in the language that has length at most k. Bounded variants of the two other problems are defined accordingly. We first give a lower complexity bound on the emptiness and inclusion problem showing that they remain at least NP-hard (for unarily encoded additional input parameter k). A corresponding co-NP-lower bound for the universality problem cannot be presented for lack of space but does exist.

In Sect. 4 we first develop a generic propositional logic over integer and real-valued constraints and show that its satisfiability problem is in NP. We then present an encoding of the bounded inclusion problem into the satisfiability problem for this logic. Encodings of the two other bounded problems can easily be obtained from this one. This shows NP-completeness of the bounded emptiness and inclusion problems as well as inclusion in co-NP of the bounded universality problem; compare this to the PSPACE-completeness of the unbounded versions. Hence, the transition from the unbounded to the bounded problems reduces the complexity measurably but it also is the case that the bounded problems are not necessarily solvable in polynomial time (which would make the approach via satisfiability problems questionable).

This generic propositional logic can easily be embedded into logics that are supported by many modern SMT solvers. An SMT ("satisfiability modulo theories") is a satisfiability checker for predicate logics that are obtained by expanding propositional logic with a so-called theory, for example the theory of arithmetic over the real numbers.

SMT solvers extend SAT solvers which can only check plain propositional formulas for satisfiability but cannot handle numbers or interpreted function symbols like addition for instance. This does not mean that SAT solvers cannot be used for satisfiability problems using natural numbers for example: if their domain is bounded then they can easily be modeled by propositional variables

based on the bit representation. This is not restricted to natural numbers. Even timed automata analysis which inherently relies on adding real numbers has been done with SAT solvers [12]. This requires some sort of abstraction though, for instance by approximating real numbers up to a feasible precision which enables representations with a fixed number of bits. In fact, this is nothing more than the region-based abstraction technique seen in classical algorithms on timed automata.

Here we argue that SMT solvers over theories incorporating simple real arithmetic form a promising alternative to the analysis of timed automata using SAT solvers. In effect, they offer the following advantages.

1) SMT solvers can get rid of the need for abstraction techniques. The abstraction needed to solve a problem over a continuous domain with a discrete step algorithm is removed from the timed automata analysis and entirely handled by the SMT solver. This simplifies correctness proofs on the timed automata side a lot as the proofs contained in this paper show. We emphasize the lack of need for such abstractions by calling the SMT-based analysis "exact".
2) SMT solvers are not restricted to problems of a certain complexity like SAT solvers are restricted to problems in NP (unless one accepts exponentially large encodings). In this paper we only consider a very weak fragment of logics that can be handled by modern SMT solvers: quantifier-free logic over very simple linear integer or real-valued inequalities. This is enough for the problems on timed automata considered here. However, SMT solvers offer a lot more in terms of available predicate logics. The question of which other problems on timed automata can be tackled (not necessarily "solved" because of decidability issues) using SMT solvers needs to be answered in future investigations.
3) Research in SMT solving is catching up with SAT solving. For example, incremental solving—the possibility of adding and deleting constraints after solving—is available for some SMT solvers as well. Incrementality is what makes the transition from the unbounded emptiness, universality and inclusion problem to the bounded variant a feature rather than a weakness [20].

In Sect. 5 we examine the practical feasibility of using SMT solvers for timed automata analysis by reporting on some experimental results obtained from a prototype implementation using the SMT solver Z3 [1]. We conclude the paper in Sect. 6 with ideas for future work on top of the one presented here.

Related Work. Many works also consider restricted cases in which the universality or inclusion problem becomes decidable [2, 4, 18, 19]. While we also restrict these problems (to some deterministic automata), the purpose of doing so is entirely different. We are not primarily concerned with obtaining a decidable subcase. It is just that this decidable subcase can be handled with a relatively simple translation into the SMT framework, and the complexity considerations in this paper are done in order to classify the obtained subcase accordingly. The NP-hardness lower bound shows that there is no known way of solving the problem efficiently. The NP upper bound shows that the analysis problems fit into a

relatively simple fragment of quantifier-free predicate logic over the integers and the reals.

The works that are closely related to the one presented here are probably [6, 17] where the authors extend some existing SAT-solvers in order to verify timed automata against reachability properties specified as LTL formula. This is when, in this paper we compare the (generalized) timed automata with each other to investigate properties like language inclusion and universality as well. Our approach is also different from [12, 21] where the authors present some encoding of the emptiness problem for networks of timed automata into SAT. The main difference between these and the work presented here does not lie in the extension to asynchronous networks. It is not hard to see that the encoding presented here can be extended to networks of timed automata as well. The main difference is, as argued above, the use of SMT instead of SAT solving technology. The simplification due to lack of need for explicit discretisation is imminent when comparing the correctness proofs here with the ones in there.

Timed automata as a subclass of hybrid automata can of course be dealt with using the approaches introduced in [5, 13]. These methods are designed in such a way that they can handle the non-linear constraints in hybrid automata efficiently. However, our intention is to provide a technique that is specifically designed for timed automata, whose clock constraints in a generalized form only compare the difference of two variables with a constant value, e.g. $x - y \leq 2$.

2 Timed Automata

2.1 Syntax, Semantics, Runs

A timed automaton consists of a finite state automaton together with a finite set of clocks. Clocks are non-negative real valued variables which keep track of the time delay since the last reset. A finite state automaton describes the system control states and its discrete transitions. All clocks are initially set to 0, and evolve at the same speed. A configuration of the system is given by the current control location of the automaton and the value of each clock, denoted $\langle q, v\rangle$, where q is the control location and v is the valuation function which assigns to each clock its current value. Transitions are enabled by *guards* which are constraints on clock values. The language $\Phi(X)$ of all constraints over a set of clocks X is given by

$$g ::= \mathtt{true} \mid x \lhd c \mid x - y \lhd \mathtt{c} \mid \neg g \mid g \wedge g$$

where $x, y \in X$, $\mathtt{c} \in \mathbb{N}$ and $\lhd \in \{\leq, <\}$. We will also use other Boolean and arithmetic operators that are definable using the ones above like $g_1 \vee g_2 := \neg(\neg g_1 \wedge \neg g_2)$, etc.

