SAT-based Unbounded Model Checking of Timed Automata

Wojciech Penczek and Maciej Sreter
Institute of Computer Science, PAS

Abstract. We present an improvement of the SAT-based Unbounded Model Checking (UMC) algorithm. UMC, a symbolic approach introduced in [9], 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 build blocking clauses of 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 the 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, many very efficient algorithms testing satisfiability have been designed recently. Several verification problems are translated to satisfiability of propositional formulas. Bounded Model Checking (BMC) seems to be the state-of-the-art SAT-based model checking method [12]. Some types of errors can be easily found in very large systems. Despite these well-known advantages, BMC has some weak points. It is still rather a method of falsification than validation of timed systems. 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) [8] is an obvious analogue. BDD-based model checking constitutes a well developed branch of symbolic verification methods. Unbounded Model Checking (UMC) [9] 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 [7], it seems that the performance of the algorithm is still inferior to other symbolic approaches based on BDDs. In the conclusions of [9], two major problems are stated:

- the formulas encoding the whole state space are represented by semi-canonical Directed Acyclic Graphs. This representation can be much less concise than BDD in the case of 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 focus on the second issue. In [9], 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.

Related Work A similar idea of generalizing blocking assignments uses circuit cofactoring [6] but does not consider timed automata. In [4] a BDD-based algorithm for verification of a restricted class of timed automata is described. Several variants of BDDs capable of representing time constraints were developed [3, 10]. The symbolic verification of timed systems where the quantifiers are eliminated using a BDD-based algorithm is explored in [13]. The principles of the SAT solvers can be found in [12].

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: $\lor$ - disjunction and $\land$ - negation, with the derived operators $\land$ - conjunction and $\Rightarrow$ - implication. Let $\mathcal{F}$ denote a set of all the propositional formulas. For each propositional formula $\alpha$ a set of its subformulas $Subform(\alpha)$ is defined in the usual way. A literal is a variable of $PV$, or its negation. A clause $c$ is a disjunction of zero or more literals $l_1 \lor \cdots \lor l_n$. By $C$ we denote a set of 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 $A : PV \to \{1, 0\}$, where 1 stands for $true$ and 0 stands for $false$. An assignment is said to be total when its domain equals to $PV$. A total assignment is said to be 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 $p \in dom(A)$ such that $A(p) = 0$ and $p$ for all $p \in dom(A)$ such that $A(p) = 1$.

Following [1], we represent propositional formulas by directed acyclic graphs (DAGs), with the graph $DAG(\alpha)$ 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 $SAT()$ refer to a generic SAT-solver, which given a formula either returns its satisfying assignment or diagnoses that no such assignment exists.

Conversion to CNF Most of the SAT-solvers accept formulas in CNF. For $\alpha \in \mathcal{F}$, let $PV(\alpha) \subseteq PV$ denote the set of variables used in $\alpha$ and $PV^C(\alpha) = \{ l_3 \in PV \mid \beta \in Subform(\alpha) \}$ be a set of the literals corresponding to the subformulas of $\alpha$.

The standard translation of propositional formulas to CNF is called $toCNF()$ [11]. For each formula $\alpha$, $toCNF(\alpha)$ returns the formula in CNF defined over variables of $PV^C(\alpha)$. Every subformula $\beta$ of $\alpha$ is represented by the literal $l_3 \in PV^C(\alpha)$, and for every assignment $A$ such that $A(toCNF(\alpha)) = 1$, we have $A(l_3) = A(\beta)$. Consequently, the CNF formula $toCNF(\alpha) \land l_\alpha$ is satisfiable iff $\alpha$ is satisfiable. This fact is commonly exploited for testing satisfiability. Moreover, the formula $\alpha$ is valid when the CNF formula $\beta = toCNF(\alpha) \land l_{\neg \alpha}$ is unsatisfiable, which is used for eliminating universal quantifiers. More details can be found in [11].
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:

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

The semantics of the quantifiers is as follows: \( \exists p. \alpha \equiv \alpha(p \leftarrow \text{true}) \lor \alpha(p \leftarrow \text{false}), \) and \( \forall p. \alpha \equiv \alpha(p \leftarrow \text{true}) \land \alpha(p \leftarrow \text{false}), \) where \( p \in PV \) and \( \alpha(p \leftarrow q) \) denotes a substitution with the variable \( q \) of every occurrence of the variable \( p \) in \( \alpha. \) For a vector of variables \( v = (v[1], \ldots, v[m]) \), we use the notation \( \forall v. \alpha, \) to denote \( \forall v[1] \ldots \forall v[m]. \alpha, \) 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 \( SAT() \) can be used for removing universal quantifiers [9] from a QBF formula. In Alg. 1 a pseudo-code of the algorithm \( forall() \) is shown; \( forall(\alpha, U) \) returns a propositional CNF formula equivalent to \( \forall U. \alpha. \) It is based on the fact that in a CNF formula equivalent to the input formula, each clause must be satisfied for any assignment of the quantified variables for which the input formula is satisfied. Thus, 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 \( forall() \) (Alg. 1). A satisfying assignment \( A_\alpha \) for \( \beta, \) found by \( SAT(\beta), \) is called a blocking assignment. A blocking clause \( c_b \) for \( A_\alpha \) is a clause over the set of variables \( PV(\alpha) \) with the following two properties: (i) \( A_\alpha(c_b) = 0, \) and (ii) \( \alpha \Rightarrow c_b. \)

