SAT-based Unbounded Model Checking of Timed Automata

Wojciech Penczek    Maciej Szreter
Institute of Computer Science, PAS
Ordona 21, 01-237 Warsaw, Poland
{penczek, mszreter}@ipipan.waw.pl

Abstract

We present an improvement of the SAT-based Unbounded Model Checking (UMC) algorithm. UMC, a symbolic approach introduced in [7], uses propositional formulas in conjunctive normal form (CNF) instead of binary decision diagrams (BDD). The key part of the method consists in elimination of universal quantifiers, where the assignments making a formula non-valid are blocked by blocking clauses. The algorithm suffers from an exponential number of such clauses. Our idea is to allow in blocking clauses literals corresponding not only to variables encoding states, but also to more general subformulas over these variables, thus describing sets of states. A hybrid algorithm is proposed for computing timed part of these clauses, based on the well-known Difference Bound Matrices. The optimization results in a considerable reduction in the size and the number of generated blocking clauses, thus improving the overall performance. This is shown on the standard benchmark of Fischer’s Mutual Exclusion protocol.

1 Introduction

Nowadays, model checking is becoming an acknowledged method supporting the design of complex systems, with many successful applications around. However, the combinatorial explosion is one of its major problems. Since the limitations of the algorithms representing state spaces explicitly are well known, the search for new techniques is mostly focused on symbolic methods, working with sets of states rather than with separate states only.

The advances in this area are closely related to the theory and practical methods for propositional logic. The problem of checking satisfiability for propositional formulas (SAT) is known to be NP-complete. However, in the last decade many very efficient algorithms testing satisfiability have been designed. Several verification problems can be translated to checking satisfiability of propositional formulas. Bounded Model Checking (BMC) seems to be the state-of-the-art SAT-based model checking method. This field has seen a rapid development recently [10]. Some types of errors can be easily found in surprisingly large systems. Despite of these undoubtful advantages, BMC has a few weak points. It is still rather a method of falsification than validation. Moreover, BMC is restricted to the universal or the existential fragment of a specification logic. Given these facts, one can ask whether SAT could be used in model checking in a different way. The verification based on Binary Decision Diagrams (BDD) [6] is an obvious analogue. BDD-based model checking constitutes a well developed branch of symbolic methods in automated verification. Unbounded Model Checking (UMC) [7] emerged as a SAT-based counterpart of BDD in 2002. However, the method has not achieved a wide popularity since then, and although some extensions were reported [5], it seems that the performance of the algorithm is still inferior to other symbolic approaches based on BDDs. In the conclusions of [7], two major problems are stated:

- although CNF formulas are used in the scope of a single fixpoint iteration, the formulas encoding the whole state space are represented by semi-canonical Directed Acyclic Graphs. This representation can be much less concise than BDD, because it often distinguishes between equivalent but syntactically different formulas,
- blocking clauses are built over the set of state variables only. This level is too detailed and it often leads to generating exponentially many clauses.

In this paper we propose a solution to the second problem. In [7], it is stated: “If a solution can be found to this problem, a dramatic improvement in performance might result”.

Our modification consists in generating blocking clauses over an extended set of variables, including the variables encoding subformulas over propositions.

1.1 Related Work

As we have already mentioned the idea of UMC was introduced in [7] and extended into epistemic logics and
2 Quantified Propositional Logic

In this section we introduce the preliminary notions concerning Propositional Logic, a conversion of propositional formulas to CNF, and an elimination of universal quantifiers from quantified propositional formulas.

Let \( PV \) be a finite set of (propositional) variables. The formulas of Propositional Logic are built from variables of \( PV \) in the standard way using boolean operators: \( \land \) - conjunction, \( \lor \) - disjunction, \( \neg \) - negation, and \( \Rightarrow \) - implication. Let \( \mathcal{F} \) denote a set of all the propositional formulas. For each propositional formula \( \alpha \) a set of its subformulas \( \text{Subform}(\alpha) \) is defined in the usual way. A literal \( l \) is a variable of \( PV \), or its negation. A clause \( c \) is a disjunction of zero or more literals \( l_1 \lor \cdots \lor l_m \). By \( \mathcal{C} \) we denote a set of the clauses. A formula is in conjunctive normal form (CNF) if it is a conjunction of zero or more clauses \( c_1 \land \cdots \land c_k \).

An assignment \( A \) is a partial function \( A: PV \rightarrow \{0, 1\} \), where 1 stands for \( \text{true} \) and 0 stands for \( \text{false} \). An assignment is said to be \textit{total} when its domain equals to \( PV \). A total assignment is said to be \textit{satisfying} for a formula \( \alpha \) when the value of \( \alpha \) for the assignment \( A \) under the usual interpretation of the Boolean connectives is 1, denoted by \( A(\alpha) = 1 \). We will equate an assignment \( A \) with a conjunction of a set of literals, specifically the set containing \( \neg p \) for all \( p \in \text{dom}(A) \) such that \( A(p) = \text{false} \) and \( p \) for all \( p \in \text{dom}(A) \) such that \( A(p) = 1 \).

Following [1], we represent propositional formulas by directed acyclic graphs (DAGs), with the graph \( \text{DAG}(\alpha) \) representing a formula \( \alpha \). Contrary to semantic representations like BDD, our representation encodes explicitly the syntax instead of the truth table of a propositional formula.

In this paper we make an extensive use of efficient SAT-solvers, i.e., algorithms checking satisfiability of propositional formulas. Let \( \text{SAT}(\cdot) \) refer to a generic SAT-solver, which given a formula either returns its satisfying assignment or diagnoses that no such assignment exists.

2.1 Conversion to CNF

Most of the SAT-solvers accept formulas in CNF, what is motivated mostly by efficient data structures and operations exploiting CNF. For \( \alpha \in \mathcal{F} \), let \( PV(\alpha) \subseteq PV \) denote the set of variables used in \( \alpha \) and \( PV^C(\alpha) = \{l_\beta \in PV \mid \beta \in \text{Subform}(\alpha)\} \) denote a set of the literals corresponding to the subformulas of \( \alpha \).