The definition above, as called in the literature [8, 10] *diagonal* constraint, renders timed automata more expressive than its classical variant [3, 9] where atomic constraints contain only one clock variable, e.g. $x \leq c$.

A valuation for a set X of clocks is a function $v : X \to \mathbb{R}_{\geq 0}$, assigning non-negative real values to the clocks. The satisfication of a clock constraint over

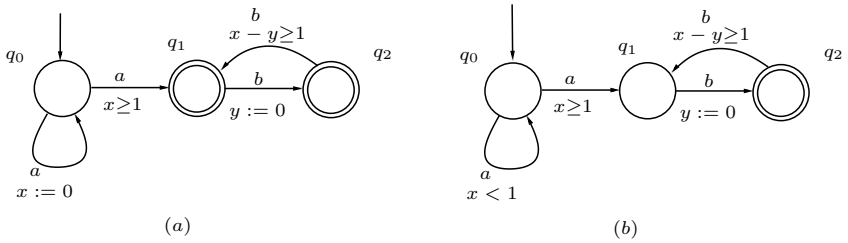

Fig. 1. a) a nondeterministic timed automaton. b) a deterministic timed automaton.

X by some valuation over X is defined straight-forwardly by induction on the structure of the constraint, e.g. $v \models x \lhd \mathsf{c}$ iff $v(x) \lhd c$ for $\lhd \in \{\leq, <\}$. A constraint g is satisfiable if there is a v s.t. $v \models g$. Otherwise it is unsatisfiable.

A (nondeterministic) *timed automaton* (TA in short) is a tuple $\langle L, L_0, L_F, \Sigma, X, E\rangle$ where L is a finite set of locations, $L_0 \subseteq L$ is a set of initial locations, $L_F \subseteq L$ is a set of final locations, Σ is a finite alphabet called *events*, X is a finite set of clock variables, and $E \subseteq L \times \Phi(X) \times \Sigma \times 2^X \times L$ is a finite set of *symbolic* transitions.

There are two kinds of concrete transitions in a timed automaton, *delay* and *discrete* ones. A delay transition lets time elapse, in which case the value of all clocks will increase accordingly. For instance, a delay transition of some non-negative time $t \in \mathbb{R}_{\geq 0}$ transforms state $\langle q, v\rangle$ into state $\langle q, v + t\rangle$, where $v + t$ assigns to each clock x the value $v(x) + t$. It is denoted $\langle q, v\rangle \xrightarrow{t} \langle q, v + t\rangle$.

A discrete transition can change the control location. There is a discrete transition with event a from state $\langle q, v\rangle$ to state $\langle q', v'\rangle$, denoted by $\langle q, v\rangle \xrightarrow{a} \langle q', v'\rangle$, if there is a symbolic transition $(q, g, a, R, q') \in E$ s.t. the clock valuation v satisfies the guard g, and v' is obtained from v by setting all clocks in R to 0.

It should be clear that two successive delay transitions with time delays t_1 and t_2 can be combined into a single delay. Furthermore, a delay transition followed by a discrete transition can be explained as a single transition based on a symbolic transition $(q, g, a, R, q') \in E$ by simply requiring that the guard g is satisfied by the clock valuation obtained from adding the delay to the current valuation: we have $\langle q, v\rangle \xrightarrow{t,a} \langle q', v'\rangle$ if $\langle q, v\rangle \xrightarrow{t} \langle q, v''\rangle$ and $\langle q, v''\rangle \xrightarrow{a} \langle q', v'\rangle$ for some (necessarily unique) valuation v''. In the following we will consider only this kind of combined delay-discrete transition to explain the operational semantics.

A *timed word* of length k over Σ is a finite sequence $(a_0, t_0), (a_1, t_0 + t_1), \ldots, (a_{k-1}, t_0 + \cdots + t_{k-1})$, where for each $0 \leq i < k:\ a_i \in \Sigma$ and $t_i \in \mathbb{R}_{\geq 0}$. The set of all timed words is denoted by $T\Sigma^*$.

Timed automata have runs on timed words just like finite automata have runs on ordinary finite words. Each event-delay pair of a timed word causes a delay-discrete transition in the automaton. As usual, runs start in initial states and are accepting if they reach a final state at the end of the input word.

Given a timed automaton $\mathcal{A} = \langle L, L_0, L_F, \Sigma, X, E\rangle$ and a timed word $w = (a_0, t_0), (a_1, t_0{+}t_1), \ldots (a_{k-1}, t_0{+}\cdots{+}t_{k-1})$ over Σ, a run of $\mathcal{A}$ on w is a sequence

$$\langle q_0, v_0\rangle \xrightarrow{t_0,a_0} \langle q_1, v_1\rangle \xrightarrow{t_1,a_1} \ldots \xrightarrow{t_{k-1},a_{k-1}} \langle q_k, v_k\rangle$$

where for all $0 \leq i < k$ there is a $(q_i, g, a_i, R, q_{i+1}) \in E$ for some g and a such that: $v_i + t_i \models g$, and for each $x \in X$ we have $v_{i+1}(x) = 0$ if $x \in R$, and $v_{i+1}(x) = v_i(x) + t_i$ otherwise. The run is accepting if additionally we have $q_0 \in L_0$ and $q_k \in L_F$. The language $L(\mathcal{A})$ of the TA $\mathcal{A}$ is the set of all timed words for which there is an accepting run of $\mathcal{A}$.

Example 1. The timed word $w = (a, 1), (a, 2.5), (b, 2.7), (b, 2.8)$ is accepted by the TA of Fig. 1 (a) which is witnessed by the run

$$\langle q_0, 0, 0\rangle \xrightarrow{1,a} \langle q_0, 0, 1\rangle \xrightarrow{1.5,a} \langle q_1, 1.5, 2.5\rangle \xrightarrow{0.2,b} \langle q_2, 1.7, 0\rangle \xrightarrow{0.1,b} \langle q_1, 1.8, 0.1\rangle$$

where $\langle q, t, t'\rangle$ represents the state $\langle q, v\rangle$ for which $v(x) = t$ and $v(y) = t'$. The language of this automaton can be described as $L(\mathcal{A}) = \{(a, t_0), \ldots (a, t_i), (b, t'_1), \ldots, (b, t'_j) \mid t_i \geq 1 \text{ and } i, j \geq 0\}$.

