Electronic Communications of the EASST
Volume 076 (2019)

Automated Verification of Critical Systems 2018
(AVoCS 2018)

Backward Reachability Analysis for Timed Automata with Data Variables

Rebeka Farkas, Tamás Tóth, Ákos Hajdu, András Vörös

20 pages

Guest Editors: David Pichardie, Mihaela Sighireanu
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Backward Reachability Analysis for Timed Automata with Data Variables

Rebeka Farkas[1], **Tamás Tóth**[2*], **Ákos Hajdu**[3], **András Vörös**[4]

[1] farkasr@mit.bme.hu [2] totht@mit.bme.hu [3] hajdua@mit.bme.hu [4] vori@mit.bme.hu
Department of Measurement and Information Systems,
Budapest University of Technology and Economics, Hungary

[134] MTA-BME Lendület Cyber-Physical Systems Research Group

Abstract: Efficient techniques for reachability analysis of timed automata are zone-based methods that explore the reachable state space from the initial state, and SMT-based methods that perform backward search from the target states. It is also possible to perform backward exploration based on zones, but calculating predecessor states for systems with data variables is computationally expensive, prohibiting the successful application of this approach so far. In this paper we overcome this limitation by combining zone-based backward exploration with the weakest precondition operation for data variables. This combination allows us to handle diagonal constraints efficiently as opposed to zone-based forward search where most approaches require additional operations to ensure correctness. We demonstrate the applicability and compare the efficiency of the algorithm to existing forward exploration approaches by measurements performed on industrial case studies. Although the large number of states often prevents successful verification, we show that data variables can be efficienlty handled by the weakest precondition operation. This way our new approach complements existing techniques.

Keywords: timed automata, reachability analysis, backward exploration, weakest precondition

1 Introduction

The ubiquity of real-time safety-critical systems makes it desirable to model and verify time-dependent behaviour. One of the most common formalisms for this purpose is the timed automaton, introduced by Alur and Dill [AD91] which extends the finite automaton with real-valued variables called *clock variables* that represent the elapse of time. This makes it a suitable formalism for modelling time-dependent behaviour of systems such as communication protocols [RSV10] and logical circuits with propagation delay [MP95].

State reachability has a prominent role in verification. Most reachability analysis algorithms perform *forward search*: they check if the set of states reachable from the initial state contains the target state. *Backward search* can also be used for checking reachability – in this case the

* This work was partially supported by Gedeon Richter's Talentum Foundation (Gyömrői út 19-21, 1103 Budapest, Hungary).

algorithm explores set of states from where the target state is reachable and checks if it contains the initial state [Mit07]. The motivation behind backward search is that when the target state is unreachable, the state space explored from the initial state is disjunct from the one explored backwards from the target states and the latter might be smaller. The combination of backward and forward approaches (often referred to as *bidirectional search*) can be more efficient than either of the original approaches [VGS13].

Reachability for timed automata is decidable, but the complexity is exponential in the number of clock variables [AD91]. The most efficient techniques for deciding reachability either rely on SMT solvers or abstract state space representations. The key idea of SMT-based techniques is to transform the system into a set of constraints that is satisfiable if the target state is reachable, and then give it to an SMT solver. Although this technique is efficient for other domains and there is an efficient first-order theory for the timed domain, called *difference logic* [BM07], in practice solver-based techniques are usually more efficient for other timed formalisms, such as the timeout automaton [DS$^+$04]. As for timed automata, the most efficient algorithms rely on the *zone* abstract domain [YPD94] that is usually used for (forward) state space exploration [Bou05].

While it is possible to perform zone-based backward state space exploration of timed automata, the forward approach is preferred. The general reason behind this is that in practice extensions of timed automata are used, such as *timed automata with data variables*, and it is expensive to calculate predecessor states for systems with data variables [BLR05, Bou05]. On the other hand, zone-based backward exploration of timed automata would be desired, because for a class of timed automata called *timed automata with diagonal constraints* backward exploration is the only zone-based method that does not introduce an additional exponential factor to the complexity [BLR05, BC05].

In this paper we propose an algorithm for backward reachability analysis of timed automata that combines zone abstraction for clock variables with the *weakest precondition* operation for data variables. Using the operation we can explore the state space of the data variables backwards. We also compare the applicability and the efficiency of zone-based backward and forward exploration by measurements performed on the *XtaBenchmarkSuite* presented in [FB18].

We show that data variables can be handled by the weakest precondition operation but the large number of target states often prevents successful verification. We propose some possible improvements and other ideas about the applicability of zone-based backward exploration algorithm for the verification of timed automata.

The paper is organized as follows. Section 2 presents the related work, Section 3 provides some theoretical background on the reachability analysis of timed automata, our algorithm is explained in Section 4. Section 5 presents and evaluates the performed measurements, and finally Section 6 concludes the paper.

2 Related work

To the best of our knowledge there is no zone-based backward reachability analysis algorithm for timed automata with data variables in the literature. However, many zone-based algorithms have been proposed for forward exploration [HNSY92] as well as SMT-based methods that perform backward search [MPS11]. Such algorithms often fail to support extensions of timed automata, such as *diagonal constraints* or *data variables*. Our algorithm is applicable in both cases.

Zone-based state space exploration Zone-based techniques explore an *abstract reachability graph* whose nodes contain a zone-based abstraction of a set of reachable states.

The basics of zone-based algorithms and the zone operations were introduced in [YPD94] where they proposed the backward exploration of timed automata (without data variables) that provides the timed part of our algorithm. Zone-based forward exploration algorithms were also developed [BY03] and preferred for efficiency reasons – despite the fact that they are not sound for timed automata with diagonal constraints [Bou03].

Many optimizations of zone abstraction have been defined. An optimal zone-based abstraction – that is, the weakest abstraction that is still safe – has been presented in [HSW13]. The optimization was extended to timed automata with diagonal constraints in [GMS18]. Other approaches to apply zone-based forward exploration for timed automata with data variables were discussed in [BLR05].

Backward exploration Generally, SMT-based methods can operate both forwards and backwards, and they can be applied to systems with data variables as well as timed automata with diagonal constraints. However, despite the many advantages, in practice SMT-based methods are less efficient for timed automata, than zone-based techniques. One of the most efficient SMT-based methods for timed automata is presented in [MPS11], where backward exploration is used, however, before exploration the timed automaton is translated to another formalism called *finite state machine with time*. The transformation requires lower and upper bounds on the integer variables.

