*Fundamenta Informaticae 85 (2008) 1–16*
*IOS Press*

# SAT-based Unbounded Model Checking of Timed Automata[*]

**Wojciech Penczek[†][‡], Maciej Szreter**
*Institute of Computer Science, PAS*
*Ordona 21, 01-237 Warsaw, Poland*
*penczek@ipipan.waw.pl*
*mszreter@ipipan.waw.pl*

**Abstract.** We present an improvement to the SAT-based Unbounded Model Checking (UMC, for short) algorithm [13]. Our idea consists in building blocking clauses of literals corresponding not only to propositional variables encoding states, but also to more general subformulas over these variables encoding sets of states. This way our approach alleviates an exponential blow-up in the number of blocking clauses. A hybrid algorithm for verifying Timed Automata is proposed, where the timed part of blocking clauses is computed using 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

Model checking is becoming an acknowledged method supporting the design of complex systems having many successful applications around. However, the exponential state space 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 authors acknowledge partial support from Ministry of Science and Higher Education (3 T11C 011 28).
[†]Address for correspondence: Institute of Computer Science, PAS, Ordona 21, 01-237 Warsaw, Poland
[‡]Also works: Department of Computer Science, University of Podlasie, Poland

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, known as the SAT-problem, is NP-complete. However, many very efficient algorithms (heuristics) for testing satisfiability have been designed recently. Therefore, numerous verification problems have been translated to the SAT-problem. Bounded Model Checking (BMC, for short) seems to be the state-of-the-art SAT-based model checking method [18]. Some types of flaws can be easily found in very large systems. However, despite of these well-known advantages, BMC has also 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 branching time temporal logic. Given these facts, one can ask whether the SAT-based approach could be used in model checking in a different way. The symbolic verification based on Binary Decision Diagrams (BDD) [12] is an obvious analogue. It turns out that UMC [13] emerged in 2002 as a SAT-based counterpart of BDD. However, the method has not achieved a wide popularity since then and although some extensions were reported [9], it seems that the performance of the algorithm is still inferior to other symbolic approaches based on BDDs. In the conclusions of [13], two major problems are stated:

1. The formulas encoding the whole state space are represented by semi-canonical Directed Acyclic Graphs (DAG). This representation can be much less concise than BDD in the case of equivalent but syntactically different formulas.

2. Blocking clauses are built over a 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 item. In [13], it is stated: *"If a solution can be found to this problem, a dramatic improvement in performance might result"*. As far as the above mentioned algorithm of quantifier elimination is concerned, our modification consists in generating blocking clauses over an extended set of variables, including the variables encoding subformulas over propositions. Blocking clauses are generated by searching an input formula, instead of taking into account the way in which blocking assignments have been found by a solver. Then, the above algorithm is optimized for dealing with formulas arising from the verification of timed automata: the range of quantification is restricted by exploring the structure of the system given by a current assignment, and a method based on Difference Bound Matrices (DBMs, for short) is introduced for dealing effectively with formulas encoding timed constraints.

The rest of the paper is organized as follows. In Chapter 2 the original method of SAT-based quantifier elimination is shown. Chapter 3 presents Timed Automata and their discretized abstract timed models, based on the detailed region graphs. The key ideas of the generalizations are described in Chapter 4, beginning with the algorithm generating generalized blocking clauses, and then its further optimizations exploring the structure of verified systems. Chapter 5 contains experimental results on verification of the standard mutual exclusion protocol. The paper is concluded with a summary and some directions of a possible future work.

## 1.1. Related Work

The ideas similar to ours can be found in [8], where the generalized blocking assignments use circuit cofactoring, but are not applied to 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, 14]. A symbolic verification of timed systems, where the quantifiers are eliminated using a BDD-based algorithm is explored in [19]. The principles of the SAT-solvers can be found in [18].

## 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.

### 2.1. Propositional Logic

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: $\vee$ - disjunction and $\neg$ - negation, with the derived operators $\wedge$ - 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** $l$ is a variable of $PV$, or its negation. A **clause** $c$ is a disjunction of zero or more literals $l_1 \vee \cdots \vee l_n$. 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 $c_1 \wedge \cdots \wedge c_k$.