A TA is *deterministic*, DTA in short, when L_0 has only one element and each two transitions with same source location and same label have disjoint guards. I.e., if $(q, g, a, R, q') \in E$ and $(q, g', a, R', q'') \in E$ then $g \wedge g'$ is unsatisfiable. A DTA $\mathcal{B}$ is *complete* if for each timed word $w \in T\Sigma^*$ there is a run of $\mathcal{B}$ over w.

It is not hard to see that every DTA $\mathcal{B} = (L, q_0, L_F, \Sigma, X, E)$ can be made complete by adding a new non-final control location $q_\perp$ and transitions $(q_\perp, \texttt{true}, a, \emptyset, q_\perp)$ for every $a \in \Sigma$ as well as $(q, \bigwedge_{g \in G_{q,a}} \neg g, a, \emptyset, q_\perp)$ for every $q \in L$, and $a \in \Sigma$ where $G_{q,a} = \{g \mid \exists R. \exists q'. (q, g, a, R, q') \in E\}$. We therefore assume that DTA are always complete.

2.2 Three Decision Problems on TA and DTA

We consider three decision problems for timed automata.

1. *Non-emptiness for TA* (nEMPTY) is: given a TA $\mathcal{A}$, is $L(\mathcal{A}) \neq \emptyset$?
2. *Universality for DTA* (dUNIV) is: given a DTA $\mathcal{B}$, is $L(\mathcal{B}) = T\Sigma^*$?
3. *TA-DTA inclusion* (ndINCL) is: given a TA $\mathcal{A}$ and a DTA $\mathcal{B}$, is $L(\mathcal{A}) \subseteq L(\mathcal{B})$?

Note that more general problems like universality or inclusion between a TA and a TA are undecidable [3]. It is not difficult to see that ndINCL subsumes the two other problems: the universal language can easily be recognised by a one-state TA $\mathcal{A}_{univ}$, and the empty language can also be recognised by a one-state TA $\mathcal{A}_{empty}$ which is in fact also a DTA. Then we have $L(\mathcal{B}) \neq \emptyset$ iff $L(\mathcal{B}) \subseteq L(\mathcal{A}_)$ for any TA $\mathcal{B}$, and we have $L(\mathcal{B}) = T\Sigma^*$ iff $L(\mathcal{A}_{univ} \subseteq L(\mathcal{B})$ for any DTA $\mathcal{B}$.

Proposition 1 ([3]). *There are linear-time reductions from* nEMPTY *and* dUNIV *to* ndINCL.

Consequently, we can concentrate on ndINCL for the remainder of this paper, and any method for this problem easily induces methods for nEMPTY and dUNIV as well.

3 Bounded Decision Problems

Bounded versions restrict the length of witnesses or counterexamples to a solution by an additional parameter. The *bounded TA-DTA inclusion problem* (NDINCL$^{\leq}$) for instance is the following: given a TA $\mathcal{A}$, a DTA $\mathcal{B}$ and a $k \in \mathbb{N}$, is every timed word *of length at most* k that is accepted by $\mathcal{A}$ also accepted by $\mathcal{B}$? In that case we write $L(\mathcal{A}) \subseteq_k L(\mathcal{B})$. A *witness* (for non-inclusion) is a timed word w s.t. $|w| \leq k$, $w \in L(\mathcal{A})$ and $w \notin L(\mathcal{B})$.

The following is easy to see. It shows how the bounded TA-DTA inclusion problem can be used in order to approximate the TA-DTA inclusion problem.

Proposition 2. *Let $\mathcal{A}$ be a TA, $\mathcal{B}$ be a DTA. Then $L(\mathcal{A}) \subseteq L(\mathcal{B})$ iff for all $k \in \mathbb{N}$: $L(\mathcal{A}) \subseteq_k L(\mathcal{B})$.*

The *bounded TA emptiness problem* and the *bounded DTA universality problem* are defined in the same way by adding an input parameter k and asking for words of length at most k which are (not) in the language of the given TA, resp. DTA.

It is not hard to see that the constructions in the proof of Prop. 1 go through for the bounded versions of the three considered problems there. The bounding parameter always remains the same. We therefore state the following without an extra proof.

Theorem 1. *There are linear-time reductions from* NEMPTY$^{\leq}$ *and* DUNIV$^{\leq}$ *to* NDINCL$^{\leq}$.

It is known that the three decision problems NEMPTY, DUNIV and NDINCL are all PSPACE-complete [3, 4].[1] We will show that bounding these problems makes them computationally easier: NEMPTY$^{\leq}$ and NDINCL$^{\leq}$ are NP-complete. For the upper bound we consider a more general satisfiability problem in the next section which will also be used to obtain implementations. It can also be used to show that DUNIV$^{\leq}$ is in co-NP. Here we prove the NP-lower bounds. DUNIV$^{\leq}$ is also co-NP-hard which can be shown by a reduction from the complement of NEMPTY$^{\leq}$.

Theorem 2. NEMPTY$^{\leq}$ *and* NDINCL$^{\leq}$ *are NP-hard for a singleton alphabet and two clocks already.*

Proof. We prove the claim for NEMPTY$^{\leq}$ by a reduction from the well-known Hamiltonian path problem. Given a directed graph $G = (V, E)$ and a node $u \in V$, is there a path starting in u that visits each vertex exactly once? This problem is known to be NP-complete [15]. The result then follows for NDINCL$^{\leq}$ immediately with Thm. 1.

Let $G = (V, E)$ and $u_0 \in V$ be given. W.l.o.g. we assume $V = \{0, \ldots, |V|-1\}$. We build a TA $\mathcal{A}_G$ that has (almost) the same transition structure as G and

[1] [3] states PSPACE-completeness of DUNIV but only shows the upper bound by a reduction to NDINCL. DUNIV is also PSPACE-hard though: it is possible (but more complicated) to reduce NDINCL to DUNIV in polynomial time.

uses two clocks: x is used to enforce a certain time delay, namely exactly time 2^i in state i; and y is used to measure the overall time used in a run. We only add a single final state which is reachable whenever time $2^n - 1$ has passed.