**Algorithm 1** procedure \( forall(\alpha, U) \)

1: \( \chi = \emptyset, \beta = toCNF(\alpha) \land l_\alpha \)
2: while \( A_\alpha = SAT(\beta) \neq UNSAT \) do
3: compute the blocking clause \( c_b \)
4: for each \( p \in U, \) remove literals \( p \) and \( \neg p \) from \( c_b \)
5: \( \chi = \chi \land c_b, \beta = \beta \land c_b \)
6: return \( \chi \)

**Theorem 1.** \( \beta \) When the formula \( \beta \) becomes unsatisfiable in Algorithm 1 (the condition in line 2 is false), \( \chi \) is a propositional formula in CNF equivalent to \( \forall U. \alpha. \)

**Quantifier elimination under a restriction** In symbolic model checking some operations are considered under a restriction, i.e., a propositional formula \( \beta \) describing a restriction is given and the valuations satisfying the resulting formula should also satisfy \( \beta. \) In \( forall(), \) we can evaluate \( \left( \forall U. \alpha \right) \land \beta \) by substituting \( toCNF(\alpha) \) with \( toCNF(\alpha) \land toCNF(\beta) \land l_\beta \) in line 1. Then, the algorithm considers only assignments that make \( \alpha \) false but \( \beta \) true. The restriction is used in fixpoint computations in UMC, as it 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 [2], the flow of time is modeled by means of clocks. From a semantic viewpoint the duration of actions is equal to zero and the time flows when no action is taken. By \( \mathcal{X} \) we denote a finite set \( \{x_1, \ldots, x_n\} \) of variables, called clocks. A clock constraint over \( \mathcal{X} \) is defined by the following grammar: \( \psi = \text{true} \mid x_i \sim c \mid x_i - x_j \sim c \mid \psi \land \psi \), where \( x_i, x_j \in \mathcal{X}, c \in \mathbb{N}, \text{ and } \sim \in \{\leq, <, =, >, \geq\} \). The constraints of the form \( \text{true} \), \( x \sim c \) and \( x_i - x_j \sim c \) are called atomic. \( C_0^\mathcal{X} \) is the set of clock constraints over \( \mathcal{X} \). Let \( C_0^\mathcal{X} \) denote the restricted set of clock constraints \( C_0^\mathcal{X} \) without inequalities involving clock differences.

The function \( v : \mathcal{X} \to \mathbb{R}_+ \), assigning to each clock \( x \) a positive value \( v(x) \) is called a clock valuation. Let denote by \( \mathbb{R}_+^\mathcal{X} \) the set of all valuations. For simplicity, we assume a fixed ordering on \( \mathcal{X} \). For a valuation \( v \) and \( \delta \in \mathbb{R}_+ \), \( v + \delta \) denotes the valuation \( v' \) s.t. for all \( x \in \mathcal{X}, v'(x) = v(x) + \delta \). Moreover, for a subset of clocks \( X \subseteq \mathcal{X}, v[X = 0] \) denotes the valuation \( v' \) such that for all \( x \in X, v'(x) = 0 \) and for all \( x \in \mathcal{X} \setminus X, v'(x) = v(x) \). For \( v \in \mathbb{R}_+^\mathcal{X} \), the satisfaction relation \( \models \) for a clock constraint \( \psi \in C_0^\mathcal{X} \) is defined inductively as follows: \( v \models \text{true} \), \( v \models (x_i \sim c) \) iff \( v(x_i) \sim c \), \( v \models (x_i - x_j \sim c) \) iff \( v(x_i) - v(x_j) \sim c \), \( v \models (\psi \land \psi') \) iff \( v \models \psi \) and \( v \models \psi' \). For a constraint \( \psi \in C_0^\mathcal{X} \), let \( [\psi] \) denote the set of all the clock valuations satisfying \( \psi \), i.e., \( [\psi] = \{v \in \mathbb{R}_+^\mathcal{X} \mid v \models \psi\} \). By a (time) zone in \( \mathbb{R}_+^\mathcal{X} \) we mean each convex polyhedron \( Z \subseteq \mathbb{R}_+^\mathcal{X} \) defined by a clock constraint, i.e., \( Z = [\psi] \) for some \( \psi \in C_0^\mathcal{X} \) (for simplicity, we identify a zone with the clock constraint that defines it).

The set of all the zones for \( \mathcal{X} \) is denoted by \( Z(\mathcal{X}) \).

**Definition 2.** A timed automaton \( TA = (\Sigma, L, l_0, \mathcal{X}, E, \mathcal{T}) \) is a tuple \( (\Sigma, L, l_0, \mathcal{X}, E, \mathcal{T}) \), where \( \Sigma \) is a finite set of actions, \( L \) is a finite set of locations, \( l_0 \in L \) is the initial location, \( \mathcal{X} \) is a finite set of clocks, \( E \subseteq L \times \Sigma \times C_0^\mathcal{X} \times 2^X \times L \) is a transition relation, and \( \mathcal{T} : L \to C_0^\mathcal{X} \) is a state invariant function.

Each element \( e \in E \) is denoted by \( e = (l, v, l') \), which represents a transition from the location \( l \) to the location \( l' \) labelled with an action \( a \); \( Y \subseteq \mathcal{X} \) is a set of clocks to be reset after executing the transition \( e \), while \( \psi \in C_0^\mathcal{X} \) is the guard condition for \( e \).

In order to reason about systems represented by timed automata, for a set of propositions \( PV \) we define a valuation function \( V : L \to 2^{\mathbb{P}V} \), assigning a set of propositions to each location.