Besides state reachability, backward exploration is often used for model checking CTL properties on timed automata. The tool Kronos [Yov97] performs zone-based backward exploration for model checking timed automata against TCTL properties. However, Kronos does not support data variables.

Algorithms for timed automata with data variables Independent of the algorithm, the values of data variables can be encoded to the locations of timed automata as a preprocessing operation. In addition, forward exploration algorithms can be applied for extended timed automata by calculating the values of data variables on the fly. However, these approaches only allow explicit state space exploration over data variables.

On the other hand, in case of approaches with abstractions over data variables the possible operations are often restricted to ensure the abstraction is safe. In [IW14] the restrictions ensure integer variables can be encoded in zones (just like clocks). In [FS01] they use the concept of *well-structured transition systems*, that require *well-quasi orderings* to verify several formalisms, including timed systems.

There are very few methods applicable to extended timed automata without restrictions that do not calculate the values of data variables explicitly. In [HRSY14] where they use Horn-clauses for verification of timed automata with data variables. In [TM18] the visibilities of data variables are controlled based on interpolations.

3 Preliminaries

This section presents the important aspects of modeling timed systems, the extensions of timed automata used in practice, as well as the basic operations of zone-based state space exploration. The weakest precondition operation is also defined.

3.1 Timed automata

3.1.1 Basic definitions

Clock variables (or *clocks*, for short) are introduced to represent the elapse of time. The set of clock variables is denoted by $\mathscr{C}$.

The most commonly used type of predicates over clocks is *clock constraints*.

Definition 1 A clock constraint is a conjunction of *atomic clock constraints*, that can either be *simple constraints* of the form $x \sim n$ or *diagonal constraints* of the form $x - y \sim n$, where $x, y \in \mathscr{C}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. The set of clock constraints is denoted by $\mathscr{B}(\mathscr{C})$.

In other words clock constraints define upper and lower bounds on the values of clocks and the differences of clocks.

A *timed automaton* extends a finite automaton with clock variables.

Definition 2 A timed automaton $\mathscr{A}$ is a tuple $\langle L, l_0, E \rangle$ where

- L is the set of locations (or *control states*),
- $l_0 \in L$ is the initial location,
- $E \in L \times \mathscr{B}(\mathscr{C}) \times 2^{\mathscr{C}} \times L$ is the set of edges. An edge is defined by the source location, the guard (represented by a clock constraint), the set of clocks to reset and the target location.

Remark 1 *It is usual to allow* invariants *on the locations of the automaton. In this paper we omit them to keep the explanations simple. However, the presented algorithms can be adjusted to handle invariants.*

In the literature many algorithms are only defined for a subclass of timed automata called *diagonal-free timed automata*.

Definition 3 A diagonal-free timed automaton is a timed automaton where all atomic clock constraints are simple constraints.

It is possible to transform a timed automaton containing diagonal constraints to a diagonal-free timed automaton – i.e. the expressive power of diagonal-free timed automata is the same as that of timed automata. However, the number of locations in the resulting automaton is exponential in the number of diagonal constraints [BC05].

3.1.2 Operational semantics

The values of the variables at a given moment is described by a valuation.

Definition 4 Let us denote by $\mathrm{dom}(v)$ the set of possible values for a variable v, i.e. the *domain* of v. Given a set of variables V and the corresponding domains a *valuation* is a function $v : V \to \cup_{v \in V} \mathrm{dom}(v)$ that assigns a value for each variable $v \in V$ such that $v(v_i) \in \mathrm{dom}(v_i)$.

Remark 2 *For simplicity in this paper we shall write* $\mathrm{dom}(V)$ *instead of* $\cup_{v \in V} \mathrm{dom}(v)$

Accordingly, a clock valuation is a function $v^c : \mathscr{C} \to \mathbb{R}_{\geq 0}$. In this paper we denote clock valuations by v^c. Valuations over other types of variables are denoted by v^d. We denote by $v \models c$ if valuation v satisfies a constraint c. The set of all valuations satisfying a constraint c is denoted by $[\![c]\!]$.

Clock variables are initialized to 0 – that is, initially $v_0^c(c) = 0$ for all $c \in \mathscr{C}$ –, and their values are constantly and steadily increasing (at the same pace for all clocks).

The only operation on clock valuations is the operation *reset*, denoted by $[C \leftarrow 0]v^c$ which sets the value of a set $C \subseteq \mathscr{C}$ to 0, that is $[C \leftarrow 0]v^c(c) = 0$ for $c \in C$ and $[C \leftarrow 0]v^c(c) = v^c(c)$ otherwise. It is an instantaneous operation, after which the value of the clocks will continue to increase.

The state of a timed automaton is defined by the current location and the values of the clocks. Formally, a *state* of $\mathscr{A}$ is a pair $\langle l, v^c \rangle$ where $l \in L(\mathscr{A})$ is a location and v^c is the current valuation.

Two kinds of operations are defined. The state $\langle l, v^c \rangle$ has a *discrete transition* to $\langle l', v^c\prime \rangle$ if there is an edge $e = \langle l, g, r, l' \rangle \in E$ in the automaton such that $v^c \models g$ and $v^c\prime = [r \leftarrow 0]v^c$. The state $\langle l, v^c \rangle$ has a *time transition* to $\langle l, v^c\prime \rangle$ if $v^c\prime$ assigns $v^c(c) + d$ for some non-negative d to each $c \in \mathscr{C}$.

A run of $\mathscr{A}$ is a finite sequence of consecutive transitions $R = \langle l_0, v_0^c \rangle \xrightarrow{t_0} \langle l_0, v_0^c\prime \rangle \xrightarrow{d_0} \langle l_1, v_1^c \rangle \xrightarrow{t_1} \langle l_1, v_1^c\prime \rangle \ldots \langle l_n, v_n^c \rangle \xrightarrow{t_n} \langle l_n, v_n^c\prime \rangle$ where for all $1 \leq i \leq n, t_i$ denotes a time transition and d_i denotes a discrete transition.

3.1.3 Extensions of timed automata

In practice the low descriptive power of the original timed automaton formalism makes it impractical for modeling systems. Over the years many extensions of the formalism have been invented to overcome this issue. In this paper we are focusing on *timed automata with data variables*.