Thus, let $\mathcal{A}_G = (V \cup \{\mathsf{fin}\}, \{u_0\}, \{\mathsf{fin}\}, \{a\}, \{x, y\}, E')$ where

$$E' \;:=\; \{(i, (x = 2^i), a, \{x\}, j) \mid (i, j) \in E\} \cup \{(i, (y = 2^{|V|} - 1), a, \emptyset, \mathsf{fin}) \mid i \in V\}$$

Finally, let $k_G := |V| + 1$. It should be clear that this construction is polynomial. Note that the representation of 2^i for instance only requires i bits. We now claim that G has a Hamiltonian path starting in u iff $L(\mathcal{A}_G)$ contains a word of length at most k_G.

"$\Rightarrow$" Let $n = |V|$ and suppose that $u_0, \ldots, u_{n-1}$ is a Hamiltonian path. It is not hard to see that $\mathcal{A}_G$ has an accepting run on the timed word $(u_0, 2^{u_0}), \ldots, (u_{n-1}, 2^{u_{n-1}}), (\mathsf{fin}, 0)$. The delay 2^{u_i} in the i-th part of this word is exactly what is needed in order to enable the guard on a transition leaving location u_i. Furthermore, since every state is visited exactly once, we have $\sum_{i=0}^{n-1} 2^{u_i} = 2^n - 1$ which enables the transition to the final state in the end.

"$\Leftarrow$". Suppose $(u_0, t_0), \ldots, (u_{n-1}, t_{n-1}), (\mathsf{fin}, t_n)$ is an accepting run of some word in $L(\mathcal{A}_G)$ and $n \leq k_G$. Since fin is only reachable by enabling the guard $y = 2^{|V|} - 1$, and no transition ever resets the clock y, we must have $\sum_{i=0}^{n-1} t_i = 2^{|V|} - 1$. Furthermore, every transition resets the clock x, and every transition out of some state i is only possible after a delay of exactly 2^i in this state. Hence, all the t_i for $i = 0, \ldots, n-1$ are powers of 2, add up to $2^{|V|} - 1$, and there are at most $|V|$ of them. This is only possible if $n - 1 = |V|$ and each of the 2^i occurs exactly once in this sum. But this is only possible if the accepting run visits each location exactly once, i.e. forms a Hamiltonian path in the underlying graph G. □

4 Incremental Encodings of Some Decision Problems

4.1 A Generic Constraint Logic

We consider a generic propositional logic which can have constraints over integer- and real-valued variables as literals, and present an encoding of the bounded inclusion problems into this logic. Decidability and implementability of the bounded inclusion problem can then be tackled by translating this logic into some known and specialised formalism.

Let $\mathcal{V}^{\mathbb{Z}}$ and $\mathcal{V}^{\mathbb{R}}$ be two disjoint sets of natural and real-valued variables. We use two different fonts in order to distinguish variables and constant values: $x, y, \ldots$ are variables, $\mathsf{b}, \mathsf{c}, \ldots$ are constants. We also assume that it will always be clear from the context what the type of a given variable is.

An *integer constraint* is of the form $\mathsf{b} \cdot x \leq \mathsf{c}$ where $x \in \mathcal{V}^{\mathbb{Z}}$, $\mathsf{b} \in \{1, -1\}$, and $\mathsf{c} \in \mathbb{Z}$. A *real constraint* is of the form $\sum_{i=1}^{m} \mathsf{b}_i \cdot x_i \lhd \mathsf{c}$ where $\lhd \in \{<, \leq\}$, $\mathsf{b}_i \in \{0, 1\}$, $x_i \in \mathcal{V}^{\mathbb{R}}$ for all $i = 1, \ldots, m$, and $\mathsf{c} \in \mathbb{R}$.

Formulas of the *generic constraint logic* (GCL) are simply propositional formulas over atomic constraints.

$$\varphi \;::= C \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi$$

where C is one of the constraints above. It should be clear that, with Boolean operators at hand, many other constraints become definable, for instance $x = \mathsf{c} := x \leq \mathsf{c} \wedge -x \leq -\mathsf{c}$, $x < \mathsf{c} := x \leq (\mathsf{c} - 1)$ in case of an integer constraint, etc.

We write $|\varphi|$ to denote the size of φ measured in terms of occurrences of logical, relational and arithmetic symbols as well as size of a representation for the constants, for instance encoded in binary.

A variable *assignment* is a $\vartheta = (\vartheta^{\mathbb{Z}}, \vartheta^{\mathbb{R}})$ s.t. $\vartheta^{\mathcal{X}} : \mathcal{V}^{\mathcal{X}} \to \mathcal{X}$ for $\mathcal{X} \in \{\mathbb{Z}, \mathbb{R}\}$. Hence, it assigns integer, resp. real values to the integer, resp. real variables. The *truth* value of a formula φ under an assignment ϑ is defined straight-forwardly by evaluating the constraints and subformulas under the usual rules for $<$, $\leq$, $+$, $-$, $\wedge$, $\vee$, and $\neg$. We write $\vartheta \models \varphi$ to state that φ evaluates to *true* under the assignment ϑ.

A formula is *satisfiable* if there is a ϑ s.t. $\vartheta \models \varphi$. Clearly, such a ϑ only needs to be defined on the variables that actually occur in φ which allows us to assume that these assignments have finite domains. Two formulas are equivalent, written $\varphi \equiv \psi$, if for all ϑ we have $\vartheta \models \varphi$ iff $\vartheta \models \psi$.

A formula is *positive* it it does not contain any occurrences of the negation operator $\neg$. Note that this is stronger than the well-known concept of positive normal form which still would allow negation operators. The following is easy to prove using the duality between $\leq$ and $<$.

Lemma 1. *For every φ there is a positive ψ s.t. $\varphi \equiv \psi$ and $|\psi| \leq 2|\varphi|$.*

Theorem 3. *The satisfiability problem for* GCL *is NP-complete.*

Proof. NP-hardness is a simple consequence of the fact that integer variables can be restricted to the domain $\{0, 1\}$ using the available constraints and thus model Boolean variables. It is therefore possible to embed the NP-hard satisfiability problem for propositional logic [11] into the satisfiability problem for GCL.