An **assignment** $A$ is a partial function $A : PV \rightarrow \{1, 0\}$, where $1$ stands for $true$ and $0$ stands for $false$. An assignment is said to be **total** if its domain equals to $PV$. An assignment is extended form $PV$ to $\mathcal{F}$ in the standard way, i.e., assuming the standard interpretation of the boolean operators. A total assignment is said to be **satisfying** for a formula $\alpha$ if the value of $\alpha$ is 1 for the assignment $A$ (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), where the graph $DAG(\alpha)$ represents a formula $\alpha$. Contrary to the BDD-like semantic representation, 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 (so it returs UNSAT).

#### 2.1.1. 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 propositional variables used in $\alpha$ and $PV^C(\alpha) = \{l_\beta \in PV \mid \beta \in Subform(\alpha)\}$ be a set of the literals corresponding to the subformulas of $\alpha$.

Let $toCNF$ be the standard translation of propositional formulas to CNF [16]. Given a 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 $l_\beta \in PV^C(\alpha)$, and for every assignment $A$ such that $A(toCNF(\alpha)) = 1$, we have $A(l_\beta) = A(\beta)$. Consequently, the CNF formula $toCNF(\alpha) \wedge l_\alpha$ is satisfiable iff $\alpha$ is satisfiable. This fact is commonly exploited in algorithms testing satisfiability. Moreover, a formula $\alpha$ is valid when the CNF formula $\beta = toCNF(\alpha) \wedge l_{\neg\alpha}$ is unsatisfiable, which is used in algorithms eliminating universal quantifiers. More details can be found in [16].

### 2.1.2. Elimination of universal quantifiers

Quantified Boolean Formulas (QBF, for short) are a fragment of the First-Order Logic extending propositional logic with quantifiers ranging over propositions. The syntax of QBF is defined in the following way:

$$\alpha := p \mid \neg\alpha \mid \alpha \vee \alpha \mid \exists p.\alpha \mid \forall p.\alpha.$$

The semantics of the quantifiers is as follows:

- $\exists p.\alpha \equiv \alpha(p \leftarrow \mathbf{true}) \vee \alpha(\mathbf{p} \leftarrow \mathbf{false})$, and
- $\forall p.\alpha \equiv \alpha(p \leftarrow \mathbf{true}) \wedge \alpha(\mathbf{p} \leftarrow \mathbf{false})$,

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 $\mathbf{v} = (v[1], \ldots, v[m])$, we use the notation $\overline{\mathbf{v}}$ for the set $\{v[1], \ldots, v[m]\}$, and $\forall \mathbf{v}.\alpha$ to denote $\forall v[1] \ldots \forall v[m].\alpha$. Moreover, 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 [13] from a QBF formula. A pseudo-code of the procedure $forall()$ is shown in Algorithm 1. Notice that $forall(\alpha, U)$ returns a propositional formula in CNF, which is equivalent to $\forall U.\alpha$.

The algorithm exploits the fact each clause of a CNF formula equivalent to the input formula 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., these 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 the quantified variables as soon as a new blocking clause is generated.

**Definition 2.1. (Blocking assignment, blocking clause)**
Consider the procedure $forall()$ (Alg. 1). A satisfying assignment $A_\alpha = SAT(\beta)$ for $\beta$ is called a *blocking assignment*. A *blocking clause* $c_b$ for $A_\alpha$ is a clause over the set of variables $PV(\alpha)$ having the following two properties: (i) $A_\alpha(c_b) = 0$, and (ii) $\alpha \Rightarrow c_b$.

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

```
1: $\chi = \mathbf{true}$, $\beta = toCNF(\alpha) \wedge l_{\neg\alpha}$
2: **while** $(SAT(\beta) \neq UNSAT)$ **do**
3:     compute the blocking clause $c_b$
4:     for each $p \in U$, remove literals $p$ and $\neg p$ from $c_b$
5:     $\chi = \chi \wedge c_b$, $\beta = \beta \wedge c_b$
6: **return** $\chi$
```

**Theorem 2.1. ([13])**
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 $\forall U.\alpha$.

### 2.1.3. Quantifier elimination under a restriction

Some operations in symbolic model checking are considered under a restriction in order to improve on their efficiency. Intuitively, given a propositional formula $\beta$ describing a restriction (denoted by $\downarrow \beta$), the valuations satisfying the resulting formula have to satisfy $\beta$ as well. For example, $forall(\forall U.\alpha) \downarrow \beta$ is evaluated by substituting $toCNF(\alpha)$ with $toCNF(\alpha) \wedge toCNF(\beta) \wedge l_\beta$ in line 1. This way the algorithm considers only assignments that make $\alpha$ false but $\beta$ true. The restriction is used also in fixpoint computations in UMC, which is shown and explained at the end of the next section.