**Semantics of timed automata** Let \( TA = (\Sigma, L, l_0, \mathcal{X}, E, \mathcal{T}) \) be a timed automaton. A concrete state of \( TA \) is a pair \( (l, v) \), where \( l \in L \) and \( v \in \mathbb{R}_+^\mathcal{X} \) is a clock valuation. The concrete (dense) state space of \( TA \) is a structure \( C(TA) = (Q_c, q_0', \to_c, c) \), where \( Q_c = L \times \mathbb{R}_+^\mathcal{X} \) is the set of all the concrete states, \( q_0' = (l_0', v') \) with \( v'(x) = 0 \) for all \( x \in \mathcal{X} \) is the initial state, and \( \to_c \subseteq Q_c \times (\Sigma \cup \mathbb{R}) \times Q_c \) is the transition relation, defined by the union of the action- and time-successors as follows:

- for \( \delta \in \mathbb{R}_+ \), \( (l, v, \delta) \) if \( v \models \mathcal{T}(l) \) (time successor).
- for \( a \in \Sigma \), \( (l, v, a) \) if \( (\exists \psi \in C_0^\mathcal{X} \mid \exists Y \subseteq \mathcal{X} \) such that \( l = (l', v) \in E, v \in [\psi], v' = v[Y = 0] \) and \( v' \in [\mathcal{T}(l')] \) (action successor).

For \( (l, v) \in Q \) and \( \delta \in \mathbb{R}_+ \), let \( (l, v, \delta) \) denote \( (l, v + \delta) \). A \( \delta_0 \)-run \( \rho \) of \( TA \) is a maximal sequence of concrete states \( \rho = q_0 \to_c q_0 \to_c q_1 \to_c q_1 \to_c q_2 \to_c q_2 \to_c \ldots \), where \( a_i \in \Sigma \) and \( \delta_i \in \mathbb{R} \), for each \( i \geq \mathbb{N} \) (notice that due to the fact that \( \delta \) can be equal to 0 two consecutive transitions can be executed without any time passing in between). A run \( \rho \) is said to be progressive if \( \Sigma \in \mathbb{N} \delta_i \) is unbounded. A timed automaton is progressive if all its runs beginning in the initial state are progressive. We model concurrent system by networks of progressive timed automata. A network of timed automata \( \mathcal{NA} \) is a set of \( n \) timed automata (called components) \( \mathcal{NA} = \{TA_i \mid i \in \{1, \ldots, n\}\} \), where \( TA_i = (\Sigma_i, L_i, l_0, E_i, \mathcal{X}_i, \mathcal{T}_i) \). For \( a \in \Sigma \), let \( \Sigma(a) = \{i \in \{1, \ldots, n\} \mid a \in \Sigma_i\} \) be the set of the indices of all the components including \( a \).
Definition 3. Given a network $\mathcal{N}$, the product of $\mathcal{N}$ is the timed automaton $\mathcal{T} = (\Sigma, L, l^1, E, X, I)$, where $\Sigma = \bigcup_{i \in \{1, \ldots, n\}} \Sigma_i$, $L = \prod_{i \in \{1, \ldots, n\}} L_i$, $l^1 = (l_1, \ldots, l_n)$, $X = \bigcup_{i \in \{1, \ldots, n\}} X_i$, $I_i = (l_i, \ldots, l_i)$, and the transition relation is given by:

\[(l_i, \ldots, l_i), a, l, \xi \in \Sigma \Rightarrow (l_i', \ldots, l_i') \in E_i \text{iff } (\forall i \in \Sigma(a))(l_i, a, \xi, l_i', l_i') \in E_i \text{ and } (\forall i \in \{1, \ldots, n\} \setminus \Sigma(a)) l_i' = l_i.
\]

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. Furthermore, $c_{\text{max}}$ denotes the largest constant occurring in the clock constraints of the automaton.

Abstract Discretized Models

In this section we define an equivalence on clocks and abstract discretized models. For $\delta \in \mathbb{R}_+$, $\frac{\delta}{\delta}$ denotes the fractional part of $\delta$, and $\lfloor \delta \rfloor$ denotes its integral part. Because the concrete space is infinite, an abstraction is introduced [2, 11] in order to identify the equivalent clock valuations:

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

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

Let $\mathcal{T} = (\Sigma, L, l^1, E, X, I)$ be a timed automaton with $n_X$ clocks and $V_{\mathcal{T}}$ a valuation function. Next, let $\mathcal{M}(\mathcal{T}) = (C(\mathcal{T}), V_{\mathcal{T}})$ be the concrete model for $\mathcal{T}$, where $V_{\mathcal{T}}(l, v) = V_{\mathcal{T}}(l)$. As in BMC [15], we choose the discretization step $\Delta = 1/d$, where $d$ is a fixed even number\(^1\) greater than $2n_X$. The discretized clock space is defined as $\mathbb{U}^n$, where $\mathbb{U} = \{2k\Delta \mid 0 \leq k\Delta \leq c_{\text{max}} + 1\}$ for $k \in \mathbb{N}$.