For the upper bound we describe a nondeterministic algorithm. Let φ be a GCL formula. According to Lemma 1 we can assume φ to be positive. First guess a truth value for each atomic constraint in it. Clearly, there are at most $|\varphi|$ many, and the truth values can be propagated up in the formula in polynomial time to see whether the result is true. Next we need to decide whether there are variable assignments that fulfil the atomic constraints which are guessed to be true. Note that the integer constraints and the real constraints are independent of each other because they do not share any variables. Hence, they can be dealt with separately.

Solving a set of integer constraints of the form $\mathsf{b} \cdot x \leq \mathsf{c}$ can be done in polynomial time using interval arithmetic. Note that the solution to each such constraint is given by an interval $(-\infty, \mathsf{c}]$ if $\mathsf{b} = 1$, and $[\mathsf{c}, \infty)$ if $\mathsf{b} = -1$. Computing intersections of such constraints is easy.

In order to deal with real-valued constraints we use the linear programming optimisation problem. First of all, we add a new variable y and replace every strict inequality $\sum_{i=1}^{m} \mathsf{b}_i \cdot x_i < \mathsf{c}$ by $y + \sum_{i=1}^{m} \mathsf{b}_i \cdot x_i \leq \mathsf{c}$. This creates a linear program with cost function y, i.e. we are looking for a solution to these constraints that maximises the value of y. In order to ensure that there is a maximal value

we also add a constraint like $y \leq 1$. It is known that the linear programming optimisation problem can be solved in deterministic polynomial time when the coefficients are also taken into account as part of the input [16]. Finally, the result of the optimisation problem needs to be translated back into a result of the original system of constraints. If the system is not solvable then so is the original one. If the maximal feasible value for y is at most 0 then the original system is also not solvable. If the maximal feasible value for y is strictly positive, then there is also a solution to the original system. □

4.2 The Incremental Encoding

We fix a TA $\mathcal{A} = (L^{\mathcal{A}}, L_0^{\mathcal{A}}, L_F^{\mathcal{A}}, \Sigma, X^{\mathcal{A}}, E^{\mathcal{A}})$, a DTA $\mathcal{B} = (L^{\mathcal{B}}, q_0^{\mathcal{B}}, L_F^{\mathcal{B}}, \Sigma, X^{\mathcal{B}}, E^{\mathcal{B}})$ and a natural number k for the remainder of this section. We will define a formula $\varphi_k^{\mathcal{A},\mathcal{B}}$ that is satisfiable iff $L(\mathcal{A}) \not\subseteq_k L(\mathcal{B})$. Moreover, its size will be linear in the sizes of $|\mathcal{A}|$, $|\mathcal{B}|$, and in k. Note that this is the value of k, not the number of bits needed to represent it.

In order to make the presentation of $\varphi_k^{\mathcal{A},\mathcal{B}}$ as intuitive as possible, we divide it into several parts and present it statically first. At the end we will discuss how to make it incremental in the sense that the formula $\varphi_{k+1}^{\mathcal{A},\mathcal{B}}$ can be obtained from $\varphi_k^{\mathcal{A},\mathcal{B}}$ by deleting and adding single components.

The first part states that the witnessing word is well-formed. It uses natural number variables $a_0, \ldots, a_{k-1}$ to encode the events in a witness, as well as real-valued variables $t_0, \ldots, t_{k-1}$ for the time delays of such a witness. W.l.o.g. we assume the alphabet to be $\{1, \ldots, |\Sigma|\}$. Furthermore, remember that the formula to be built should state that there is a word of length *at most* k with some property. In order to capture words of length $< k$ as well, we take an additional value, say 0, and represent such a shorter timed word w by $w(0,t_1)\ldots(0,t_m)$ where $m = k - |w|$ and the t_i are arbitrary values. The following formula then states well-formedness of such a timed word.

$$word_k \;:=\; \Big(\bigwedge_{i=0}^{k-1} 0 \leq t_i\Big) \;\wedge\; \Big(\bigwedge_{i=0}^{k-1} 0 \leq a_i \wedge a_i < |\Sigma|\Big) \;\wedge\; \Big(\bigwedge_{i=0}^{k-2} a_i = 0 \;\rightarrow\; a_{i+1} = 0\Big)$$

Next we formalise that the timed word w represented by values for those variables is accepted by $\mathcal{A}$. We use $k+1$ integer variables $\ell_0, \ldots, \ell_k$ to represent the locations in an accepting run of $\mathcal{A}$ on w. W.l.o.g. we can assume that $L^{\mathcal{A}} = \{0, \ldots, |L^{\mathcal{A}}| - 1\}$. We also use $(k+1) \cdot |X^{\mathcal{A}}|$ many real-valued variables v_i^c where $c \in X$ and $0 \leq i \leq k\}$, in order to represent the values of the clocks in each state of this accepting run. We first state that the run is well-formed in the sense that each pair of adjacent states is connected by a transition that is possible in $\mathcal{A}$ unless the event symbol at that position is 0.

$$run_k^{\mathcal{A}} \;:=\; \bigwedge_{i=0}^{k-1} \bigwedge_{\mathsf{q} \in L^{\mathcal{A}}} \ell_i = \mathsf{q} \wedge \neg(a_i = 0) \;\rightarrow\; \bigvee_{(\mathsf{q},g,\mathsf{a},R,\mathsf{q}') \in E^{\mathcal{A}}} sat_i(g) \;\wedge\; a_i = \mathsf{a} \;\wedge\; progress_i(R) \;\wedge\; \ell_{i+1} = \mathsf{q}'$$

where $sat_i(g)$ states that the guard g is satisfied by the clock valuation after the i-th delay has passed. It is defined by induction on the structure of g:

$$\begin{array}{lcllcl} sat_i(x \leq \mathsf{c}) & := & v_i^x + t_i \leq \mathsf{c} & \qquad sat_i(\neg g) & := & \neg sat_i(g) \\ sat_i(x < \mathsf{c}) & := & v_i^x + t_i < \mathsf{c} & sat_i(g_1 \wedge g_2) & := & sat_i(g_1) \wedge sat_i(g_2) \end{array}$$