The standard translation of propositional formulas to CNF is called \( \text{toCNF}(\cdot) \) [9]. For each formula \( \alpha \), \( \text{toCNF}(\alpha) \) returns the formula in CNF defined over variables of \( PV^C(\alpha) \). Every subformula \( \beta \) of \( \alpha \) is represented by the literal \( l_\beta \in PV^C(\alpha) \), and for every assignment \( A \) such that \( A(\text{toCNF}(\alpha)) = 1 \), we have \( A(l_\beta) = A(\beta) \). Consequently, the CNF formula \( \text{toCNF}(\alpha) \lor l_0 \) is satisfiable iff \( \alpha \) is satisfiable. This fact is commonly exploited for testing satisfiability. Moreover, the formula \( \alpha \) is valid when the CNF formula \( \beta = \text{toCNF}(\alpha) \land l_\alpha \) is unsatisfiable, which is used for translating \( \alpha \) to an equivalent CNF formula. More details can be found in [9].

2.2 Elimination of universal quantifiers

Quantified Boolean Formulas (QBF, for short) constitute a fragment of the first-order logic extending propositional logic with quantifiers ranging over propositions. The syntax of QBF is defined in BNF as follows:

\[
\alpha := p \mid \neg \alpha \mid \alpha \land \alpha \mid \exists p.\alpha \mid \forall p.\alpha.
\]

The semantics of the quantifiers is as follows: \( \exists p.\alpha \equiv \alpha(p \leftarrow \text{true}) \lor \alpha(p \leftarrow \text{false}) \) and \( \forall p.\alpha \equiv \alpha(p \leftarrow \text{true}) \land \alpha(p \leftarrow \text{false}) \). According to this semantics, we use the notation \( \forall p.\alpha \) to denote \( \forall v[1], \forall v[2], \ldots, \forall v[m].\alpha \), and for a set of variables \( U \subseteq PV \), by \( \forall U.\alpha \) we mean the universal quantification of \( \alpha \) over all the elements of \( U \).

The algorithm \( \text{SAT}(\cdot) \) can be used for removing universal quantifiers [7] from a QBF formula. This is applied in order to compute a propositional formula in CNF equivalent to \( \forall U.\alpha \). In Fig. 1 a pseudo-code of the algorithm \( \text{forall}(\cdot) \) is shown; \( \text{forall}(\alpha, U) \) returns a propositional formula equivalent to \( \forall U.\alpha \). The algorithm exploits the fact that each clause in a CNF formula equivalent to the input formula must be satisfied for any assignment of the quantified variables. Moreover, the satisfying assignments for \( \beta \), i.e., those which falsify \( \alpha \), are excluded by means of blocking clauses. These clauses produce the resulting CNF formula \( \chi \). The algorithm works on-the-fly removing quantified variables as soon as a new blocking clause is generated.

**Definition 1 (Blocking assignment, blocking clause)**

Consider the algorithm \( \text{forall}(\cdot) \) (Alg. 1). A satisfying assignment \( A_\alpha \) for \( \beta \), found by the algorithm, is called a blocking assignment. A blocking clause \( c_b \) for \( A_\alpha \) is a clause over the set of variables \( PV(\alpha) \) satisfying the following two properties: (i) \( A_\alpha(c_b) = 0 \), and (ii) \( \alpha \Rightarrow c_b \).
Algorithm 1 procedure forall(α, U)
1: χ = ∅, β = toCNF(F(α)) ∨ l_α
2: while A_α = SAT(β) ≠ ∅ do
3: compute the blocking clause cb
4: for each p ∈ U, remove literals p and ¬p from cb
5: χ = χ ∧ cb, β = β ∧ cb
6: return χ

Theorem 2.1 [7] In Algorithm 1, when the formula β becomes unsatisfiable (the condition in line 2 is false), χ is a propositional formula in CNF equivalent to ∀U.ϕ.

2.3 Quantifier elimination under ∪

It is typical in the symbolic model checking that some operations are considered under a restriction, i.e., a propositional formula β describing a restriction is given and the valuations satisfying the resulting formula should also satisfy β. In the case of elimination of universal quantifiers we would like to evaluate (∀U.α) ∪ β. This can be introduced into the algorithm forall() by substituting CNF(F(α)) with CNF(α) ∧ CNF(β) ∧ l_β in line 1. Thanks to that, the algorithm will consider only assignments that make α false but β true. The assignments making both α and β false will not be obtained at all, reducing the computational cost.

In the case of UMC, the restriction is used to simplify the fixpoint computations, what is explained later.

3 Timed automata and model checking

In this section we define timed automata, their discretizations, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).

In timed automata, the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions, and models generated. We start with some preliminary notions. In what follows, let \( \mathbb{N} (\mathbb{R}_+) \) denote the set of natural (non-negative real numbers, respectively).
model concurrent system by networks of progressive timed automata.

For an action \( a \in \Sigma \), we define the set of the indices of all components including \( a \): \( \Sigma(a) = \{i \in \{1, \ldots, n\} \mid a \in \Sigma_i\} \).