We use an abstract discretized model, which is time-bisimilar with a detailed region graph implementing a time-abstract semantics (action transitions combined with future projection) of [2]:

Definition 5. The (abstract) discretized model of a timed automaton $\mathcal{T}$ is a finite structure $\mathcal{M}(\mathcal{T}) = (Q, q^0, \rightarrow, \rightarrow_c, V_{\mathcal{T}}^c)$, where $Q = L \times \mathbb{U}^n$, $q^0 = (l^1, \mathbb{U})$ and $\rightarrow_c \subseteq Q \times \Sigma \times Q$ is defined as follows:

\[(l, w) \xrightarrow{\alpha} (l, v) \text{ iff } (l, w') \xrightarrow{\delta} (l, v') \text{ for some } \delta \in \mathbb{R}_+ \text{ and some } w', v' \in \mathbb{U}^n \text{ such that } w \simeq w' \text{ and } v \simeq v' \text{ (action transition combined with future projection transition)}.
\]

We use the logic $\mathcal{CTL}$ having the syntax as follows: $\varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid AX\varphi \mid AG\varphi \mid A(\varphi U \varphi)$. $\mathcal{CTL}$ is interpreted in the standard way over $\mathcal{M}(\mathcal{T})$.

Encoding of the transition relation (based on [15]) For the encoding we require that every component $\mathcal{T}_i = (\Sigma_i, L_i, l_i^1, E_i, X_i, I_i)$ of the network $\mathcal{T}$ 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 $\mathcal{T}_i$, 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 encoding. Since the set of states $Q$ of our model is finite, every state $q = (l, u) \in Q$ of $\mathcal{M}(\mathcal{T})$ can be represented as a bit vector $s_q = (s_q[1], \ldots, s_q[n_0])$.

\(^1\) A good choice for $d$ is the minimal such a number, which equals to $2^d$ for some $d'$. 
of length $n_b$ depending on the number of locations in $L$, the size of the set $\mathbb{U}$ and the number of clocks. Consequently, this bit vector can be encoded by a valuation of a vector $w = (w[1], \ldots, w[n_b])$, where $w[i]$, for $i = 1, \ldots, n_b$ is a propositional variable (called state variable). Here, the bit vector $s_q = (s^C_q, s^F_q)$ is composed of two subvectors representing $l$ and $u$ respectively, and is encoded by the vector $w = (w^C, w^F)$. Let $\text{lit} : \{0, 1\} \times PV \to \mathcal{F}$ be a function defined as follows: \[ \text{lit}(0, p) = \neg p \text{ and } \text{lit}(1, p) = p. \]

Concerning locations, if $|L_i|$ is the number of locations in $\mathcal{T}_A$, then $m_i = \lceil \log_2(|L_i|) \rceil$ state variables suffice to encode every location. The subvector $w^C_i$ of $w^C$ encodes the locations of $\mathcal{T}_A$. The vector $w^C = (w^C_1, \ldots, w^C_{n_b})$ of length $m = \sum_{i=1}^{n_b} m_i$. We define $w^C_i(a)$ to be the subvector of $w^C$ composed of $w^C_i$ for $i \in \Sigma(a)$. Concerning clocks, a valuation $v \in \mathbb{U}$ of a clock $x \in \mathcal{X}$ 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, so that $w^v$ consists of $n_X$ subvectors $w^v_i$ and $n_X$ subvectors $w^v_i$ representing $I_x$ and $F_x$, respectively\(^3\). Thus, each clock is encoded by $r_X$ state variables, and the size of $w^v$ is $r = n_X \cdot r_X$. A discretized clock valuation $(v_1, \ldots, v_{n_X})$ is encoded by $w^v = (w_{l_1}, w_{l_2}, \ldots, w_{l_{n_X}}, w_{F_{n_X}})$.

Next, we introduce the propositional formulas $I_q(w)$ and $T(w, v)$ encoding a discretized state $q$ and the transition relation of $\mathcal{M}(\mathcal{T}_A)$ (see [15] for the details). We have $I_q(w) = \bigwedge_{i=1}^{n_b} \text{lit}(s^C_q[i], w[i])$. By $A_q$ we denote the assignment encoding $q$ over $w$, that is $A_q(w[i]) = s^C_q[i]$ for all $1 \leq i \leq n_b$. The formula $T(w, v)$ is such that for each two states $q, q' \in (L \times \mathbb{U}^{n_X})$ and for every assignment $A$ encoding them over $w$ and $v$ (i.e., $A_q(w) = A(w)$ and $A_q(v) = A(v)$) we have $q \overset{A}{\longrightarrow} q'$ iff $A(T(w, v)) = 1$. In order to implement $T(w, v)$ the clock constraints are encoded. For $c \in C^X_{n_X}$, we use $l_c$ to denote the encoding of $c$ over the vector $w^v$.

**Characterizing temporal formulas** The standard fixpoint characterization [5] of CTL is used. Given a 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 CTL formula $\varphi$, the translation $[\varphi]$ is inductively defined as follows:

- $[p](w)$ is a formula such that we have $q \models p$ if and only if $A_q([p](w)) = 1$, for every $q \in Q$, 
- $[\neg \varphi](w) = \neg [\varphi](w)$, 
- $[\varphi \lor \psi](w) = [\varphi](w) \lor [\psi](w)$,

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

---

**Algorithm 2** $fssm_{AG}([\varphi](w))$

```
αZ(w) = αQ(w) = [\varphi](w)
while (αZ ≠ true) do
  αZ(w) = (∀v. (¬T(w, v) ∨ [\varphi](w ← v)))↓
  αQ(w)
  αQ(w) = αQ(w) ∧ αZ(w)
return αQ(w)
```

**Algorithm 3** $lfp_{PAU}([\varphi](w), [\psi](w))$

```
αQ(w) = false, αZ(w) = [\psi](w)
while (¬(αZ ⇒ αQ) ≠ UNSAT) do
  αQ(w) = αQ(w) ∨ αZ(w)
  αZ(w) = ∀v. (¬T(w, v) ∨ [\varphi](w ← v)) ∧ [\psi](w)
return αQ(w)
```

---

\(^3\) 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 $i = 1, \ldots, n$.