Furthermore, $progress_i(R)$ formalises the progression of time. In particular, the clock valuation at the next moment results from delaying at the current moment and resetting clocks afterwards.

$$progress_i(R) \;:=\; \big(\bigwedge_{x \in R} v_{i+1}^x = 0\big) \;\wedge\; \big(\bigwedge_{x \in X^{\mathcal{A}} \setminus R} v_{i+1}^x = v_i^x + t_i\big)$$

Finally, we need to say that the run is accepting, i.e. starts in an initial location with all the clocks set to 0 at that moment. A final location needs to be reached at that point when the next event symbol either does not exist anymore (i.e. the k-the location in the run) or is 0 (for words of length $< k$).

$$\begin{aligned} acc_k^{\mathcal{A}} \;:=\; & \big(\bigwedge_{x \in X^{\mathcal{A}}} v_0^x = 0\big) \;\wedge\; \big(\bigvee_{\mathsf{q} \in L_0^{\mathcal{A}}} \ell_0 = \mathsf{q}\big) \;\wedge\; \big(\bigwedge_{i=0}^{k-1} a_i = 0 \;\rightarrow\; \bigvee_{\mathsf{q} \in L_F^{\mathcal{A}}} \ell_i = \mathsf{q}\big) \;\wedge \\ & \big(\neg(a_{k-1} = 0) \;\rightarrow\; \bigvee_{\mathsf{q} \in L_F^{\mathcal{A}}} \ell_k = \mathsf{q}\big) \end{aligned}$$

Next we need to state that $\mathcal{B}$ does not accept the timed word given by the values for the a_i and t_i variables. In general, this is a universal statement over all runs of the automaton but, since it is assumed to be deterministic and complete, it can equivalently be rephrased as "there is a run that is not accepting". Hence, the constraints $run_k^{\mathcal{B}}$ are formed exactly in the same way as for $run_k^{\mathcal{A}}$ but using fresh variables $\ell'_0, \ldots, \ell'_k$ for the locations in the run of $\mathcal{B}$. W.l.o.g. we can assume $X^{\mathcal{A}} \cap X^{\mathcal{B}} = \emptyset$, i.e. the two automata use different clocks. We then have variables v_i^x as above for every clock $x \in X^{\mathcal{B}}$ and every $i = 0, \ldots, k$.

We also define a constraint $rej_k^{\mathcal{B}}$ similar to $acc_k^{\mathcal{A}}$ now stating that the run of $\mathcal{B}$ given by the valuations of the ℓ'_i and v_i^x variables is rejecting.

$$rej_k^{\mathcal{B}} \;:=\; \big(\bigwedge_{x \in X^{\mathcal{B}}} v_0^x = 0\big) \wedge \ell_0 = \mathsf{q}_0^{\mathcal{B}} \wedge \big(\bigwedge_{i=0}^{k-1} \bigwedge_{\mathsf{q} \in L_F^{\mathcal{A}}} a_i = 0 \rightarrow \ell'_i \neq \mathsf{q}\big)$$

Finally, we define $\varphi_k^{\mathcal{A},\mathcal{B}} := word_k \wedge run_k^{\mathcal{A}} \wedge acc_k^{\mathcal{A}} \wedge run_k^{\mathcal{B}} \wedge rej_k^{\mathcal{B}}$. There are two important aspects to note about $\varphi_k^{\mathcal{A},\mathcal{B}}$: it is small, i.e. linear in its three parameters, and it characterises the bounded TA-DTA inclusion problem.

Proposition 3. $|\varphi_k^{\mathcal{A},\mathcal{B}}| = \mathcal{O}(k \cdot (|\mathcal{A}| + |\mathcal{B}|))$.

Theorem 4. *$\varphi_k^{\mathcal{A},\mathcal{B}}$ is satisfiable iff $L(\mathcal{A}) \not\subseteq_k L(\mathcal{B})$.*

Proof. "⇐" Suppose $L(\mathcal{A}) \not\subseteq_k L(\mathcal{B})$. Then there is a $w = (\mathsf{a}_0, \mathsf{t}_0), \ldots, (\mathsf{a}_{n-1}, \mathsf{t}_{n-1}) \in T\Sigma^*$ s.t. $n \leq k$ and $w \in L(\mathcal{A})$ and $w \notin L(\mathcal{B})$. Then there is an accepting run $\langle q_0, v_0\rangle, \ldots, \langle q_n, v_n\rangle$ of $\mathcal{A}$ on w. Furthermore, the unique run $\langle q'_0, v'_0\rangle, \ldots, \langle q'_n, v'_n\rangle$ of $\mathcal{B}$ on w is not accepting, i.e. it does not end in a final location.

It is now routine to check that the variable assignment $\vartheta = (\vartheta^{\mathbb{Z}}, \vartheta^{\mathbb{R}})$ satisfies $\varphi_k^{\mathcal{A},\mathcal{B}}$, where

$$\vartheta^{\mathbb{Z}}(a_i) = \begin{cases} \mathsf{a}_i & , \text{ if } i < n \\ 0 & , \text{ if } n \leq i < k \end{cases} \qquad \vartheta^{\mathbb{R}}(t_i) = \begin{cases} \mathsf{t}_i & , \text{ if } i < n \\ 0 & , \text{ if } n \leq i < k \end{cases}$$

$$\vartheta^{\mathbb{Z}}(\ell_i) = \begin{cases} \mathsf{q}_i & , \text{ if } i \leq n \\ \mathsf{q}_{n-1} & , \text{ if } n < i \leq k \end{cases} \qquad \vartheta^{\mathbb{R}}(v_i^x) = \begin{cases} v_i(x) & , \text{ if } i \leq n \\ 0 & , \text{ otherwise} \end{cases}$$

$$\vartheta^{\mathbb{Z}}(\ell'_i) = \begin{cases} \mathsf{q}'_i & , \text{ if } i \leq n \\ \mathsf{q}'_{n-1} & , \text{ if } n < i \leq k \end{cases}$$