**Definition 3 (Network and product of timed automata)**

A network of timed automata \( \mathcal{T} \mathcal{A} \) is a set of \( n \) timed automata (called components) \( \mathcal{T} \mathcal{A} = \{ \mathcal{T} \mathcal{A}_i \mid i \in \{1, \ldots, n\}\} \), where \( \mathcal{T} \mathcal{A}_i = (\Sigma_i, L_i, l_0^i, E_i, \mathcal{X}_i, I_i) \). The product of \( \mathcal{T} \mathcal{A} \) is the timed automaton \( \mathcal{T} \mathcal{A} = (\Sigma, L, l^\prime, E, \mathcal{X}, I) \), where \( \Sigma = \bigcup_{i \in \{1, \ldots, n\}} \Sigma_i \), \( L = \prod_{i \in \{1, \ldots, n\}} L_i \), \( l^i = (l_1^i, \ldots, l_n^i) \), \( \mathcal{X} = \bigcup_{i \in \{1, \ldots, n\}} \mathcal{X}_i \), \( I((l_1, \ldots, l_n)) = \bigwedge_{i \in \{1, \ldots, n\}} I_i(l_i) \), and the transition relation is given by:

\[
((l_1, \ldots, l_n), a, \bigwedge_{i \in \Sigma(a)} c_{i}, \bigcup_{i \in \Sigma(a)} Y_i, (l_1', \ldots, l_n')) \in E
\\iff
(V_{i} \in \Sigma(a)) \ (l_i, a, c_{i}, Y_i, l_i') \in E_i \ and \ \forall i \in \{1, \ldots, n\} \Sigma(a) \ l_i' = l_i.
\]

For the reasons to be explained later, in this paper we consider only automata without upper invariants, i.e., those of the form \( x \sim c \) for \( \sim \in \{<, \leq\} \). We also assume that the sets of clocks of each two components are disjoint.

### 3.2 Abstract Discretized Models

In this section we define an equivalence on clocks and abstract discretized models. Because the concrete space is infinite, an abstraction must be introduced in order to identify the equivalent clock valuations:

**Definition 4 (Equivalence of clock valuations)** For two clock valuations \( v, v' \in \mathbb{R}^{n_\Sigma} \), \( v \sim v' \iff \) for all \( x, x' \in \mathcal{X} \) the following conditions are met:

1. \( v(x) > c_{max} \iff v'(x) > c_{max} \).
2. If \( v(x) \leq c_{max} \ and \ v(x') \leq c_{max} \) then
   a) \( |v(x)| = |v'(x)| \).
   b) \( \text{frac}(v(x)) = 0 \iff \text{frac}(v'(x)) = 0 \), and
   c) \( \text{frac}(v(x)) \leq \text{frac}(v'(x)) \iff \text{frac}(v'(x)) \leq \text{frac}(v(x)) \).

Let \( \mathcal{T} \mathcal{A} = (\Sigma, L, l^\prime, E, \mathcal{X}, I) \) be a diagonal-free timed automaton with \( n_\Sigma \) clocks and \( V_{\mathcal{T} \mathcal{A}} \) be a valuation function. Next, let \( \mathcal{M}(\mathcal{T} \mathcal{A}) = (C(\mathcal{T} \mathcal{A}), V_{\mathcal{T} \mathcal{A}}^\Delta) \) be the concrete model for \( \mathcal{T} \mathcal{A} \), where \( V_{\mathcal{T} \mathcal{A}}^\Delta(l, v) = V_{\mathcal{T} \mathcal{A}}(l) \). We choose the discretization step \( \Delta = 1/d \), where \( d \) is a fixed even number greater than \( 2n_\Sigma \). The discretized clock space is defined as \( \mathbb{D}^{n_\Sigma}, \) where \( \mathbb{D} = \{ k \Delta \mid 0 \leq k \leq 2c_{max} + 2 \} \) for \( k \in \mathbb{N} \).

\(^1\) A good choice for \( d \) is the minimal such a number, which equals to \( 2^d \) for some \( d' \).

**Definition 5** The (abstract) discretized model of a timed automaton \( \mathcal{T} \mathcal{A} \) is a finite structure \( \mathcal{M}(\mathcal{T} \mathcal{A}) = (Q, q^i, \rightarrow, a) \), where \( Q = \mathbb{L} \times \mathbb{D}^{n_\Sigma}, q^i = (l^i, Z^i) \) and \( \rightarrow, a \subseteq Q \times \Sigma \times Q \) is defined as follows:

1. \( (l, v) \xrightarrow{a} (l, w) \iff (l, v') \xrightarrow{\delta} (l, w') \) for some \( \delta \in \mathbb{R}^+ \) and some \( v', w' \in \mathbb{D}^{n_\Sigma} \) such that \( v \sim v' \) and \( w \sim w' \).

(\text{action transition combined with future projection transition})

We use the logic \( CTL \) having the syntax as follows:

\[
\varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid EX \varphi \mid EG \varphi \mid E(\varphi U \varphi).
\]

\( CTL \) is interpreted in the standard way over \( \mathcal{M}(\mathcal{T} \mathcal{A}) \).

### 3.3 Encoding of the transition relation

For the encoding we require that every component \( \mathcal{T} \mathcal{A}_i = (\Sigma_i, L_i, l_0^i, E_i, \mathcal{X}_i, I_i) \) of the network \( \mathcal{T} \mathcal{A} \) satisfies the following two conditions: (i) each pair of local locations is connected with at most one transition labelled with an action \( a \in \Sigma_i \), (ii) all the local transitions of \( \mathcal{T} \mathcal{A}_i \) labelled with an action \( a \in \Sigma_i \), reset the same clocks. Notice that each automaton can be translated to the above form by adding fresh transition labels.

Now we give details of this translation. We begin with the encoding of the states in the model under consideration. Since the set of states \( Q \) of our model is finite, every element of \( Q \) can be represented as a bit vector \( (s[1], \ldots, s[b_i]) \) of length \( b_i \) depending on the number of locations in \( L \), the size of the set \( D \) and the number of clocks. The bit vector consists of two parts, the first of which is used to represent the location of a state of the automaton, whereas the second represents the timed part of that state (i.e., the clock valuation)\(^2\).