Data variables are variables whose value can only be changed by discrete transitions, e.g. variables of types *integer, bool*, etc. They can appear in constraints to enable transitions (*data guards*) and can be modified by transitions (*update*). However, clock variables are not allowed to appear in data guards or updates. The extended formalism can be formally defined as follows.

Definition 5 A *timed automaton with data variables* is a tuple $\langle L, l_0, E, v_0^d \rangle$ where

- L is the set of locations,
- $l_0 \in L$ is the initial location,

- $E \in L \times \mathscr{B}(\mathscr{C}) \times \mathscr{P}_V \times 2^{\mathscr{C}} \times \mathscr{U}_V \times L$ is the set of edges, where $\mathscr{P}_V$ denotes the set of first order predicates over a set of data variables V and $U \in \mathscr{U}_V$ is a sequence of assignments (updates) of the form $v \leftarrow t$ where $v \in V$ and t is a term built from variables in V and function symbols interpreted over $\mathrm{dom}(v)$.

  An edge is defined by the source location, the clock guard, the data guard, the set of clocks to reset, the updates, and the target location.

- v_0^d is the initial data valuation.

The possible types of data variables and the expressions appearing in data guards and updates may vary by model checker, however, for efficient verification it is important, that the expressive power of the extended formalism is the same as that of the original timed automaton. In many cases this is achieved by restricting the set of possible values to a finite set (e.g. an interval for integers) – this way, the model can be translated to a (simple) timed automaton by encoding the values of data variables to locations. In this paper we denote the set of general conditions on a set of variables V by $\mathrm{cond}(V)$.

The operational semantics are also modified. A state of the extended formalism $\mathscr{A}$ over a set of clock variables $\mathscr{C}$ and a set of data variables V can be described by a tuple $\langle l, v^d, v^c \rangle$ where $l \in L$ is the current location, $v^d : V \to \mathrm{dom}(V)$ is the current *data valuation* and $v^c : \mathscr{C} \to \mathbb{R}_{\geq 0}$ is the *clock valuation*.

While the operational semantics of time transitions are not affected by data variables, discrete transitions become more complex: data guards have to be satisfied and updates have to be executed. Let us denote by $U(v^d)$ the data valuation after executing the sequence of updates U on a system with the initial data valuation v^d. Formally, the state $\langle l, v^d, v^c \rangle$ has a discrete transition to $\langle l', v^d\prime, v^c\prime \rangle$ if there is an edge $e = \langle l, g, p, r, U, l' \rangle \in E$ in the automaton such that $v^c \models g$, $v^d \models p$, $v^c\prime = [r \leftarrow 0]v^c$ and $v^d\prime = U(v^d)$.

Another popular extension of timed automata allows the decomposition of the automaton to a *network* of automata. A network $(\mathscr{A}_1 | \ldots | \mathscr{A}_n)$ is a parallel composition of a set of timed automata. Communication is possible by shared variables or handshake synchronization using *synchronization channels*. A network of timed automata can be transformed into a timed automaton by constructing the product of the automata in the network. For formal definition, the interested reader is referred to [BLL+95].

3.2 Verification of timed automata

In this section we present the basic opreations of zone-based reachability analysis of timed automata. SMT-based verification is beyond the scope of this paper.

In case of (simple) timed automata the reachability problem can be formalized as follows.

Input: A timed automaton $\mathscr{A}$, and a location $l_{\mathrm{trg}} \in L(\mathscr{A})$
Question: Is there a run $\langle l_0, v_0 \rangle \to \langle l_{\mathrm{trg}}, v_n \rangle$ for some v_n in $\mathscr{A}$?

3.2.1 Zone-based verification

An introduction to the abstract domain *zone* can be found in [BY03], along with the operations necessary for verification and an efficient representation, called *difference bound matrix* (DBM).

Definition 6 A *zone* $Z = \{v^c \mid v^c \models g\}$ is a set of clock valuations satisfying some clock constraint g.

Many operations are defined on zones. In this paper the following notations are used:

- $[R \leftarrow 0]Z$ denotes the result of the *reset* operation applied to a zone Z, $R \subseteq \mathscr{C}$:
  $[R \leftarrow 0]Z = \{v^c \mid \exists v^{c}{\prime} \in Z : v^c = [R \leftarrow 0]v^{c}{\prime}\}$
- $[R \leftarrow 0]^{-1}Z, R \subseteq \mathscr{C}$ denotes the inverse of the reset operation:
  $[R \leftarrow 0]^{-1}Z = \{v^c \mid [R \leftarrow 0]v^c \in Z\}$
- $Z^{\uparrow}$ denotes the set of valuations reachable from Z by time transitions:
  $Z^{\uparrow} = \{v^c + t \mid v^c \in Z, t \in \mathbb{N}\}$, where $v^c + t = v^{c}{\prime}$ denotes a valuation where $v^{c}{\prime}(c) = v^c(c) + t$ for all $c \in \mathscr{C}$
- $Z^{\downarrow}$ denotes the set of nonnegative valuations from where Z is reachable by time transitions:
  $Z^{\downarrow} = \{v^c \mid \exists t \in \mathbb{N} : v^c + t \in Z\}$

The core of reachability analysis is to construct an *abstract reachability graph*, that is an abstract representation of the state space. In case of zone-based reachability the abstract reachability graph is called a *zone graph*.

Definition 7 A *zone graph* is a finite graph containing $\langle l, Z\rangle$ pairs as nodes, where $l \in L$ is a location of the automaton and Z is a zone.

A node of the zone graph represents a set of states reachable from the initial state (or a set of states from where the target states are reachable, in case of backward exploration). Edges between nodes correspond to discrete transitions through which those states are reachable.

Remark 3 *In this paper we use the word* represent *for indicating that a node n certifies that the explored state space contains a state s, i.e. $n = \langle l, Z\rangle$ represents $s = \langle l, v^c\rangle$ iff $v^c \in Z$.*

The goal of state space exploration is to create the zone graph. The process is similar to any kind of graph-based state space exploration and will be presented to detail in Subsubsection 4.1.2. The primary task is *postimage calculation* (in case of forward exploration): given a node of the graph under construction $n = \langle l, Z\rangle$ representing the set of reachable states and an edge of the automaton $e = \langle l, g, R, l'\rangle$ the task is to calculate a node n' representing the set of all states that can be reached from those represented by n through e.