## 3. Timed Automata and Model Checking

In this section we define timed automata [2], their discretizations, and models generated by them. Let's start with some preliminary notions. In what follows, $\mathbb{N}$ ($\mathbb{R}_+$) denotes the set of the natural numbers (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 is equal to zero and the time flows when no action is taken. By $\mathcal{X}$ we denote a finite set $\{x_1, \ldots, x_{n_\mathcal{X}}\}$ of variables, called *clocks*. A *clock constraint* over $\mathcal{X}$ is defined by the following grammar:

$$\psi = \mathbf{true} \mid x_i \sim c \mid x_i - x_j \sim c \mid \psi \wedge \psi,$$

where $x_i, x_j \in \mathcal{X}$, $c \in \mathbb{N}$, and $\sim \in \{\leq, <, =, >, \geq\}$. The constraints of the form $\mathbf{true}$, $x \sim c$ and $x_i - x_j \sim c$ are called *atomic*. Let $\mathcal{C}_\mathcal{X}^\ominus$ denote the set of clock constraints over $\mathcal{X}$, wheres $\mathcal{C}_\mathcal{X}$ be its subset without the inequalities involving clock differences.

A function $v : \mathcal{X} \to \mathbb{R}_+$ assigning to each clock $x$ a positive value $v(x)$ is called a *clock valuation*. By $\mathbb{R}_+^{n_\mathcal{X}}$ we denote the set of all the clock 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 $v \in \mathbb{R}_+^{n_\mathcal{X}}$, the satisfaction relation $\models$ for a clock constraint $\mathfrak{cc} \in \mathcal{C}_\mathcal{X}^\ominus$ is defined inductively as follows:

- $v \models \mathbf{true}$,
- $v \models (x_i \sim c)$ iff $v(x_i) \sim c$,
- $v \models (x_i - x_j \sim c)$ iff $v(x_i) - v(x_j) \sim c$,
- $v \models (\mathfrak{cc} \wedge \mathfrak{cc}')$ iff $v \models \mathfrak{cc}$ and $v \models \mathfrak{cc}'$.

For a constraint $\mathfrak{cc} \in \mathcal{C}_\mathcal{X}^\ominus$, let $[\![\mathfrak{cc}]\!]$ denote the set of all the clock valuations satisfying $\mathfrak{cc}$, i.e., $[\![\mathfrak{cc}]\!] = \{v \in \mathbb{R}_+^{n_\mathcal{X}} \mid v \models \mathfrak{cc}\}$. By a *(time) zone* in $\mathbb{R}_+^{n_\mathcal{X}}$ we mean each convex polyhedron $Z \subseteq \mathbb{R}_+^{n_\mathcal{X}}$ defined by a clock constraint, i.e., $Z = [\![\mathfrak{cc}]\!]$ for some $\mathfrak{cc} \in \mathcal{C}_\mathcal{X}^\ominus$ (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 $Z(n_\mathcal{X})$.

**Definition 3.1. (Timed automaton)**
A *timed automaton* $\mathcal{TA}$ is a tuple $(\Sigma, L, l^\circ, E, \mathcal{X}, \mathcal{I})$, where $\Sigma$ is a finite set of actions, $L$ is a finite set of locations, $l^\circ \in L$ is the initial location, $E \subseteq L \times \Sigma \times \mathcal{C}_\mathcal{X} \times 2^\mathcal{X} \times L$ is a transition relation, $\mathcal{X}$ is a finite set of clocks, and $\mathcal{I} : L \longrightarrow \mathcal{C}_\mathcal{X}$ is a state invariant function. Each element $e \in E$ is denoted by

$e = l \xrightarrow{a,\mathfrak{cc},Y} l'$, 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 $\mathfrak{cc} \in \mathcal{C}_{\mathcal{X}}$ is the *guard* condition for $e$.

In order to reason about a system represented by a timed automaton $\mathcal{TA}$, we define a valuation function $V_{\mathcal{TA}} : L \to 2^{PV}$, assigning a subset of propositions of $PV$ to each its location.