Each state \( s \) (given as a bit vector) can be encoded by a valuation of a vector \( w = (w[1], \ldots, w[b_i]) \) (called a \emph{global state variable}), where \( w[i] \), for \( i = 1, \ldots, b_i \) is a propositional variable (called \emph{state variable}). Notice that we distinguish between states of \( Q \) represented as sequences of 0’s and 1’s (we refer to these as valuations of \( w \)) and their encodings in terms of propositional variables \( w[i] \).

Now, we give more details of this encoding. Every state \( (l, u) \in Q \) of \( \mathcal{M}(\mathcal{T} \mathcal{A}) \) is represented by a bit vector \( s = (s_C, s^C) \), composed of two subvectors encoding respectively \( l \) and \( u \). Let \( \text{lit} : \{0, 1\} \times PV \rightarrow \mathcal{F} \) be a function defined as follows: \( \text{lit}(0, p) = \neg p \) and \( \text{lit}(1, p) = p \).

\(^2\) If the system considered consists of \( n \) automata, each part of the vector can be divided into \( n \) subvectors, each of which represents respectively the location and the valuation of the local clocks for the \( i \)-th component, for \( i = 1, \ldots, n \).
Locations: if $|L_i|$ is the number of locations in $T_A$, then $m_i = \lceil \log_2(|L_i|) \rceil$ state variables suffice to encode every location. The subvector $w^C_i = (w^C_1[i], \ldots, w^C_m[i])$ encodes the locations of the $i$-th component. The vector $w^C = (w^C_1, \ldots, w^C_m)$ is of length $m = \sum_{i=1}^n m_i$. We define by $w^C(a)$ the subvector composed of variables $\bigcup_{i \in \Sigma(a)} w^C_i$.

Clocks: a valuation $v \in D$ of a clock $x \in X$ is represented by a pair of natural numbers $(I_x, F_x)$, such that $v = I_x + F_x/\Delta$. It is sufficient to encode $I_x$ and $F_x$ only, such that $w^f$ consists of $n_X$ subvectors $w^f_1$ and $n_X$ subvectors $w^f_2$ having $r^f = \lceil \log_2(2c_{\text{max}} + 2) \rceil$ and $r^F = \lceil \log_2(2n_X) \rceil$ each, and representing $I_x$ and $F_x$, respectively. Thus, each clock is encoded by $r^f + r^F$ state variables, and thus $w^f$ consists of $r = n_X \cdot r_X$ state variables. A discretized clock valuation $(v_1, \ldots, v_{n_X})$ is encoded by $w^f = (w^f_1, w^f_{F_1}, \ldots, w^f_{I_{n_X}}, w^f_{F_{n_X}})$

Next, we introduce the propositional formulas $I_q(w)$ and $T(w, v)$ encoding a discretized state $q$ and the transition relation of $\mathcal{M}(T_A)$ (see [13] for the details).

The formula $I_q(w) = 1 \iff A_q(w[i]) = q$. The formula $T(w, v)$ is such that for each two states $q, q' \in (L \times \mathbb{N}^n)$ and for every assignment $A$ encoding them over $w$ and $v$ (i.e., $A_q(w) = A(w)$ and $A_{\neg q}(v) = A(v)$) we have $q \implies q' \iff A(T(w, v)) = 1$. In order to implement $T(w, v)$ the clock constraints of $T_A$ are encoded over $w^f$. For each constraint $c$, we use $\text{Book}(cc)$ to denote the encoding of $cc$ over the vector $w^f$.

3.4 Characterizing temporal formulas

The standard fixpoint characterisation [4] of $\mathcal{CTL}$ is used. Given a $\mathcal{CTL}$ formula $\varphi$, we compute a corresponding propositional formula $[\varphi](w)$ which encodes the states of the system that satisfy $\varphi$.

Definition 6 (Translation) Given a $\mathcal{CTL}$ formula $\varphi$, the translation $[\varphi](w)$ is inductively defined as follows:

- $[p](w)$ is a formula such that we have $q \models p \iff A_q([p](w)) = 1$, for every $q \in Q$.
- $[\neg \varphi](w) := \neg [\varphi](w)$.
- $[\varphi \lor \psi](w) := [\varphi](w) \lor [\psi](w)$.
- $[AX\varphi](w) := \forall v. (T(w, v) \implies [\varphi](w \leftarrow v))$.
- $[AG\varphi](w) := f_{ssm_AG}([\varphi](w))$.

The formula $\varphi$ is satisfied in the initial state of $\mathcal{M}$ iff the propositional formula $[\varphi](w) \land I_{q_1}(w)$ is satisfiable.

For the sake of brevity, we do not present the algorithm $f_{ssm_AG}([\varphi](w))$. Notice the usage of the restriction in $f_{ssm_AG}([\varphi](w))$, where it ensures that in every fixpoint iteration, it suffices to consider transitions from states not computed in the previous iterations.

4 Generalized blocked clauses in forall()

We have implemented the original algorithm $forall(\alpha, U)$ of [7], where the blocking clauses are built over propositions of the formula $\alpha$. Our experiments show that the problems mentioned in [7] concerning efficiency actually happen. The major problem diagnosed concerns the number of blocking clauses, which are generated by exploring an Alternative Implication Graph. It seems that this approach usually works for simple formulas, but in the case of these resulting from UMC it leads to generating clauses of the maximal length – what means that the cost spent for constructing and maintaining this graph does not improve the overall performance.