"⇒" Suppose $\varphi_k^{\mathcal{A},\mathcal{B}}$ has a satisfying variable assignment $\vartheta = (\vartheta^{\mathbb{Z}}, \vartheta^{\mathbb{R}})$. The $\vartheta^{\mathbb{Z}}$-values of $a_0, \ldots, a_{k-1}$ and the $\vartheta^{\mathbb{R}}$-values of $t_0, \ldots, t_{k-1}$ encode a timed word over the alphabet $\Sigma \cup \{0\}$. Because of $word_k$ this is in fact a timed word in $T(\Sigma^*0^*)$. Hence, there is an $n \leq k$, s.t. $w' := (\vartheta^{\mathbb{Z}}(a_0), \vartheta^{\mathbb{R}}(t_0)), \ldots, (\vartheta^{\mathbb{Z}}(a_{n-1}), \vartheta^{\mathbb{R}}(t_{n-1})) \in T\Sigma^*$. Furthermore, consider the sequence $\langle \vartheta^{\mathbb{Z}}(\ell_0), v_0\rangle, \ldots, \langle \vartheta^{\mathbb{Z}}(\ell_n), v_n\rangle$ where $v_i(x) = \vartheta^{\mathbb{R}}(v_i^x)$ for every $i = 0, \ldots, n$ and every $x \in X^{\mathcal{A}}$. Because of $run_k^{\mathcal{A}}$ this forms a run of $\mathcal{A}$ on w' which can be extended to a run on w by simply repeating the last pair in this sequence $k - n$ times. Because of $acc_k^{\mathcal{A}}$ this run is accepting; it starts with all clocks set to 0 and ends in a final location.

In the same way one can see that $run_k^{\mathcal{B}}$ and $rej_k^{\mathcal{B}}$ enforce the unique run of $\mathcal{B}$ on w to be rejecting. Hence, we have $L(\mathcal{A}) \not\subseteq_k L(\mathcal{B})$. □

It should be clear that similar formulas characterising the bounded TA emptiness or the bounded DTA universality problem can easily be defined as well. In fact, it is not necessary to carry out the cnstructions sketched for Prop. 1. Instead, for bounded emptiness it suffices to remove the constraints involving $\mathcal{B}$ in $\varphi_k^{\mathcal{A},\mathcal{B}}$; for bounded universality it suffices to remove those involving $\mathcal{A}$.

Also note that the encoding can easily be made *incremental* in the sense that the formula $\varphi_{k+1}^{\mathcal{A},\mathcal{B}}$ can be obtained by certain operations on $\varphi_k^{\mathcal{A},\mathcal{B}}$. In detail:

1. Add the conjuncts $0 \leq t_k$, $0 \leq a_k$, $a_k \leq |\Sigma|$, $a_{k-1} = 0 \rightarrow a_k \rightarrow 0$ to $word_k$.
2. Extend the conjunctions in $run_k^{\mathcal{A}}$ and $run_k^{\mathcal{B}}$ by one to the range $i = 0, \ldots, k$.
3. Remove the last conjunct from $acc_k^{\mathcal{A}}$, extend the conjunction before that to the range $i = 0, \ldots, k$, and add the conjunct $\neg(a_k = 0) \rightarrow \bigvee_{\mathsf{q} \in L_F^{\mathcal{A}}} \ell_{k+1} = \mathsf{q}$;

Finally, the translation gives us an upper complexity bound matching the lower bound in Thm. 2. It follows immediately from Thm. 4 and Thm. 3.

Corollary 1. nEMPTY$^{\leq}$ *and* ndINCL$^{\leq}$ *are NP-complete for a unarily encoded bounding parameter* k*. The lower bound requires at least two clocks. Also,* dUNIV$^{\leq}$ *is in co-NP.*

4.3 Translations into Other Formalisms

The extension of the propositional logic proposed above is extensive enough for our purpose here, which is the encoding of bounded representations of the three decidability problems under consideration. A natural question that arises is how this can help to solve these problems in practice.

We refer to the SMT-LIB [7] framework, a standardisation for propositional and predicate logics over various domains. It contains well-documented input languages, including theories of difference logic over integers and reals and their combinations. It is not hard to see that the propositional logic defined above can esily be embedded into some of these formalisms, for instance the combination of `QF_IDL` (difference logic over integers) and `QF_RDL` (difference logic over the reals).

Furthermore, most SMT solvers nowadays comply with the SMT-LIB standards in the sense that it is clearly stated which theories they handle.

5 Implementation and Experimental Results

The approach described above is realised in a prototypical implementation done in OCaml. It creates constraints according to some pre-defined families of benchmarks, and uses the SMT solver Z3 in order to solve the bounded emptiness, inclusion or universality problem as defined above in an incremental way.

All tests reported here were run on a compute server equipped with 16 Intel Xeon cores at 1.87GHz and 256GB main memory. Note that neither the reduction nor the SMT solver support parallelism, thus each test only occupies one core.

Here we report on three benchmarks testing different aspects of this approach, all phrased as some series of emptiness problems.

Fischer's Mutex Protocol. The first benchmarking family models Fischer's protocol for mutual exclusion between n processes communicating via one shared variable as one timed-automaton. Note that the state-spaces of the TA are exponential in n. Their languages consist of all runs of the asynchronous product of these n processes which end in a state that has at least two processes in the critical section. Thus, this is a classical safety verification problem. In this

Table 1. Experimental analysis of emptiness for timed automata

Fischer's Mutex protocol

n	size	create	solve
2	4238	0.01	0.02
3	33750	0.05	0.93
4	230760	0.20	20.86
5	1433948	1.16	6511.16

Diagonal Constraints

s	length	size
1	1141	197422
5	1040	179096
10	1160	198972
20	1299	224922
50	1499	259522
100	1699	294122

Exponential Witnesses

n	size	create	solve	c
2	1136	0.01	0.00	5
3	3230	0.02	0.02	8
4	8415	0.05	0.27	14
6	37121	0.32	42.77	38
8	179827	89.54	7050.78	134

model, the delay times are set such that mutual exclusion is not guaranteed for otherwise the languages would be empty.