\(^4\) Notice that every clock is represented by the same number of bits, irrespectively of its maximal constant (the maximal constant appearing in a constraint with this clock). An optimized encoding would represent every clock with the number of bits depending on the respective constant.
Notice the usage of the restriction in $fssm_{\text{AG}}(\varphi(w))$, where it ensures that in every fixpoint iteration, it suffices to consider transitions from states not computed in the previous iterations.

4 Generalized blocking clauses in UMC

We have implemented the original algorithm for all $(\alpha, U)$ of [9], where the blocking clauses are built over propositions of the formula $\alpha$. Our experiments confirmed the efficiency problems mentioned in [9]. The major problem diagnosed concerns the number of blocking clauses, 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 produces clauses of the maximal length.

Consider the formula $(\forall U.\alpha)(w)$ to be computed by calling for all $(\alpha, U)$. The main idea of our paper, based on [14], 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, first for the general case. In Alg. 1, after reaching a satisfying assignment $A_\alpha$, the procedure $\text{DFS}_{\text{forall-opt}}(\alpha, U, A_\alpha)$ (Alg. 4, without lines 9-13 for now) performs the DFS through $\text{DAG}(\alpha)$. It begins with the root $v_\alpha$ and returns a set of literals $L_1 \subseteq PV^c(\alpha)$ which imply the false value of $\alpha$ when assigned as in $A_\alpha$. The resulting clause $c_\alpha$ is the disjunction of the literals from $L_1$ negated with respect to the current assignment $A_\alpha$. Notice that the smaller the set $U$, the shorter $c_\alpha$ is. In case of $U = PV(\alpha)$, no optimization is achieved. Formally, we have $c_\alpha = \text{genBlockingCl}(L_1, A_\alpha)$, where

$$\text{genBlockingCl}(L, A_\alpha) = \bigvee_{\beta \in L} l'_\beta,$$

with $l'_\beta = \neg l_\beta$ if $A_\alpha(\beta) = 1$ and $l'_\beta = l_\beta$ if $A(\beta) = 0$. It is shown in [14] that $c_\alpha$ is a blocking clause.

Now, we examine an application of for all to computing $\chi(\alpha)(w)$, which involves the elimination of universal quantifiers. Let $\alpha_{AX}(w, v)$ denote the formula $(T(w, v) \Rightarrow \varphi(w \leftarrow v))$. We compute $\chi(w) = \forall U.\alpha_{AX}(w, v, v)$. Unfortunately, the optimization of generalized blocking clauses is likely not to make any improvement there. The reason is that all the state variables of $v$ are quantified, so each blocking clause would describe only one model state. This reduces to describing the discretized states one by one, which in the case of full region graph is not feasible in practice. We specialize the search-based algorithm computing generalized blocking clauses. Two orthogonal optimizations are proposed that generalize formulas over state variables of the location and the timed part, respectively. The first one restricts the universal quantification over $v$ to the state variables of $v^C$, encoding the locations of the components not participating in the blocked transition (to be defined below). The second one is based on the explicit computation of the time zones generalizing the single clock valuation of the blocked transition.

Let $A_\alpha(w, v)$ be a blocking assignment, i.e., an assignment for which $\alpha_{AX}(w, v)$ evaluates to 0. Since $A_\alpha((\alpha_{AX})(w, v)) = 0$ 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 $q_\alpha$ and $q'_\alpha$ such that $A_{q_\alpha}(w) = A_{q'_\alpha}(w)$ and $A_{q'_\alpha}(v) = A_{q_\alpha}(v)$ there is a

4 The remaining sets are used in timed UMC part: $L_0$ contains subformulas over quantified variables only (which are removed anyway) and is used in proofs, while $L_2$ contains formulas encoding time constraints.

5 The cube reduction (identifying subsets of $w$ which suffice to represent a given constraint) cannot efficiently describe a zone corresponding to a constraint involving the difference of two clocks.
transition \( t = q_{a} \xrightarrow{a} q'_{a} \) for some \( a \in \Sigma \), \( q_{a} = (l_{a}, v_{a}) \) and \( q'_{a} = (l'_{a}, v'_{a}) \). The transition \( t \) is called the blocked transition whereas \( a \) - the blocking action (for \( A_{a} \)).

The starting point is a clause directly blocking \( q_{a} \): the location part \( C^{CONTR}(w^{C}) = genBlockingCl(w^{C}, A_{a}) \) blocks \( l_{a} \), and the timed part \( C^{DBM}(w^{t}) = genBlockingCl(w^{t}, A_{a}) \) blocks \( v_{a} \). Now, we generalize both the clauses: the general framework for the optimizations is the procedure \( blocking_{timed}_{clause}() \) which is executed in \( forall() \) after finding a blocking assignment \( A_{a} \). First, the blocking action is identified. Then, the input formula is searched (Alg. \( DFS_{forall\_time\_opt}() \)). The search identifies the sets \( L_{0}, L_{1}, \) and \( L_{2} \) of subformulas over the subvectors \( v^{C}(a), v^{C} \setminus v^{C}(a), \) and \( v^{t} \), respectively, which imply the false value of \( [\varphi] \). Finally, the location subclause 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\((\alpha_{AX}(w,v),v,A_{a})\)