The postimage of l is straightforward and the postimage of Z can be calculated by first determining the maximal subzone Z' that satisfies g (Z' represents the set of valuations where the discrete transition is enabled), then resetting all clocks $c \in R$ in Z' (i.e. executing the discrete transition), and finally applying time transitions. Formally, the postimage calculation can be described as $\mathrm{Post}_{g,R}(Z) = ([R \leftarrow 0](Z \cap [\![g]\!]))^{\uparrow}$.

In case of backward exploration the corresponding operation is *preimage calculation*, formally described as $\mathrm{Pre}_{g,R}(Z) = ([R \leftarrow 0]^{-1} Z \cap [\![g]\!])^{\downarrow}$. This operation is basically the inverse of postimage computation with the exception that applying time transitions remains the last step. This way the set of states represented by a node is larger and the zone graph is more compact.

In case of backward exploration the zone graph explored using the presented operation is always finite, but this is not true for forward exploration, since using loop-edges it is possible to create a timed automaton with a run of ever-increasing clock valuations [BY03]. To ensure the termination of the algorithm, extrapolation was introduced, however, in case of diagonal constraints extrapolation may introduce unreachable states to the reachability graph, making the result of the analysis a false positive space [Bou03].

Many approaches have been introduced to ensure correctness, including the operation *split* that splits the zone to extrapolate to smaller zones so that the extrapolation keeps the zone safe [BY03], and CEGAR-based methods where they explore the state space with the original overapproximating algorithm and if they find a spurious trace they refine the state space or the automaton [BLR05], but the complexity of these methods is exponential in the number of diagonal constraints.

3.2.2 Verification of extended timed automata

The reachability problem can be formulated for the extended formalisms. For instance, in case of timed automata with data variables the target states can also depend on the data valuation.

Input: A timed automaton with data variables $\mathscr{A}$, a location $l_{\text{trg}} \in L(\mathscr{A})$, and a first order predicate P_{trg} over the data variables of $\mathscr{A}$
Question: Is there a run from the initial state $\langle l_0, v_0^d, v_0^c \rangle$ to some $\langle l, v_n^d, v_n^c \rangle$ where $v_n^d \models P$?

In case of networks of timed automata, instead of a single control location, the state of the system depends on a *configuration* – that is, a set of locations containing the current location of each automaton. However, reachability criteria usually only determine the location for *some* of the automata (e.g. collision detection only requires two out of an arbitrary number of stations to be transmitting).

The extended reachability problems can be transformed to the original reachability problem over (simple) timed automata, however, in practice it is more efficient to apply the algorithms on timed automata to the extended formalisms. For example, in case of forward exploration calculating the data variables is straightforward based on the updates. On the other hand, in case of backward exploration there can be more than one predecessor states. Since it is complicated to calculate the possible previous values of data variables, in case of extended timed automata forward exploration is preferred [Bou05, BLR05, Bou04].

3.3 Weakest precondition

In our research we used the weakest precondition operation to calculate the preimage of data variables. The weakest precondition can be defined many ways. Here, we define it over predicates and updates.

Definition 8 Given a sequence of updates $U \in \mathscr{U}_V$ and a predicate $P \in \mathscr{P}_V$ describing a post-condition, the *weakest precondition*, denoted by $wp(P,U)$ is the predicate describing the weakest constraint on the initial state ensuring that the execution of U terminates in a state satisfying P:

$$wp : \mathscr{P}_V \times \mathscr{U}_V \to \mathscr{P}_V \text{ such that } [\![wp(P,U)]\!] = \{v^d : V \to \mathrm{dom}(V) \mid U(v^d) \models P\}.$$

The weakest precondition of a sequence of updates can be calculated by iterating backwards over the assignment statements and calculating the weakest precondition one by one. Formally if $U = [v_1 \leftarrow t_1; v_2 \leftarrow t_2]$, then $wp(P,U) = wp(wp(v_2 \leftarrow t_2, P), v_1 \leftarrow t_1)$.

The weakest precondition of a predicate $P \in \mathscr{P}_V$ based on an update $v \leftarrow t$ can be calculated by replacing all occurrences of v in P with t.

Example 1 *Let $y < 5$ be the postcondition of the assignment $y \leftarrow x+3$. Replacing y with $x+3$ yields $x+3 < 5$ that can be automatically simplified to $x < 2$. Therefore if $U(v^d) = v^{d\prime}$, where $v^{d\prime}(y) = v^d(x)+3$ and $v^{d\prime}(v) = v^d(v), v \neq y$, then $wp(y<5,U) = x < 2$. In other words in order to ensure $y < 5$ after assigning $x+3$ to y, initially $x < 2$ must hold.*

In practice the calculation of the weakest precondition is only efficient if the sequences of assignments can be kept small and the assignments are simple. For more information on the weakest precondition operation the reader is referred to [Lei05].

4 Backward exploration using weakest precondition

This section presents our proposed algorithm for backward state space exploration of timed automata with data variables. First the underlying state space exploration approach is explained and then the data structures and the operations specific to our algorithm are presented. The algorithm is also demonstrated on an example and its correctness is formally proven.

4.1 Algorithm

The proposed algorithm is an extension of the zone based backward exploration for simple timed automata to timed automata with data variables using the weakest precondition operation to calculate preimages for data variables.

4.1.1 Overview of the approach

The basis of the algorithm is a simple state space exploration approach that starts by creating the initial node (representing the target states) and calculates the preimage of each node that introduces new states to the explored state space. Backward exploration terminates when the initial state of the automaton is introduced to the state space (hence, reachability is confirmed) or when the complete state space is explored (hence, the target state is unreachable).

Zone-based backward exploration completes the general state space exploration approach with timed automaton-specific data structures (zones) and operations presented in Subsubsection 3.2.1. The proposed algorithm extends zone-based exploration with first-order predicates that are used to represent the values of data variables and the weakest precondition operation to calculate preimages for data variables.

4.1.2 State space exploration

Backward state space exploration initiates from a node n_{trg} representing the target states. For all incoming edges e of the target location (in the automaton) a new node n is created representing the preimage of the target states based on e. A corresponding edge $n_{\text{trg}} \dashrightarrow n$ is also introduced indicating that n represents all states that are from where n_{trg} is reachable through e. The algorithm continues from the newly created nodes.

When a node n is created it is checked if n is covered by some other node n'.

Definition 9 A node of the state space graph n is *covered* by a node n' iff n' represents all states represented by n.