## 3.1. Semantics of Timed Automata

Let $\mathcal{TA} = (\Sigma, L, l^{\circ}, E, \mathcal{X}, \mathcal{I})$ be a timed automaton. A *concrete state* of $\mathcal{TA}$ is a pair $(l, v)$, where $l \in L$ and $v \in \mathbb{R}^{n_{\mathcal{X}}}$ is a clock valuation. The *concrete state space* of $\mathcal{TA}$ is the structure $C(\mathcal{TA}) = (Q_c, q^{\circ}, \longrightarrow_c)$, where $Q_c = L \times \mathbb{R}^{n_{\mathcal{X}}}$ is the set of all the concrete states, $q^{\circ} = (l^{\circ}, v^{\circ})$ with $v^{\circ}(x) = 0$ for all $x \in \mathcal{X}$ is the initial state, and $\longrightarrow_c \subseteq Q_c \times (\Sigma \cup \mathbb{R}) \times Q_c$ is the transition relation, defined by the union of the action- and time-successors as follows:

- for $\delta \in \mathbb{R}$, $(l, v) \xrightarrow{\delta}_c (l, v + \delta)$ iff $v, v + \delta \in [\![\mathcal{I}(l)]\!]$ (*time successor*),
- for $a \in \Sigma$, $(l, v) \xrightarrow{a}_c (l', v')$ iff $(\exists \mathfrak{cc} \in \mathcal{C}_{\mathcal{X}})(\exists Y \subseteq \mathcal{X})$ such that $l \xrightarrow{a,\mathfrak{cc},Y} l' \in E$, $v \in [\![\mathfrak{cc}]\!]$, $v' = v[Y = 0]$ and $v' \in [\![\mathcal{I}(l')]\!]$ (*action successor*).

For $(l, v) \in Q$ and $\delta \in \mathbb{R}_+$, let $(l, v) + \delta$ denote $(l, v + \delta)$. A $q_0$*-run* $\rho$ of $\mathcal{TA}$ is a maximal sequence of concrete states $\rho = q_0 \xrightarrow{\delta_0}_c q_0 + \delta_0 \xrightarrow{a_0}_c q_1 \xrightarrow{\delta_1}_c q_1 + \delta_1 \xrightarrow{a_1}_c q_2 \xrightarrow{\delta_2}_c \ldots$, 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* iff $\Sigma_{i \in \mathbb{N}} \delta_i$ is unbounded. A timed automaton is *progressive* iff all its runs beginning in the initial state are progressive. We model a concurrent system by a *network of timed automata*[1] $\mathfrak{TA}$, i.e., a set of timed automata (called *components*), where $\mathfrak{TA} = \{\mathcal{TA}_i \mid 1 \leq i \leq n\}$ with $\mathcal{TA}_i = (\Sigma_i, L_i, l_i^{\circ}, E_i, \mathcal{X}_i, \mathcal{I}_i)$. For $a \in \Sigma$, let $\Sigma(a) = \{1 \leq i \leq n \mid a \in \Sigma_i\}$ be the set of the indices of all the components including $a$.

**Definition 3.2. (Product of timed automata)**
The *product* of a network $\mathfrak{TA}$ is the timed automaton $\mathcal{TA} = (\Sigma, L, l^{\circ}, E, \mathcal{X}, \mathcal{I})$, where $\Sigma = \bigcup_{1 \leq i \leq n} \Sigma_i$, $L = \prod_{i \in \{1,\ldots,n\}} L_i$, $l^{\circ} = (l_1^{\circ}, \ldots, l_n^{\circ})$, $\mathcal{X} = \bigcup_{i \in \{1,\ldots,n\}} \mathcal{X}_i$, $\mathcal{I}((l_1, \ldots, l_n)) = \bigwedge_{i \in \{1,\ldots,n\}} \mathcal{I}_i(l_i)$, and the transition relation is given by:
$((l_1, \ldots, l_n), a, \bigwedge_{i \in \Sigma(a)} \mathfrak{cc}_i, \bigcup_{i \in \Sigma(a)} Y_i, (l'_1, \ldots, l'_n)) \in E$ iff $(\forall i \in \Sigma(a))(l_i, a, \mathfrak{cc}_i, Y_i, l'_i) \in E_i$ and $(\forall i \in \{1, \ldots, n\} \setminus \Sigma(a))\ l'_i = l_i$.

For technical reasons to be explained later, we consider only timed automata without upper invariants, i.e., these 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, let $c_{max}$ denote the largest constant occurring in the clock constraints of the automaton.

## 3.2. Abstract Discretized Models

In this section we define an equivalence on clock valuations [2, 16] and a discretization in order to define finite-state abstract discretized models over which temporal properties can be model checked by means of UMC.

[1]We deal with progressive timed automata only.