Consider the formula $[\forall U, \alpha](w)$ to be computed by calling $forall(\alpha, U)$. The main idea of our paper, based on [12], consists in constructing blocking clauses not only over the propositions from $PV(\alpha)$, but over an extended set of literals $PV^C(\alpha)$ corresponding to all the subformulas of $\alpha$. Next, we give more details of our solution. After Algorithm 1 has reached a satisfying assignment $A_\alpha$, the procedure $DFS_{forall-opt}(\alpha, U, A_\alpha)$ is executed (Algorithm 3) and performs the depth-first-search through $DA(\alpha)$. It begins with the root $\alpha_0$ and returns a set of literals $L \subseteq PV^C(\alpha)$. The resulting clause $c_\alpha$ is a disjunction of the literals from $L$ negated with respect to the current assignment $A_\alpha$. Notice that the smaller the set $U$, the shorter $c_\alpha$ is. In case $U = PV(\alpha)$, no optimization is achieved. Formally, $c_\alpha = genBlockingCl(L, A_\alpha)$, where

$$genBlockingCl(L, A_\alpha) = \bigvee_{\beta \in L} l_{\beta}^l,$$

with $l_{\beta}^l = \neg l_{\beta}$ if $A_\alpha(\beta) = 1$ and $l_{\beta}^l = l_{\beta}$ if $A_\alpha(\beta) = 0$. It is shown in [12] that $c_\alpha$ is a blocking clause.
code the locations of the components not participating in the blocked transition (to be defined below). The second optimization is based on the explicit computation of the time zones generalizing the single valuation of the clocks of the blocked transition. These two methods are orthogonal and for the clarity of the presentation are presented separately, applied one by one. However, the algorithm implemented uses both at the same time for the efficiency reasons.

Let \( A_\alpha(w, v) \) be a blocking assignment, i.e., an assignment for which \( \alpha AX[w, v] \) evaluates to 0. Since \( \alpha AX((\alpha AX)[w, v]) = 0 \) and \( \alpha AX[w, v] \) is an implication, the formula \( T[w, v] \) is true in \( A_\alpha \) and it determines a transition in the model. Recall that \( A_\alpha(T[w, v]) = 1 \) implies that for the states \( q_a \) and \( q'_a \) such that \( A_q_a(w) = A_\alpha(w) \) and \( A_{q'_a}(v) = A_\alpha(v) \) there is a transition \( t = q_a \overset{a}{\longrightarrow} q'_a \) for some \( a \in \Sigma \). The transition \( t \) is called the blocked transition whereas \( a \) - the blocking action (for \( A_\alpha \)).

In order to show correctness of our optimizations we formulate two conditions \( C1 \) and \( C2 \) (below), which if satisfied by the clauses of \( \chi(w) \) returned by the optimized forall(\( C \)), then they guarantee the clauses to be blocking clauses and this way \( \chi(w) \) to properly characterize \( [AX\varphi](w) \).

Definition 7 Consider an optimized algorithm forall() called for the formula \( \alpha AX[w, v] \) and the variables of \( v \). For the clause \( c_b \) generated by the algorithm for a blocking assignment \( A_\alpha(w, v) \), define the following two conditions:

\( C1: \) for each state \( q \) such that \( A_q(c_b(w)) = 0 \), there is a state \( q' \) such that \( q \overset{a}{\longrightarrow} q' \) and \( A_{q'}([\varphi](w)) = 0 \).

\( C2: \) \( A_{q_a}(c_b(w)) = 0 \).

The following theorem shows that \( C1 \) and \( C2 \) guarantee that each clause \( c_b(w) \) is a blocking clause and thus taking into account that the algorithm terminates when there are no more blocking assignments, the formula \( \chi(w) \) correctly characterizes \( AX\varphi \).

Theorem 5.1 Given is a network of timed automata and a temporal formula \( \varphi \) characterized by \([\varphi](w) \). Let \( \chi(w) \) be the result formula computed by forall(\( \alpha AX(w, v) \), \( v \)) and composed of the generalized blocking clauses. If each clause \( c_b \) of \( \chi(w) \) satisfies the condition \( C1 \) and \( C2 \), then \( A_q(\chi(w)) = 0 \) iff \( q \not\models AX\varphi \), for each state \( q \) of the model \( M(TA) \).

Proof 1 \((\Rightarrow)\) If \( A_q(\chi(w)) = 0 \), then there is \( c_b \) in \( \chi \) such that \( A_q(c_b(w)) = 0 \). Then, by \( C1 \) there is \( q' \) such that \( q \overset{a}{\longrightarrow} q' \) and \( A_{q'}([\varphi](w)) = 0 \). This implies that \( q \not\models \varphi \).

\((\Leftarrow)\) Assume that \( q \not\models AX\varphi \). Since the algorithm has terminated, there are two cases to consider.

Case 1. A blocking assignment \( A_\alpha(w, v) \) was found such that it agrees with \( A_q(w) \) on \( w \). Then, \( \chi \) contains a
clause \( c_b \) such that \( A_q(c_b) = 0 \) (due to C2). So, we have \( A_q(\chi(w)) = 0 \) (because \( \chi \) is the conjunction of the blocking clauses).

Case 2. No blocking assignment \( A_a(w, v) \) was found such that it agrees with \( A_q(w) \) on \( w \). Then, since the algorithm has terminated (which follows from C2), it must have been generated a blocking clause \( c_q'(w) \) such that \( A_q(c_q'(w)) = 0 \) as otherwise there would be another blocking assignment found (for example such that it agrees with \( A_q(w) \) on \( w )\). If \( A_q(c_q'(w)) = 0 \), then, clearly, we have \( A_q(\chi(w)) = 0 \).

Notice that for the simplest blocking clause composed of all the state variables of \( w \) (i.e., \( c_b = genBlockingCl(w, A_a) \)), both the conditions C1 and C2 are satisfied. Now we give our optimized algorithms.

The general framework for optimizations is the procedure \( blocking_timed_clause() \), which consists of five major steps executed consecutively in \( forall() \) after finding a blocking assignment \( A_a \). First, the blocking action is identified. Then, the input formula is searched (Alg. \( DFSforall_time_opt() \)). The search identifies the sets \( L_0, L_1, \) and \( L_2 \) of subformulas over the subvectors \( vC(a), vC \backslash vC(a) \), and \( v^t \), respectively. Finally, the location sub-clause is calculated on the basis of the set \( L_1 \), and the timed subclause is computed using the set \( L_2 \). More details of the construction are given below.