If a new node n is covered by an existing node n' then n can be removed from the graph, and instead of the outgoing edge $n_{\text{post}} \dashrightarrow n$ a new edge $n_{\text{post}} \dashrightarrow n'$ is created indicating that n' represents all states from where n_{post} is reachable through e.

The algorithm terminates when there is a node in the state space graph (zone graph) that represents the initial state of the automaton or when the complete state space is explored.

4.1.3 Data structure and operations

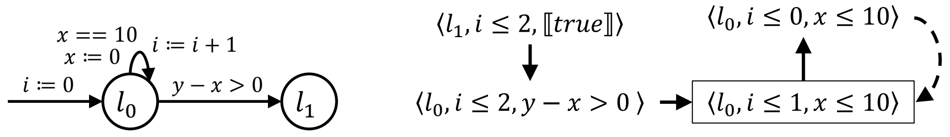
While the state space exploration approach is generally used, the underlying data structure (of the nodes), the initial node, the operations for preimage computation and coverage checking and the termination criteria depend on the formalism. In case of timed automata with data variables the proposed approach uses the following concepts.

Data structure The proposed algorithm extends the data structure of zone-based backward exploration with first order predicates that are used to denote the set of possible values of data variables. Therefore, the nodes of the zone graph can be formalized as $\langle l, P, Z\rangle$, where l is a location of the automaton, P is a first order predicate representing a set of data valuations and Z is a zone representing a set of clock valuations. A state $\langle l, v^d, v^c\rangle$ is represented by a node $\langle l, P, Z\rangle$ iff $v^d \models P$ and $v^c \in Z$.

Initialization The initial node can be formalized as $\langle l_{\text{trg}}, P_{\text{trg}}, [\![true]\!]\rangle$.

Preimage computation For a node $\langle l, P, Z\rangle$ and incoming edge $\langle l', g, p, r, U, l\rangle$ the preimage $\langle l', P', Z'\rangle$ is calculated as follows.

- The preimage of the location is the source location l' of the edge.
- The preimage of the zone can be calculated as presented in Subsubsection 3.2.1: $Z' = \text{Pre}_{g,R}(Z)$.



(a) Input automaton (b) Explored state space

Figure 1: State space exploration example

- P' is only calculated if $Z' \neq \emptyset$. In this case $P' = wp(P,U) \wedge p \wedge \text{cond}(V)$. It is important to check if P' is satisfiable (for instance by using a SAT solver).

Checking coverage A node $\langle l,P,Z\rangle$ is covered by another node $\langle l^c,P^c,Z^c\rangle$ if $l = l^c, Z \subseteq Z^c$ and $P \implies P^c$. There are many ways to check implication. In our implementation we use a solver to determine whether $P \wedge \neg P^c$ is unsatisfiable.

Termination The algorithm terminates if there are no unexplored nodes in the current graph or if it reached the initial state. The set of states represented by a node $n = \langle l,P,Z\rangle$ contains the initial state if $l = l_0$, $v_0^d \models P$, and $v_0 \in Z$.

4.2 Example

This section demonstrates the operation of backward exploration on the automaton presented in Figure 1a and the target location l_1 with predicate $i \leq 2$. In this automaton (which is a modified version of the one presented in [BY03]), there are two clock variables x and y and a data variable i that serves as a loop counter. The property to decide is whether it is possible to reach a state where the current location is l_1 and $i \leq 2$ – considering the automaton the property represents reaching l_2 by taking the loop transition less than two times.

Initially, clocks x and y are both initialized to 0, which means $x = y$ will hold as long as the system stays in l_0. When the loop edge is taken (that can first happen when $x = y = 10$) x is reset but the value of y continues to increase from the same value – that is, taking the loop edge means increasing $y - x$ by 10. The algorithm operates as follows.

The initial node is $n_0 = \langle l_1, i \leq 2, [\![true]\!]\rangle$. The only incoming edge does not change the data valuation, but has a clock guard. Therefore the preimage of the node is $n_1 = \langle l_0, i \leq 2, y - x > 0\rangle$. The preimage of n_1 based on the loop edge can be calculated as follows.

$\text{Pre}_{x=10,\{x\}}(y - x > 0) = ([\{x\} \leftarrow 0]^{-1}(y - x > 0) \cap [\![x = 10]\!])^{\downarrow} = (y > 0 \cap [\![x = 10]\!])^{\downarrow}$, resulting in $x \leq 10$, and $wp(i \leq 2, i := i+1) = i \leq 1$. Therefore, the created node is $n_2 = \langle l_0, i \leq 1, x \leq 10\rangle$.

The preimage of n_2 based on the loop edge is $n_3 = \langle l_0, i \leq 0, x \leq 10\rangle$. Since $i \leq 0 \implies i \leq 1$, n_2 covers n_3. This means all the states represented by n_3 are also represented by n_2, so if n_3 is reachable from a state, n_2 is also reachable. Because of the coverage, node n_3 does not need to be expanded further. The exploration terminates. The explored graph can be seen in Figure 1b with a frame around n_2 and a dashed arrow $n_3 \rightarrow n_2$ (indicating coverage).

Naturally, the reachability algorithm can terminate after creating n_2. Since $i = 0 \models i \leq 1$ and $x = y = 0 \models x \leq 10$, n_2 represents the initial state – the property is evaluated to *true*.

4.3 Proof of correctness

The presented algorithm is a combination of two existing approaches. Zone calculations are proven to be correct, and the weakest precondition operation is *defined* to be correct. The correctness of the algorithm can be proven based on the fact that the values of clock and data variables are independent of each other. Therefore, since the computation of zones and (data) predicates are correct, the complete algorithm is also correct.

The formal proof is constructed as follows.

Claim (Correctness) *The presented algorithm is correct:* $\langle l_0, v_0^d, v_0^c\rangle$ *is represented by one of the nodes of the abstract reachability graph* iff *there is a run* $s_0 = \langle l_0, v_0^d, v_0^c\rangle \to \cdots \to s_n' = \langle l_n, v_n^d, v_n^c{\prime}\rangle$ *of* $\mathscr{A}$ *where* $l_n = l_{trg}$ *is the target location.*

Proof. ($\Leftarrow$) s_n is represented by the initial node. Suppose a state of the run s_k' is represented by some $n_k = \langle C_k, P_k, Z_k\rangle$ in this case n_k also represents s_k since the $v_k^c \in \{v_k^c{\prime}\}^{\downarrow}$. Let $e_{k-1} = \langle l_{k-1}, g, p, r, U, l_k\rangle$ be the edge corresponding to the discrete transition $s_{k-1} \to s_k$.