In what follows, let $frac(\delta)$ denote the fractional part of $\delta$, whereas $\lfloor \delta \rfloor$ - its integral part, for $\delta \in \mathbb{R}_+$.

**Definition 3.3. (Equivalence of clock valuations)**
For two clock valuations $v, v' \in \mathbb{R}_+^{n_\mathcal{X}}$, $v \simeq v'$ 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) $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$,
   b) $frac(v(x)) = 0$ iff $frac(v'(x)) = 0$, and
   c) $frac(v(x)) \leq frac(v(x'))$ iff $frac(v'(x)) \leq frac(v'(x'))$.

Let $\mathcal{TA} = (\Sigma, L, l^\circ, E, \mathcal{X}, \mathcal{I})$ be a timed automaton with $n_\mathcal{X}$ clocks and $V_\mathcal{TA}$ be a valuation function. Next, let $\mathcal{M}^c(\mathcal{TA}) = (C(\mathcal{TA}), V^c_\mathcal{TA})$ be the concrete model for $\mathcal{TA}$, where $V^c_\mathcal{TA}(l, v) = V_\mathcal{TA}(l)$. Similarly to BMC [16], we choose the discretization step $\Delta = 1/d$, where $d$ is a fixed even number[2] greater than $2n_\mathcal{X}$. The *discretized clock space* is defined as $\mathbb{U}^{n_\mathcal{X}}$, where $\mathbb{U} = \{2k\Delta \mid 0 \leq k\Delta \leq c_{max} + 1\}$ 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 (the action successor combined with the time successor) of [2]. The choice of the abstract model to be discretized is motivated by reducing the number of transitions while still preserving the $CTL$ properties of the detailed region graph model.

**Definition 3.4.** The *(abstract) discretized model* of a timed automaton $\mathcal{TA}$ is a finite structure $\mathcal{M}(\mathcal{TA}) = ((Q, q^\circ, \longrightarrow), V^c_\mathcal{TA})$, where $Q = L \times \mathbb{U}^{n_\mathcal{X}}$, $q^\circ = (l^\circ, v^\circ)$ and $\longrightarrow \subseteq Q \times \Sigma \times Q$ is defined as follows:

- $(l, w) \xrightarrow{a} (l, v)$ iff $(l, w') \xrightarrow{\delta}_c ; \xrightarrow{a}_c (l, v')$ for some $\delta \in \mathbb{R}_+$ and some $w', v' \in \mathbb{U}^{n_\mathcal{X}}$ such that $w \simeq w'$ and $v \simeq v'$ (the time successor combined[3] with the action successor transition relation).

By $q \longrightarrow q'$ we mean that $q \xrightarrow{a} q'$ for some $a \in \Sigma$.

For specifying properties we use the logic $CTL$ [5] having the syntax as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathrm{AX}\varphi \mid \mathrm{AG}\varphi \mid \mathrm{A}(\varphi \mathrm{U} \varphi).$$

$CTL$ is interpreted in the standard way over $\mathcal{M}(\mathcal{TA})$ [16].

## 3.3. Encoding of the Transition Relation

For the encoding we require that every component $\mathcal{TA}_i = (\Sigma_i, L_i, l_i^\circ, E_i, \mathcal{X}_i, \mathcal{I}_i)$ of a network $\mathfrak{TA}$ satisfies the following two conditions for every action $a \in \Sigma_i$:

1. each pair of locations is connected with at most one transition labelled with an $a$,
2. all the local transitions of $\mathcal{TA}_i$, labelled with $a$, reset the same clocks.

[2] A good choice for $d$ is the minimal such a number, which equals to $2^{d'}$ for some $d'$.
[3] The symbol ; denotes composition of relations.

Notice that each automaton can be translated to the above form by adding fresh transition labels.

Now we give some details of this encoding, based on [22]. Since the set of states $Q$ of our model is finite, every state $q = (l, u) \in Q$ of $\mathcal{M}(\mathcal{TA})$ can be represented by a bit vector $\mathbf{s}_q = (\mathbf{s}_q[1], \ldots, \mathbf{s}_q[n_b])$ of length $n_b$ depending on the number of locations of $L$, 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 $\mathbf{w} = (\mathbf{w}[1], \ldots, \mathbf{w}[n_b])$, where $\mathbf{w}[i]$, for $i = 1, \ldots, n_b$, is a propositional variable (called *state variable*). Here, the bit vector $\mathbf{s}_q = (\mathbf{s}_q^C, \mathbf{s}_q^t)$, is composed of two[4] subvectors representing $l$ and $u$ respectively, and is encoded by the vector $\mathbf{w} = (\mathbf{w}^C, \mathbf{w}^t)$. Let $lit : \{0, 1\} \times PV \to \mathcal{F}$ be a function defined as follows: $lit(0, p) = \neg p$ and $lit(1, p) = p$. This function is used for encoding states. Sometimes we will treat vectors as sets of propositional variables.