1: Determine a blocking transition \( t = q_{a} \xrightarrow{a} q'_{a} \) and a blocked action \( a \),
2: Search the formula: \( (L_{0}, L_{1}, L_{2}) = DFS_{forall\_time\_opt}([\varphi](w \leftarrow v), v(a), A_{a}) \)
3: Compute the control part \( C^{CONTR}(w^{C}) \) using \( a \) and \( L_{1} \),
4: Compute the timed part \( C^{DBM}(w^{t}) \) using \( a \) and \( L_{2} \),
5: Return \( C(w) = C^{CONTR}(w^{C}) \lor C^{DBM}(w^{t}) \)

**Algorithm 5** DFS\_{forall\_time\_opt}\((\alpha,U,A_{a})\)

1: stack \( s \), set \( L_{0}, L_{1}, L_{2} \)
2: \( s.push(q_{a}) \)
3: while \( s \) not empty do
4: \( v_{a} \leftarrow s.pop() \)
5: if \( (\beta \text{ encodes a constraint } \varphi \in \mathcal{C}_{\Sigma}, \text{i.e., } \beta = l_{\varphi} ) \) then
6: \( L_{2} = L_{2} \cup \{l_{\varphi}\} \)
7: else
8: \( L_{2} = L_{2} \cup \{\neg l_{\varphi}\} \)
9: else if \( \varphi(\beta) \cap U = \emptyset \) then
10: \( L_{1} = L_{1} \cup \{l_{\beta}\} \)
11: else if \( \varphi(\beta) \subseteq U \) then
12: \( L_{0} = L_{0} \cup \{l_{\beta}\} \)
13: else if \( (\beta = \beta_{1} \land \beta_{2} \text{ and } A_{a}(\beta) = 0) \) \( */ \text{ an optimization}^{7}/ \) then
14: for \( (\gamma \in \{\beta_{1}, \beta_{2}\}) \) do
15: if \( A_{a}(\gamma) = 0 \) then
16: \( s.push(v_{\gamma}); \text{ break} \)
17: else
18: for \( (v_{\gamma} \in \text{suc}(v_{\gamma})) \) \( */ \text{ explore all the direct subformulas of } \beta \) \( */ \) do
19: \( s.push(v_{\gamma}) \)
20: return \( (L_{0}, L_{1}, L_{2}) \)

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

---

6 In more detail, the set \( L_{0} \) is used only for the clarity of the proof, because in each blocking clause the enabling condition of the blocked action \( a \) is explicitly encoded over subvectors \( w_{i} \) for \( i \in \Sigma(a) \).
The optimization introduced in [14] for untimed systems can be applied here. The idea consists in restricting the range of quantification over location part to a subvector encoding the blocked action, in order to exclude the variables unchanged by it. The set $L_{pre}$ contains the literals of $L_{pre}(w)$ occurring in $w^C(a)$, i.e., in the vectors $w^C_i$ for \( i \in \Sigma(a) \). Thus $L_{pre}$ encodes the location predecessor of $a$. Let $L_1(v \leftarrow w)$ denote a set of the formulas s.t. it contains $\alpha(v \leftarrow w)$ for each $\alpha \in L_1$. Finally, $\delta^\mathit{CONTR}(w^C)$ is built of the formulas in $L_1 \cup L_{pre}$.

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 zones. 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 necessary operations on zones. Let $v, v' \in \mathbb{R}^+_{\mathbb{R}}$ and $Z, Z' \in Z(n_X)$. Let $v \leq v'$ iff $\exists \delta \in \mathbb{R}_+$ s.t. $v' = v + \delta$. We have (the operations preserve zones):

1. $Z \cap Z' = \{ v \in Z \mid v \in Z' \}$ (intersection of zones),
2. $Z /\prec = \{ v' \in \mathbb{R}^+_{\mathbb{R}} \mid (\exists v \in Z) v \leq v' \}$ (time predecessor),
3. $[X = 0]Z = \{ v \mid v[X = 0] \in Z \}$ (clock reset inverse).

We use the standard form of normalized constraints. The set $X$ is extended with an additional fictitious clock $x_0 \notin X$, which represents the constant $0$. The set $X \cup \{ x_0 \}$ is denoted with $X^*$. Then, each constraint $\alpha'$ over $X^*$ can be generated by the following grammar: $\alpha' = x_i - x_j \sim c \mid x_i - x_j \sim -\infty \mid x_i - x_j \sim -\infty \mid \alpha' \land \alpha'$, where $x_i, x_j \in X^*$, and $\sim \in \{ <, \leq \}$. The standard conversion of the constraints in $C^\mathit{DN}$ to the normalized form is described in [11]. Now, we formally introduce DBMs:

**Definition 7 (Difference Bounds Matrix).** A difference bounds matrix (DBM) in $\mathbb{R}^+_{\mathbb{R}}$ is a $(n_X + 1) \times (n_X + 1)$ matrix of bounds, with rows and columns indexed from $0$ to $n_X$. The DBM $D = (d_{ij})$, where for each $i, j \in \{ 0, \ldots, n_X \}$ $d_{ij} = (d_{ij}, \sim_{ij})$, represents the zone $Z = \{ x_i - x_j \sim_{ij} \}$. 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 $Z = [D]$. It is assumed that an implementation of DBMs is available together with the operations to calculate union, clock resets, time precondition, and the canonical form. The details can be found in [11].