In this case the following claims hold:

- $v_{k-1}^c \in Pre_{g,r}(\{v_k^c\})$, since the zone operations are correct,
- $v_{k-1}^d \models wp(\bigwedge_{v\in V} v = v_k^d(v), U)$ (since the weakest precondition was defined like that), therefore
- $v_{k-1}^d \models wp(P, U)$ for any P where $v_k^d \models P$,
- $v_{k-1}^d \models p$ because the transition is enabled, and
- $v_{k-1}^d \models \text{cond}(V)$.

Therefore, there is also an edge of the graph representing e from n_k to some other node n_{k-1} that represents s_{k-1} and – by induction – there is also a node representing s_0.

($\Rightarrow$) Let node $n_0 = \langle l_0, P_0, Z_0\rangle$ be a node of the graph, where l_0 is the initial location $v_0^c \in Z_0$ and $v_d^0 \models P_0$. If a node n of the abstract reachability graph is not the initial node, then it was created along with an incoming edge $e(n)$. Find the path $n_i \xrightarrow{e(n_{i-1})} n_{i-1} \ldots n_1 \xrightarrow{e(n_0)} n_0$ where n_i is the initial node.

Because of the way the edges are chosen, all states represented by a node n_k are a preimage of some state represented by n_{k+1} (other incoming edges might have been introduced based on coverage and in that case, the preimage might be just a subset of the states represented by n_{k+1}). Therefore for all states s_k represented by n_k there is a state s_{k+1} represented by n_{k+1} such that s_{k+1} is reachable from s_k by the edge corresponding to $e(n_k)$. By induction, n_i represents a state reachable from s_0.

□

Remark 4 *The path* $n_i \xrightarrow{e(n_{i-1})} n_{i-1} \ldots n_1 \xrightarrow{e(n_0)} n_0$ *is the abstract representation of a run from* s_0 *represented by* n_0 *to* s_i' *represented by* n_i *through the discrete transitions* $e(n_{i-1}) \to \cdots \to e(n_0)$*. Delay transitions can also be derived by further analysis of the path.*

5 Comparison of backward and forward exploration

This section presents the measurements ran to analyze and compare the efficiency of the proposed algorithm to forward approaches. The results are discussed and possible improvements are also proposed.

5.1 Measurements

To analyze the applicability and the efficiency of the algorithm we have implemented it in the Theta verification framework [THV$^+$17] and ran measurements on the XtaBenchmarkSuite[1], that is a set of test cases for benchmarking timed automaton verification algorithms (we have presented an earlier version of the benchmark suite in [FB18], but it has been improved since). To compare the algorithm to forward exploration we also ran measurements on the algorithm presented in [HSW13] that we have also implemented in the Theta framework.

The benchmark suite contains *networks of timed automata with data variables*. As mentioned in Subsubsection 3.2.2, for some inputs the properties only define the target location for a few of the automata in the network. In these cases, the possible target configurations were manually collected – all possible combinations of the locations of the remaining automata were included. In many cases, this resulted in a set of target configurations too large for efficient verification. However, by observing the models, we could eliminate many of the configurations. To demonstrate the significance of describing a precise property, we ran the backward exploration algorithm on both the original and the modified property. These modifications did not affect the forward exploration algorithms.

The measurements were executed on a virtual 64 bit Windows 7 operating system with a 2 core CPU (2.50 GHz) with 1,5 GB of memory. Each algorithm was run 10 times on each input, the longest and the shortest was eliminated and the average of the remaining runtimes was taken. The timeout was 10 minutes.

5.2 Evaluation

The XtaBenchmarkSuite consists of more models than those mentioned here, but we have excluded those that contain a model element that is not yet supported by Theta (e.g. broadcast channels) and those where the property to check is not a reachability property (e.g. liveness properties) or only constrains the data variables. We have also excluded the models where both of the studied algorithms timed out.

Table 1 describes the results of the benchmarks on diagonal-free timed automata. The column *Name* contains the name of the model in the XtaBenchmarkSuite and in case of scalable models, the value of the parameter. The next two columns contain the number of target configurations as described by the original and – when we managed to refine it – the precise property. The remaining columns contain the runtime and the number of explored nodes by the algorithms. The columns denoted by *LU* correspond to the algorithm presented in [HSW13], while the remaining columns correspond to the execution of our backward exploration algorithm on the model with the original and the modified property.

[1] https://github.com/farkasrebus/XtaBenchmarkSuite

Table 1: Results on diagonal-free systems

Name	Target configurations		LU		BW			
	original	precise			original		precise	
			time (ms)	nodes	time (ms)	nodes	time (ms)	nodes
EXSITH	1	-	1	4	-	-	2	5
LATCH	1	-	2	7	-	-	1	2
SRLATCH	4	1	0	6	5	28	4	16
ANDOR	4	1	11	9	42	39	13	21
ENGINE	288	-	35	674	3731	8783	-	-
SIMOP	8960	128	72	325	9820	26271	800	1517
MALER	1	-	553	7157	-	-	134127	53519
MUTEX	5	-	1248	5563	[TO]	-	-	-
CRITICAL2	56	1	135	112	1631	1914	18	22
CRITICAL3	784	8	148	1632	[TO]	-	599	811
CRITICAL4	10976	64	770	25202	[TO]	-	233114	51488
CSMA2	4	-	23	18	16	26	-	-
CSMA3	16	-	29	71	247	338	-	-
CSMA4	64	-	55	262	16547	11306	-	-
CSMA5	256	-	177	855	[TO]	-	-	-
FISCHER2	1	-	14	18	19	11	-	-
FISCHER3	4	-	11	65	93	78	-	-
FISCHER4	16	-	59	220	476	594	-	-
FISCHER5	64	-	79	727	10221	10426	-	-
FISCHER6	256	-	366	2378	96102	93735	-	-
FISCHER7	124	-	1913	7737	[TO]	-	-	-
LYNCH2	1	-	20	38	18	33	-	-
LYNCH3	9	-	32	125	656	821	-	-
LYNCH4	81	-	47	380	35344	27523	-	-
TRAIN2	64	-	14	44	156	380	-	-
TRAIN3	256	-	37	165	694	1330	-	-
TRAIN4	1024	-	315	1123	6886	10290	-	-
TRAIN5	4096	-	3874	6488	66837	79892	-	-
TRAIN6	16384	-	103455	60379	[TO]	-	-	-