Table 1 presents experimental data showing the size of the resulting formula in number of logical operators in it, the time it takes to create it and the time it takes to solve it in seconds. This benchmark is created in order to stress the point that SMT solvers can check relatively large formulas for satisfiability.

Diagonal Constraints. There is an example of an 8-state TA using diagonal constraints in [9]. Its language is empty. The example shows that having diagonal constraints in the timed automaton makes the classical forward analysis approach, implemented in most timed automata verification tools like KRONOS and UPPAAL, unsound.

We use this benchmark to demonstrate that the approach in this paper can easily deal with diagonal constraints, but also to examine the effect of varying the step width s in which larger and larger ranges of lengths of timed words are considered. Thus, a value of $s = 7$ for instance means checking for witnesses of length $\{0, \ldots, 6\}$ first, then of length $\{7, \ldots, 13\}$, etc.

Table 1 presents the maximal range length that could be examined within a run time of one hour, as well as the size of the formula at the end. Thus, the implementation showed for instance within one hour, that the TA under consideration does not accept any word of length at most 1699. The interpretation of the data is: larger incremental steps can save some time.

Exponential Witnesses. As a last benchmark we consider a family of TA with $n + 1$ states s.t. their language is non-empty but the shortest word that they accept is of length $2^n - 1 + n$. This can be enforced with a trick that is similar to the one used in the NP-hardness proof in Thm. 2. However, a naïve approach would cause a quadratic number of transitions. Here we use a slightly randomised model of such TA which only needs a linear number of transition.

Table 1 shows size of formula, creation and solving time in seconds as well as number c of calls to the SMT solver before a witness is being found. The step width for ranges in the incremental analysis is fixed, thus this number grows exponentially. Note that the reported solving time is the accumulated time of c calls. This benchmark shows that the SMT solver can still find relatively long witnessing words, here for instance of length $263 = 2^8 - 1 + 8$. Note that the numbers for formula sizes in the table clearly do not grow linearly. This is because they represent formulas for exponentially differing ranges of word length, namely those of the shortest words witnessing non-emptiness.

6 Further Work

There is a vast possibility for follow-up work. For starters, the encoding should be extended to richer models, for instance TA with state invariants which is very easy. It is also worth considering asynchronous networks of TA. It is easy to see that the emptiness problem can be encoded in the same way, but the two other

problems may not be capturable as concurrency introduces nondeterminism. Then there is the question of extending this to TA on infinite timed words.

There are two more extensions which show the vast potential that the use of SMT solvers has as opposed to using SAT solvers: the latter are restricted to problems in NP, the former are hardly restricted at all. Thus, far richer clock constraint languages (for instance with all arithmetic operations) can easily be handled by SMT solvers. At last, a very promising extension is the consideration of richer logical (rather than just arithmetic) formalisms on the SMT side, for instance logics with quantification. It remains to be seen which problems on timed automata can also be encoded this way. Note that this is not restricted to decidable problems since there are SMT logics which are undecidable in general. Still, using SMT solvers for these problems may offer a practically viable approach which may just not be complete or not always terminating.

References

1. http://research.microsoft.com/en-us/um/redmond/projects/z3/
2. Abdulla, P.A., Deneux, J., Ouaknine, J., Quaas, K., Worrell, J.: Universality Analysis for One-Clock Timed Automata. Fundamenta Informaticae, 89 (2008)
3. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theo. Comp. Sci. (1994)
4. Alur, R., Madhusudan, P.: Decision Problems for Timed Automata: A Survey. In: SFM School (2004)
5. Audemard, G., Bozzano, M., Cimatti, A., Sebastiani, R.: Verifying industrial hybrid systems with mathsat. ENTCS 119(2) (2005)
6. Audemard, G., Cimatti, A., Kornilowicz, A., Sebastiani, R.: Bounded model checking for timed systems. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE 2002. LNCS, vol. 2529, Springer, Heidelberg (2002)
7. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. Technical report (2010), http://www.SMT-LIB.org
8. Bérard, B., Petit, A., Diekert, V., Gastin, P.: Characterization of the expressive power of silent transitions in timed automata. Fundam. Inform. 36(2-3) (1998)
9. Bouyer, P.: Untameable timed automata! In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 620–631. Springer, Heidelberg (2003)
10. Bouyer, P., Laroussinie, F., Reynier, P.-A.: Diagonal constraints in timed automata: Forward analysis of timed systems. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 112–126. Springer, Heidelberg (2005)
11. Cook, S.A.: The complexity of theorem-proving procedures. In: 3rd Annual ACM Symposium on Theory of Computing (STOC), pp. 151–158 (1971)
12. de Moura, L.M., Rueß, H., Sorea, M.: Bounded model checking and induction: From refutation to verification (extended abstract, category a). In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 14–26. Springer, Heidelberg (2003)
13. Fränzle, M., Herde, C.: HySAT: An efficient proof engine for bounded model checking of hybrid systems. Formal Methods in System Design 30(3) (2007)
14. Henzinger, T.A., Manna, Z., Pnueli, A.: What good are digital clocks? In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, Springer, Heidelberg (1992)
15. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103 (1972)

16. Khachiyan, L.G.: A polynomial algorithm in linear programming. Doklady Akademiia Nauk SSSR, 224 (1979)
17. Niebert, P., Mahfoudh, M., Asarin, E., Bozga, M., Maler, O., Jain, N.: Verification of timed automata via satisfiability checking. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 225–243. Springer, Heidelberg (2002)
18. Ouaknine, J., Worrell, J.: Revisiting digitization, robustness, and decidability for timed automata. In: LICS (2003)
19. Ouaknine, J., Worrell, J.: On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap. In: LICS (2004)
20. Strichman, O.: Pruning techniques for the SAT-based bounded model checking problem. In: Margaria, T., Melham, T.F. (eds.) CHARME 2001. LNCS, vol. 2144, pp. 58–70. Springer, Heidelberg (2001)
21. Zbrzezny, A.: SAT-based Reachability Checking for Timed Automata with Diagonal Constraints. Fundam. Inform. 67(1-3), 303–322 (2005)