Now, we are ready to define generalized timed subclauses. Recall that $t$ is a blocked transition, $a$ - the blocked action (labeling the blocked transition), and $L_2$ - some subformulas of $\alpha_{AX}(w, v)$. Next, let $L'$ be a set of the constraints encoded by the literals in $L_2$, defined as $L' = \{ \alpha \mid l_{\alpha} \in L_2 \}$. Then, we build the zone $Z'$ which is constructed from the tightest constraints of $L'$. Formally, we introduce the ordering $\leq$ on the constraints as follows: $<$ is strictly less than $\leq$ and for $\alpha = x_i - x_j \sim c$ and $\alpha' = x_i - x_j \sim c'$ we have $\alpha \leq \alpha'$ if either $c < c'$ or $c = c'$ and $\sim < \sim'$. 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 $a$:

$$Z = ( \bigcap_{i \in \Sigma(a)} \mathcal{I}((l_{\alpha})_{ij}) ) \cap ( \bigcap_{i \in \Sigma(a)} [Y = 0](Z' \cap ( \bigcap_{i \in \Sigma(a)} \mathcal{I}((l_{\alpha})_{ij}) ))$$

The timed subclause encodes the constraints of $Z$: $L_Z = \{ l_{\alpha} \mid \alpha \in Z \}$ and $c_b^{DBM}(w^t) = \mathit{gen Blocking Cl}(L_Z, A_\alpha)$. 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 $\mathit{forall()}$, guarantee the clauses to be blocking clauses and this way $\chi(w)$ to properly characterize $[AX\varphi](w)$.

\footnote{A similar optimization exists for the formula $\beta = \beta_1 \setminus \beta_2$ and $A_\alpha(\beta) = 1$}
Definition 8. Consider an optimized algorithm forall() called for the formula $\alpha_{AX}(w, v)$ and the variables of $v$. Define the following two conditions for the clause $c_b$ in a blocking assignment $A_\alpha(w, v)$ with the blocking transition $q_\alpha \rightarrow q'_\alpha$:

C1: for each state $q$ such that $A_q(c_b(w)) = 0$, there is a state $q'$ such that $q \rightarrow q'$ and $A_q'(|\varphi|)(w)) = 0$.

C2: $A_{q_\alpha}(c_b(w)) = 0$.

Then we have:

Theorem 2. For 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 conditions C1 and C2, then $A_\alpha(\chi(w)) = 0$ iff $q \not\in AX\varphi$, for each state $q$ of the model $M(TA)$.

Proof. ($\Rightarrow$) If $A_q(\chi(w)) = 0$, then there is $c_b$ in $\chi$ such that $A_q(c_b(w)) = 0$. Then, by C1 and C2, there is $q'$ such that $q \rightarrow q'$ and $A_q'(|\varphi|)(w)) = 0$. This implies that $q' \not\in \varphi$.

($\Leftarrow$) Assume that $q \not\in AX\varphi$. 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 $A_q(c_b(w)) = 0$ (due to C2). So, we have $A_\alpha(\chi(w)) = 0$ (because $\chi$ is the conjunction of the blocking clauses).

Case 2. No blocking assignment $A_\alpha(w, v)$ 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 $c_b'(w)$ such that $A_q(c_b'(w)) = 0$ as otherwise there would be another blocking assignment found (for example such that it agrees with $A_q(w)$ on $w$). If $A_q(c_b'(w)) = 0$, 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., $c_b = genBlockingCl(w, A_\alpha)$), both the conditions C1 and C2 are satisfied.

Lemma 1. The blocking clause $c_b$ generated by the optimized algorithm forall() satisfies the conditions C1 and C2.

Proof. C1: Consider a state $q = (l, v)$ blocked by $c_b$. First we prove that there is a transition enabled in $q$. Let $A_\alpha(w, v)$ be the blocking assignment for $c_b(w)$, and $t = q_\alpha \rightarrow q'_\alpha$, where $q_\alpha = (l_\alpha, v_\alpha)$ and $q'_\alpha = (l'_\alpha, v'_\alpha)$, be the blocking transition for $A_\alpha(w, v)$. Note that in $l_\alpha$, the locations of the components of $\Sigma(a)$ are the same as in $l_\alpha$ (as they are encoded in $c_b$). Moreover, recall that the zone $Z$ is encoded in $c_b^{DBM}$. Because $c_b(w) = c_b^{DBM}(w) \vee c_b^{CONTR}(w)$ and $c_b(w)$ is false both in $A_\alpha$ and $A_q$, $c_b^{DBM}$ 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 $v' \in Z'$ such that $(l, v) \rightarrow (l', v')$. So we have $q \rightarrow q'$ for $q' = (l', v')$ and some $v' \in Z'$.

Next, we prove that $A_q'(\varphi)(w) = 0$. Recall that the variables in the sets $L_0, L_1$, and $L_2$ assigned as in $A_\alpha$ imply that $A_q'(\varphi)(v) = 0$ (the formula search of $\varphi$ identified these sets so that this property was true). The assignments of the corresponding variables over $v$ and $w$ in $L_0$ are the same in $A_\alpha(v)$ and $A_q'(w)$, and the same is true also for $L_1$ but with possibly different locations encoded over $v$ and $w$ (these encodings imply however the same values of $L_1$ variables). Because $v' \in Z'$ and $v'_\alpha \in Z'$, and because the most strict constraints of $L_2$ were chosen to $Z'$, also the variables of $L_2$ have the same values in $A_\alpha(v)$ and $A_q'(w)$.