Table 2: Results on diagonal timed automata

Name	LU				BW	
	Transformation time (ms)	Algorithm time (ms)	Total time (ms)	Nodes	Time (ms)	Nodes
SPLIT (original)	4	4	8	9	3	4
SPLIT (modified)	27	16	43	10	4	4
DIAG (original)	14	6	20	22	8	8
DIAG (modified)	32	22	54	24	14	8

Table 2 describes the results of the benchmarks on timed automata with diagonal constraints. We did not manage to find an industrial case study containing diagonal constraints, thus we use the models we found in the literature as examples. Model *SPLIT* was introduced in [BY03] to demonstrate the incorrectness of the forward exploration algorithm and *DIAG* was used to demonstrate techniques describes in [BLR05]. We also run the algorithms on modified versions: we added new edges to these examples to create cyclic models.

5.2.1 Weakest precondition

One of our research goals was to find out if the weakest precondition operation can be used to efficiently calculate data variables during backward exploration of timed automata. Models of the benchmark suite only contain a few data variables (protocols usually one for each participant, and inputs of category *system* have integer variables, but their set of possible values is rather small). A possible difficulty would have occurred if the abstract reachability graphs were deeper, but this was not the case as we used breadth-first search. Out of these models *ENGINE* had the deepest reachability graph with a depth of 63.

The results show that despite its limitations (such as the solver overhead introduced by coverage checking) in practice weakest precondition can be used to handle data variables. Furthermore, while most algorithms calculate the values of data variables explicitly, weakest precondition makes it possible to use predicate abstraction over the data variables that can also increase the efficiency of an algorithm.

5.2.2 Property definition

One of the motivations for using backward exploration is that the state space explored from the target states might be smaller than the reachable state space. However, in practice the properties describing the target states are kept simple, only mentioning a few locations and variables of the system. While an understandable property definition is usually beneficial, it also increases the number of target states exponentially.

For instance, in case of mutual exclusion protocols (that appears to be the most common area of applicability for timed automata) the target state is any state where at least two processes are in the critical section. In case of a system of n similar processes containing k locations there are k^{n-2} possible control vectors that satisfy this property. Clocks and data variables can be handled with symbolic methods, but locations have to be explored explicitly.

We believe this issue could be overcome *in many cases*, such as the *CRITICAL* example, where the target location is only given for the *Prod cell* automata, however it is easy to determine the current location of the corresponding *Arbiter* automata. Therefore, we believe that some initial static analysis of the automaton and the constraints (e.g. the *forward* exploration of the location graph) could decrease the number of initial states.

As it is seen in Table 1 adding details to the property increases the efficiency of backward exploration.

5.2.3 Timed automata with diagonal constraints

One of the key strengths of backward exploration for timed automata is that it is *correct* (sound) for timed automata with diagonal constraints. The transformation of a timed automaton containing diagonal constraints to a diagonal-free automaton takes time and increases the size of the automaton and thus makes verification less efficient – as demonstrated in Table 2.

We were only able to perform these measurements on small examples from the literature, since *there are no public industrial case studies of timed automata with diagonal constraints*. We believe this is due to the fact that most analysis tools do not support diagonal constraints and because of this the case studies are stored in diagonal-free form.

5.2.4 General conclusions

In case of timed automata with diagonal constraints, backward exploration performs better on the small examples, although, we could not find an industrial example with diagonal constraints to support this claim.

In case of diagonal-free timed automata forward exploration performs better in most cases. However, for many small models backward exploration explored a smaller state space, e.g. the *LATCH* circuit and protocols with a small number of participants.

Forward and backward exploration are complementary techniques. Theta is able to perform both which makes it possible to chose the more appropriate one for a certain model.

5.3 Possible improvements

One of the reasons why forward exploration performs better on the case studies is that there are many optimizations for zone-based forward search (e.g. LU abstraction, lazy abstraction, etc. – see Section 2). It would be desired to apply these optimizations for backward exploration.

Throughout these measurements we analyzed the models manually to strenghten the properties, although it would be possible (and beneficial) to perform automatic preprocessing on the input to decrease the number of target states (initial states for the algorithm).

Backward exploration can also be combined with other techniques to improve the efficiency of existing algorithms. A common application for backward exploration is to run it in parallel with forward exploration – i.e. *bidirectional search*. This way, if the target state is reachable, the explored state space becomes smaller than a single search from any direction. The two algorithms can also increase each others efficiency by exchanging information on the explored state space [VGS13]. Bidirectional search for timed automata appears to be an area worth exploring.

6 Conclusions and future work

In this paper we have presented an algorithm that uses zone-based backward exploration for the reachability analysis of timed automata. Unlike zone-based forward exploration, this algorithm is also applicable for timed automata with diagonal constraints. We have extended the algorithm to handle complex data: the weakest precondition operation is suitable for calculating the values of data variables. The combination of the two algorithms resulted and efficient procedure for timed systems with data.

XtaBenchmarkSuite is a benchmark suite consisting of timed automata including small examples and industrial case studies. XtaBenchmarkSuite was extended in this paper to evaluate the various timed algorithms more rigorously. Measurements on the XtaBenchmarkSuite demonstrated the efficiency and the applicability of our new approach. We have shown that while the algorithm performs better on timed automata with diagonal constraints than forward exploration, some improvements and optimizations are necessary in order to be applicable for industrial case studies. Our results provide a step towards the efficient verification of real-time systems.

In the future we will improve the backward exploration algorithm to solve complex industrial case studies. We will analyze the applicability of the improvements of the forward exploration algorithm presented in [HSW13, TM17] to our backward exploration approach. We will also combine backward and forward search algorithms – as it was investigated that bidirectional search is efficient for SAT-based algorithms in [VGS13]. We will also study the combination of forward and backward algorithms in the CEGAR framework as it was discussed in [RRT08].

In order to further investigate the strengths and weaknesses of the timed algorithms, we will extend the XtaBenchmarkSuite with more industrial case studies (e.g. the FlexRay protocol [GEFP10]) and perform an exhaustive benchmark on the existing approaches. Based on the results a – possibly automatic – process can be derived for deciding which algorithm is expected to be the most efficient for verifying a given system. Diagonal constraints represent one extension of timed automata, which proved to be useful for a certain class of models. In the future, we have to analyze other extensions from the verification point of view.