Algorithm 4 \( blocking_timed_clause(\alpha_{\chi}(w, v), v, A_a) \)

1. Determine a blocking transition \( t = q_0 \xrightarrow{a} q_0' \) and a blocked action \( a \), where \( q_0 = (l_a, v_{\alpha}) \) and \( q_0' = (l_a', v_{\alpha}') \).
2. Search the formula \( \chi[w \leftarrow v] \); \( (L_0, L_1, L_2) = DFSforall_time_opt(\chi[w \leftarrow v], v, A_a) \).
3. Compute the control part \( c_b^{CONTR}(w^C) \) using \( a \) and \( L_1 \).
4. Compute the timed part \( c_b^{DBM}(w^t) \) using \( a \) and \( L_2 \).
5. Return \( c_b(w) = c_b^{CONTR}(w^C) \lor c_b^{DBM}(w^t) \).

5.1 Restricting the range of quantification

The first optimization generalizes the location part of each blocking clause. As networks of timed automata use the asynchronous semantics determining the behavior with respect to action transitions, the optimization introduced in [12] for untimed systems can be applied here. The idea consists in restricting the range of quantification to a subvector encoding the location part.

Algorithm 5 \( DFSforall_time_opt(\alpha, v(a), A_a) \)

1. s.push( \( q_0 \) )
   while s not empty do
   \( v_B = s.pop(); \)
   if ( \( \beta \) encodes a constr. \( cc \in \mathcal{C}_X \), i.e., \( \beta = \text{Bool}(cc) \) ) then
   \( A_\alpha(\beta) = 0 \)
   \( L_2 = L_2 \cup \{ l_{cc} \} \)
   else
   \( L_2 = L_2 \cup \{ l_{\sim cc} \} \)
   \( \) if \( PV(\beta) \lor v(a) = \emptyset \) then
   \( L_1 = L_1 \cup \{ l_{\beta} \} \)
   else if \( PV(\beta) \subseteq v(a) \) then
   \( L_0 = L_0 \cup \{ l_{\beta} \} \)
   else if \( (\beta = \beta_1 \land \beta_2) \land A_\alpha(\beta) = 0 \) /* the optimisations */ do
   s.push(\( v_B \))
   return (\( L_0, L_1, L_2 \))

Thus, for each action we restrict the range of quantification in order to exclude the variables unchanged by this action. The set \( L_{\text{pre}} \) contains the literals of \( I_{q_0}(w) \) (recall that this formula is a conjunction) occurring in the set \( \bigcup_{l \in \Sigma(a)} \mathcal{W}_l \) (thus it decodes the location predecessor of \( a \)). As before, the clock part encodes \( v_{\gamma} \); \( c_b^{DBM}(w^t) = genBlockingCl(w^t, A_a) \). Let denote by \( L_1(v \leftarrow w) \) the set of formulas such that for every \( \alpha \in L_1 \), it contains \( \alpha(v \leftarrow w) \). The location part is built of the formulas in \( L_1 \) and \( L_{\text{pre}} \); \( c_b^{CONTR}(w^C) = genBlockingCl(L_1(v \leftarrow w) \cup L_{\text{pre}}, A_a) \).

Lemma 5.2 Each blocking clause \( c_b(w) \) generated by the optimized algorithm \( forall() \) satisfies the conditions C1 and C2.

Proof 2 C1: Consider a state \( q = (l, v) \) blocked by \( c_b \). First we prove that there is a transition enabled in \( q \). Let \( A_a(w, v) \) be the blocking assignment for \( c_b(w) \), and \( t = q_0 \xrightarrow{a} q_0' \) be the blocking transition for \( A_a(w, v) \). Note that the locations of \( l \) in the components of \( \Sigma(a) \) are the same as in \( l_{q_0} \) (as they are explicitly encoded in \( c_b \)). Moreover, \( v = v_{q_0} \) as this clock valuation is also encoded in \( c_b \). This guarantees that \( a \) is enabled in the components of \( \Sigma(a) \). Concerning the remaining components, because no upper invariants are present, nothing can prevent time flowing in their locations. Thus we have that \( (l, v) \xrightarrow{a} (l', v') \) for some \( l' \in L \) and \( v' = v_{q_0}' \).

Next, we prove that \( A_q(\chi[w]) = 0 \). Recall that the variables in the sets \( L_0, L_1, \) and \( L_2 \) assigned as in \( A_a \) imply
that $A_\alpha([\varphi])(v) = 0$ (because the formula search of $[\varphi]$ identified these sets to ensure this). The assignments of the corresponding variables over $w$ and $v$ in $L_2$ are the same in $A_\alpha(v)$ and $A_q(w)$, and the same holds true also for $L_1$ but with possibly different locations encoded over $w$ and $v$ (these encodings imply however the same values of the variables in $L_1$). Because $v' = v'_{\alpha}$, the variables in $L_2$ have the same values in $A_\alpha(v)$ and $A_q(w)$ as well.

$\mathbf{C2:}$ As every literal of $c_0(w)$ is false in $A_\alpha$, we have $A_{\alpha_0}(c_0(w)) = A_\alpha(c_0(w)) = 0$.

5.2 Efficient predecessor calculation

We have described sets of locations by single blocking clauses, but still only one clock valuation is encoded in every clause. Now, we show how to compute timed subclauses by operating on time constraints. Our approach is based on Difference Bound Matrices (DBMs), which are an efficient representation of time constraints. The search for generalized subformulas is extended to subformulas encoding constraints over clock variables; these constraints are then transformed to a zone and the computations are performed using DBMs. Finally, the resulting constraints are encoded in the propositional logic and added directly to the blocking clause.