C2: As every literal of $c_b(w)$ is false in $A_\alpha$, we have $A_q(c_b(w)) = A_\alpha(c_b(w) = 0$.

Note that the upper invariants are forbidden for efficiency reasons: some components irrelevant for the property and not participating in the blocked action can be abstracted. Without these invariants, it is not necessary to ensure that time can flow in them without forcing any action, so the corresponding locations need not be taken into account.
5 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 [11], 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 \( (n \geq 2) \) and a controlling process. The processes indexed with 1, \ldots, \( n \) compete for an exclusive access to the shared resource (indexed with 0).

![Diagram showing Fischer's mutual exclusion protocol]

**Fig. 1.** Fischer’s mutual exclusion

The experimental results comparing Verics UMC to RED and UPPAAL\(^8\) are shown in Table 1. Notice that UMC clearly performs best when \( \delta > \Delta \). The tested property consists in reachability of a state where two processes are in their critical sections.

<table>
<thead>
<tr>
<th>System</th>
<th>Time [s]</th>
<th>System</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>n (\Delta\delta)</td>
<td>U</td>
<td>R</td>
<td>V</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>2</td>
<td>T</td>
</tr>
</tbody>
</table>

**Table 1.** Testing \( \varphi^1 = \text{EF}(\forall_{1 \leq i, j \leq n, i \neq j}(\text{crit}_i \land \text{crit}_j)) \), U - Uppaal, R - RED, V - Verics; T/F - formula true/false

Notice that the performance of our tool degrades with the increase of the parameter values, which 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.

6 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

\(^8\) Verics: verics.ipipan.waw.pl, RED: www.iis.sinica.edu.tw/~farn/red, UPPAAL: www.uppaal.com
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 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

7 Appendix

not an integral part of the paper, added to explain the presented ideas

7.1 Example

A simple calculation is shown for the system given in Fig. 2, composed of two timed automata \(\mathcal{T}_A_1\) and \(\mathcal{T}_A_2\). The verified property is \(AG(p_1 \lor p_2)\). There are two clocks \(x\) and \(y\), and clock valuations are of the form \(u = (x, y)\).

\[
\begin{array}{c}
\begin{tikzpicture}
\node (A) at (0,0) {\mathcal{T}_A_1};
\node (B) at (2,0) {\mathcal{T}_A_2};
\draw[->] (A) -- node[above]{$p_1$} node[below]{$x < 3$} (B);
\node (C) at (1,1) {$p_2$};
\draw[->] (A) -- node[above]{$p_2$} node[below]{$y > 1$} (B);
\draw[->] (B) -- node[above]{$p_1$} node[below]{$y \geq 2$} (C);
\end{tikzpicture}
\end{array}
\]

Fig. 2. A simple network of automata used in the example [left]; the formula encoding the computed state set after the first iteration, with the sets of subformulas found by the search algorithm

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

1. The blocking assignment \(A_1^a\) is found which determines the states \(q_0 = ((l_1^1, l_2^1), u)\) and \(q_0' = ((l_1^2, l_2^2), u')\), with \(v = (1.5, 4)\) and \(v' = (0, 6.5)\). The algorithm \(DFS_{forall\_time\_opt}()\) finds the following sets: \(L_0 = \{[p_2](v)\}, L_1 = \{[p_1](v)\}, L_2 = \emptyset\). Thus \(Z' = true\) and we compute \(Z = x < 3\). Finally, \(c_0^2 = \neg pre(a_1)(w C) \lor \neg [p_1](w C) \lor -l_{x < 3}(w')\).

2. The second blocking assignment \(A_2^a\) determines the states \(q_0 = ((l_1^1, l_2^1), v)\) and \(q_0' = ((l_1^2, l_2^2), v')\), with \(v = (4, 2)\) and \(v' = (4, 5.2)\). The algorithm \(DFS_{forall\_time\_opt}()\) finds the following sets: \(L_0 = \{- pre(a_1)(v)\}, L_1 = \{[p_1](v)\}, L_2 = \{l_{x < 3}\}\). The formula searched is shown in Fig. 2, notice that the subformula \(\phi\) is skipped as the another argument of the conjunction suffices to imply the false value. Finally, we get \(c_0^2 = \neg pre(a_2)(w C) \lor \neg [p_1](w C) \lor -l_{x < 3}(w')\).

The result of the computation is \([AG(p_1 \lor p_2)](w) = \alpha_Q = [p_1 \lor p_2] \land c_0^1 \land c_0^2\). This formula conjuncted with \(I_{\phi'}(w)\) is not satisfiable, what means that the verified property is not true in the model.

7.2 Explaining the mutex verification

We explain in detail why the presented method is effective for the mutual exclusion system shown in the paper. Let’s consider a pair of the processes (1, 2), but all the pairs are processed in the same way. In the first iteration, \((l_1^1, l_2^1) \xrightarrow{enter_1} (l_1^3, l_2^3)\) 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 \(\mathcal{T}_A_0\). Then, in the third iteration, there are two blocking transitions in the first one \(\mathcal{T}_A_2\) still remains in its critical section while \(\mathcal{T}_A_1\) 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. The above considerations are illustrated in Figure 3.
Fig. 3. The back-image exploration of the state \((l_1^2, l_2^2)\) with \(\delta > \Delta\) (left); the zone \(Z\) corresponding to the location \((l_1^2, l_2^2, l_0)\) (middle); the zone \(Z\) corresponding to the state \((l_1^2, l_2^2, l_0)\) with \(\delta_1 < \Delta\) and \(\delta_2 > \Delta\) (right).