Bibliography

[AD91] R. Alur, D. L. Dill. The Theory of Timed Automata. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*. Pp. 45–73. Springer, 1991. doi:10.1007/BFb0031987

[BC05] P. Bouyer, F. Chevalier. On Conciseness of Extensions of Timed Automata. *Journal of Automata, Languages and Combinatorics* 10(4):393–405, 2005.

[BLL$^+$95] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, W. Yi. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In *DIMACS/SYCON 1995*. Pp. 232–243. Springer, 1995. doi:10.1007/BFb0020949

[BLR05] P. Bouyer, F. Laroussinie, P. Reynier. Diagonal Constraints in Timed Automata: Forward Analysis of Timed Systems. In *FORMATS 2005*. Pp. 112–126. Springer, 2005. doi:10.1007/11603009_10

[BM07] A. R. Bradley, Z. Manna. *The calculus of computation - decision procedures with applications to verification*. Springer, 2007. doi:10.1007/978-3-540-74113-8

[Bou03] P. Bouyer. Untameable Timed Automata! In *STACS 2003*. Pp. 620–631. Springer, 2003. doi:10.1007/3-540-36494-3_54

[Bou04] P. Bouyer. Forward Analysis of Updatable Timed Automata. *Formal Methods in System Design* 24(3):281–320, 2004. doi:10.1023/B:FORM.0000026093.21513.31

[Bou05] P. Bouyer. An Introduction to Timed Automata. *ETR 2005*, pp. 25–52, 2005.

[BY03] J. Bengtsson, W. Yi. Timed Automata: Semantics, Algorithms and Tools. In *ACPN 2003*. Pp. 87–124. Springer, 2003. doi:10.1007/978-3-540-27755-2_3

[DS+04] B. Dutertre, M. Sorea et al. Timed systems in SAL. Technical report, SRI International, Computer Science Laboratory, 2004.

[FB18] R. Farkas, G. Bergmann. Towards Reliable Benchmarks of Timed Automata. In *Proceedings of the 25th PhD Mini-Symposium*. Pp. 20–23. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2018.

[FS01] A. Finkel, P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science* 256(1-2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X

[GEFP10] M. Gerke, R. Ehlers, B. Finkbeiner, H.-J. Peter. Model Checking the FlexRay Physical Layer Protocol. In *FMICS 2010*. LNCS 6371, pp. ”132–147”. Springer, 2010. doi:10.1016/j.entcs.2005.04.014

[GMS18] P. Gastin, S. Mukherjee, B. Srivathsan. Reachability in Timed Automata with Diagonal Constraints. In *CONCUR 2018*. Pp. 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.28

[HNSY92] T. A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic Model Checking for Real-time Systems. In *LICS 1992*. Pp. 394–406. IEEE, 1992. doi:10.1109/LICS.1992.185551

[HRSY14] H. Hojjat, P. Rümmer, P. Subotic, W. Yi. Horn Clauses for Communicating Timed Systems. In *HCVS 2014*. Pp. 39–52. 2014.
doi:10.4204/EPTCS.169.6

[HSW13] F. Herbreteau, B. Srivathsan, I. Walukiewicz. Lazy Abstractions for Timed Automata. In *CAV 2013*. Pp. 990–1005. Springer, 2013.
doi:10.1007/978-3-642-39799-8_71

[IW14] T. Isenberg, H. Wehrheim. Timed Automata Verification via IC3 with Zones. In *ICFEM 2014*. Pp. 203–218. Springer, 2014.
doi:10.1007/978-3-319-11737-9_14

[Lei05] K. R. M. Leino. Efficient weakest preconditions. *Inf. Process. Lett.* 93(6):281–288, 2005.
doi:10.1016/j.ipl.2004.10.015

[Mit07] I. M. Mitchell. Comparing Forward and Backward Reachability as Tools for Safety Analysis. In *HSCC 2007*. Pp. 428–443. Springer, 2007.
doi:10.1007/978-3-540-71493-4_34

[MP95] O. Maler, A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In *CHARME 1995*. Pp. 189–205. Springer, 1995.
doi:10.1007/3-540-60385-9_12

[MPS11] G. Morbé, F. Pigorsch, C. Scholl. Fully Symbolic Model Checking for Timed Automata. In *CAV 2011*. Pp. 616–632. Springer, 2011.
doi:10.1007/978-3-642-22110-1_50

[RRT08] F. Ranzato, O. Rossi-Doria, F. Tapparo. A Forward-Backward Abstraction Refinement Algorithm. In *VMCAI 2008*. Pp. 248–262. Springer, 2008.
doi:10.1007/978-3-540-78163-9_22

[RSV10] A. P. Ravn, J. Srba, S. Vighio. A Formal Analysis of the Web Services Atomic Transaction Protocol with UPPAAL. In *ISoLA 2010*. Pp. 579–593. Springer, 2010.
doi:10.1007/978-3-642-16558-0_47

[THV$^+$17] T. Tóth, A. Hajdu, A. Vörös, Z. Micskei, I. Majzik. Theta: a Framework for Abstraction Refinement-Based Model Checking. In *FMCAD 2017*. Pp. 176–179. IEEE, 2017.
doi:10.23919/FMCAD.2017.8102257

[TM17] T. Tóth, I. Majzik. Lazy Reachability Checking for Timed Automata using Interpolants. In *Formal Modelling and Analysis of Timed Systems*. LNCS 10419, pp. 264–280. Springer, 2017.
doi:10.1007/978-3-319-65765-3_15

[TM18] T. Tóth, I. Majzik. Lazy Reachability Checking for Timed Automata with Discrete Variables. In *SPIN 2018*. Pp. 235–254. Springer, 2018.
doi:10.1007/978-3-319-94111-0_14

[VGS13] Y. Vizel, O. Grumberg, S. Shoham. Intertwined Forward-Backward Reachability Analysis Using Interpolants. In *TACAS 2013*. Pp. 308–323. Springer, 2013. doi:10.1007/978-3-642-36742-7_22

[Yov97] S. Yovine. KRONOS: A Verification Tool for Real-Time Systems. *International Journal on Software Tools for Technology Transfer* 1(1-2):123–133, 1997. doi:10.1007/s100090050009

[YPD94] W. Yi, P. Pettersson, M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In *IFIP 1994*. Pp. 243–258. 1994.