First, we define the operations on zones that we use later. Let $v, v' \in \mathbb{R}_{\mathbb{N}}^n$ and $Z, Z' \in (\mathbb{N}_\mathbb{Z})$. Let $v \preceq v'$ iff $\exists 0 \in \mathbb{R}_+$ s.t. $v' = v + \delta$. The following operations are defined:

1. $Z \cap Z' = \{ v \in Z \mid v \in Z' \}$ (intersection of zones),
2. $Z \not\subset \{ v' \in \mathbb{R}_{\mathbb{N}}^n \mid (\exists v \in Z) v' \leq v \}$ (time predecessor),
3. $[X = 0]Z = \{ v \mid v[X = 0] \in Z \}$ (clock reset inverse).

All the above operations preserve zones [9].

We use the standard form of normalized constraints. The set $X$ is extended with an additional ficticious clock $x_0 \notin X$, which represents the constant 0. The set $X \cup \{x_0\}$ is denoted with $X^+$. Then, each constraint $cc'$ over $X^+$ can be generated by the following grammar: $cc' = x_i \leq x_j \sim c \mid x_i \leq x_j \sim \infty \mid x_i \leq x_j \sim \infty \mid cc' \wedge cc'$, where $x_i, x_j \in X^+$, and $\sim \in \{<, \leq\}$. The standard conversion of the constraints in $C_\mathbb{Z}$ to the normalized form is described in [9]. Now, we formally introduce DBMs:

Definition 8 (Difference Bounds Matrix) A difference bounds matrix (DBM) in $\mathbb{R}_{\mathbb{N}}^n$ is a $(n, n + 1) \times (n, n + 1)$ matrix of bounds, with rows and columns indexed from 0 to $n$. The DBM $D = (d_{ij})$, where for each $i, j \in \{0, \ldots, n\}, d_{ij} = (d_{ij}, \sim_{ij})$, represents the zone $Z = \{x \in \mathbb{R}_{\mathbb{N}}^n \mid x_i \sim_{ij} d_{ij}\}$. The zone represented by $D$ is denoted by $[D]$.

It is easy to see that for each zone $Z$ there exists a DBM $D$ such that $Z = [D]$. In general, the space requirements for representing a zone with $n \cdot \mathbb{N}$ clocks is $O(n \cdot \mathbb{N})$, but for some zones not all the constraints must be present. Some zones can be represented by more than one DBM, but the canonical representations are provided by canonical DBMs. It is assumed that an implementation of DBMs is available together with the algorithms to calculate union, clock resets, time precondition, and the canonical form. The details can be found in [9].

Now, we are ready to define generalized timed subclauses. Recall that $i$ is a blocked transition, $a$ - the blocked action (labelling the blocking transition), and $L_2$ - the subformulas of $\alpha_{AX}(w, v)$ over $v^i$. Next, let $L'$ be a set of the constraints encoded by the literals in $L_2$, defined as $L' = \{cc \mid l_{\mathbb{B}oot(cc)} \in L_2\}$. Then, we build the zone $Z'$ which is constructed from the tightestest constraints of $L'$. Formally, we introduce the ordering $\preceq$ on the constraints as follows: $\preceq$ is strictly less than $\not\subset$ and for $cc = x_i \preceq x_j \sim c$ and $cc' = x_i \preceq x_j \sim c'$ we have $cc \preceq cc'$ if either $c < c'$ or $c = c'$ and $\preceq \leq c'$. We define $Z'$ to contain all the minimal constraints from $L'$. Then, let $D'$ be the canonical DBM for $Z'$. Next, we calculate the zone $Z$ being the predecessor of $Z'$ with respect to the action $a$:

$$Z = (\bigcap_{i \in \Sigma(a)} [I(i,l)]) \cap ([cc_0 \cap] Y = 0)(Z' \cap (\bigcap_{i \in \Sigma(a)} [I(i,l)])) \not\subset$$

The timed subclause encodes the constraints of $Z$: $L_Z = \{cc \mid cc \in Z' \}$ and $c_{DBM}(w) = genBlockingCl(L_Z, A_\alpha)$, and the location subclause is the same as the first optimization: $c^{CONTR}(w^{c}) = genBlockingCl(L_1(w^{c} \rightarrow w) \cup L_{pre}, A_\alpha)$.

Lemma 5.3 The blocking clause $c_0$ generated by the above algorithm satisfies the conditions $C1$ and $C2$.

Proof 3 $C1$: Consider a state $q = (l, v)$ blocked by $c_0$. First we prove that there is a transition enabled in $q$. Let $A_\alpha(w, v)$ be the blocking assignment for $c_0(w)$, and $t = q_a \preceq q_{\alpha_0} q_0'$, where $q_0 = (l_0, v_0)$ and $q_{\alpha_0} = (l_0, v_{\alpha_0})$, be the blocking transition for $A_\alpha(w, v)$. Note that the locations of $l$ in the components of $\Sigma(a)$ are the same as in $l_0$ (as they are encoded in $c_0$). Moreover, recall that the zone $Z$ is encoded in $c_{DBM}(w)$, and $c_0(w)$ is false both in $A_\alpha$ and $A_{DBM}$ is also false in these assignments and we have $v, v_{\alpha} \in Z$. The zone $Z$ was calculated so that for every $v_\alpha \in Z$ there was some $v'_{\alpha} \in Z'$ such that $(l_0, v_\alpha) \preceq q^\rightarrow (l', v'_\alpha)$. So we have $q_a \preceq q'_{\alpha_0} q' = (l', v')$ and some $v' \in Z'$.

Next, we prove that $A_{\alpha_0}([\varphi](w)) = 0$. Recall that the variables in the sets $L_0, L_1, L_2$ assigned as in $A_\alpha$ imply that $A_\alpha([\varphi](v)) = 0$ (the formula search of $[\varphi]$ identified
6 Case study: Fischer’s mutual exclusion

The optimized algorithm has been implemented using the representation of the propositional formulas and the encoding of the transition relation of the module BMC of VerICS [9], and the SAT solver ZChaff. In order to evaluate the performance, we examine the well-known Fischer’s Mutual Exclusion protocol in its standard formulation. The example models a system consisting of \( n \) independent processes \((1, \ldots, n)\) and a controlling process. The processes indexed with 1, \ldots, \( n \) compete for an access to the shared resource (indexed with 0), which can be used by a single process at the same time.

The experimental results comparing VerICS UMC to RED and UPPAAL are shown in Table 1. Notice that UMC clearly performs best when \( \delta > \Delta \). This can be briefly explained as follows with the help of Fig. 2. The tested property consists in reachability of a state where two processes are in their critical sections. Let’s consider a pair of the processes \((1, 2)\), but all the pairs are processed in the same way. In the first iteration, \((l_1^1, l_2^1)\) is the blocking transition (the case of \( enter_2 \) is symmetric). In the second iteration, the instances of the action \( set_1 \) are blocked, for each location of \( TA_0 \). Then, in the third iteration, there are two blocking transitions: in the first one \( TA_1 \) still remains in its critical section while \( TA_1 \) executes \( try_1 \) – notice that \( exit_1 \) will not become enabled because it leads from the state in which the property is false. Consider the second transition \( enter_2 \). Note that no clock valuation with \( v_2 = 0 \) belongs to the corresponding zone \( Z \). This means that the action \( set_2 \) will not become enabled and the computation halts.

Notice that the performance of our tool degrades with the increase of the parameter values, what can be explained by our inefficient and preliminary implementation where all the possible constraints are first generated. In a lazy implementation only necessary constraints, currently present in the working formula, would be generated on-the-fly. Notice also that our method performs worst when the mutual exclusion is violated. However, then a counterexample exists, and all the methods presented would be outperformed by the BMC algorithm of VerICS.

7 Future work

Our experimental results are promising, but a lot of work is still necessary to get a reliable tool. The major problem concerns the representation of propositional formulas, which is not canonical. It should be possible to optionally use BDD graphs as a formula representation, what would significantly improve the performance. Contrary to the current representation, adding blocking clauses would reduce

\[ \phi^+ = EF \left( \bigvee_{1 \leq i, j \leq n, i \neq j} crit_i \land crit_j \right) \]

**Table 1. The results of checking \( \phi^+ \) for Fischer’s mutual exclusion.**

<table>
<thead>
<tr>
<th>System</th>
<th>Uppaal</th>
<th>RED</th>
<th>VerICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n = 12, \Delta = 1, \delta = 2 ) (true)</td>
<td>580</td>
<td>304</td>
<td>59</td>
</tr>
<tr>
<td>( n = 13, \Delta = 1, \delta = 2 ) (true)</td>
<td>-</td>
<td>657</td>
<td>88</td>
</tr>
<tr>
<td>( n = 15, \Delta = 1, \delta = 2 ) (true)</td>
<td>-</td>
<td>-</td>
<td>154</td>
</tr>
<tr>
<td>( n = 18, \Delta = 1, \delta = 2 ) (true)</td>
<td>-</td>
<td>-</td>
<td>376</td>
</tr>
<tr>
<td>( n = 20, \Delta = 1, \delta = 2 ) (true)</td>
<td>-</td>
<td>-</td>
<td>491</td>
</tr>
<tr>
<td>( n = 10, \Delta = 3, \delta = 4 ) (true)</td>
<td>33</td>
<td>53</td>
<td>50</td>
</tr>
<tr>
<td>( n = 11, \Delta = 3, \delta = 4 ) (true)</td>
<td>125</td>
<td>133</td>
<td>71</td>
</tr>
<tr>
<td>( n = 10, \Delta = 2, \delta = 1 ) (false)</td>
<td>7</td>
<td>49</td>
<td>97</td>
</tr>
</tbody>
</table>

\( ^9 \)The site of VerICS is verics.ipipan.waw.pl
the formula size. Another important aim is to relax the restriction of the upper invariants. This is easy provided an efficient representation is available. Tuning the SAT algorithm would result in a faster search.

We conjecture that after adding the features described above, the algorithm would become a part of a standard model-checking toolset, complementary to other symbolic methods.

References


8 Appendix

8.1 Example

A simple calculation is shown for the system given in Fig. 3, composed of two timed automata TA1 and TA2. The verified property is AG(p1 ∨ p2). There are two clocks x and y, and clock valuations are of the form u = (x, y).

The computation consists of the two fixpoint iterations, and during every computation a single blocking clause is generated:

1. The blocking assignment A1 = {x < 1} is found which determines the states q0 = ((1, 1), u) and q′ 0 = ((1, 1), u′), with v = (1, 4) and v′ = (0, 6). The algorithm DFSf or all_time_opt() finds the following sets: L0 = {[p2](v)}; L1 = {[p1](v)}, L2 = ∅. Thus Z′ = true and we compute Z = x < 3. Finally, c3 = ¬pre(a1)(wC) ∨ ¬[p1](wC) ∨ ¬l_{x<3}(w′).

2. The second blocking assignment, A2 = {x < 3} determines the states q′ 0 = ((1, 2), v) and q′ 0 = ((1, 2), v′), with v = (1, 2) and v′ = (4, 5). The algorithm DFSf or all_time_opt() finds the following sets: L0 = {¬pre(a1)(v)}, L1 = {[p1](v)}, L2 = {x < 3}. The searched formula is shown in Fig. 3, notice that the subformula [v] is skipped as the another one argument of the conjunction suffices to imply the false value. Finally, we get c3 = ¬pre(a2)(wC) ∨ ¬[p1](wC) ∨ ¬l_{x<1}(w′).

The result of the computation is [AG(p1 ∨ p2)](w) = αQ = [p1 ∨ p2] ∧ c3 ∧ c′ 3. This formula conjuncted with Ip_q (w) is not satisfiable, what means that the verified property is not true in the model.