Concerning locations, if $|L_i|$ is the number of locations of $\mathcal{TA}_i$, then $m_i = \lceil log_2(|L_i|) \rceil$ state variables suffice to encode every location. The subvector $\mathbf{w}_i^C$ of $\mathbf{w}^C$ encodes the locations of $\mathcal{TA}_i$. The vector $\mathbf{w}^C = (\mathbf{w}_1^C, \ldots, \mathbf{w}_m^C)$ is of length $m = \sum_{i=1}^{n} m_i$. Define $\mathbf{w}^C(a)$ to be the subvector of $\mathbf{w}^C$ composed of $\mathbf{w}_i^C$ for $i \in \Sigma(a)$. As far as clocks are concerned, 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 $\mathbf{w}^t$ consists of $n_{\mathcal{X}}$ subvectors $\mathbf{w}_i^I$ and $n_{\mathcal{X}}$ subvectors $\mathbf{w}_i^F$ having $r^I = \lceil \log_2(2c_{max} + 2) \rceil$ and $r^F = \lceil \log_2(2n_{\mathcal{X}}) \rceil$ bits each, and representing $I_x$ and $F_x$, respectively[5]. Thus, each clock is encoded by $r_{\mathcal{X}} = r^I + r^F$ state variables, and the size of $\mathbf{w}^t$ is $r = n_{\mathcal{X}} \cdot r_{\mathcal{X}}$. A discretized clock valuation $(v_1, \ldots, v_{n_{\mathcal{X}}})$ is encoded by $\mathbf{w}^t = (\mathbf{w}_{I_1}, \mathbf{w}_{F_1}, \ldots, \mathbf{w}_{I_{n_{\mathcal{X}}}}, \mathbf{w}_{F_{n_{\mathcal{X}}}})$.

Next, we introduce the propositional formulas $I_q(\mathbf{w})$ and $T(\mathbf{w}, \mathbf{v})$ encoding a discretized state $q$ and the transition relation of $\mathcal{M}(\mathcal{TA})$ (see [22] for the details). We have $I_q(\mathbf{w}) = \bigwedge_{i=1}^{n_b} lit(\mathbf{s}_q[i], \mathbf{w}[i])$. By $A_q$ we denote the assignment encoding $q$ over $\mathbf{w}$, that is $A_q(\mathbf{w}[i]) = \mathbf{s}_q[i]$ for all $1 \leq i \leq n_b$. The formula $T(\mathbf{w}, \mathbf{v})$ is such that for each two states $q, q' \in (L \times \mathbb{U}^{n_{\mathcal{X}}})$ and for every assignment $A$ encoding them over $\mathbf{w}$ and $\mathbf{v}$ (i.e., $A_q(\mathbf{w}) = A(\mathbf{w})$ and $A_{q'}(\mathbf{v}) = A(\mathbf{v})$) we have $q \longrightarrow q'$ iff $A(T(\mathbf{w}, \mathbf{v})) = 1$. In order to implement $T(\mathbf{w}, \mathbf{v})$ the clock constraints are encoded. For $\mathfrak{cc} \in \mathcal{C}_{\mathcal{X}}^{\ominus}$, we use $l_{\mathfrak{cc}}$ to denote the encoding of $\mathfrak{cc}$ over the vector $\mathbf{w}^t$.

## 3.4. Characterizing Temporal Formulas

We use the standard fixpoint characterization [7] of $CTL$. Given a $CTL$ formula $\varphi$, the corresponding propositional formula $[\varphi](\mathbf{w})$ is computed s.t. it encodes the states of the system that satisfy $\varphi$.

**Definition 3.5. (Translation)**
The translation $[\,\cdot\,]$ is inductively defined as follows:

- $[p](\mathbf{w})$ is a formula such that we have $q \models p$ iff $A_q([p](\mathbf{w})) = 1$, for every $q \in Q$,
- $[\neg\varphi](\mathbf{w}) = \neg[\varphi](\mathbf{w})$,
- $[\varphi \vee \psi](\mathbf{w}) = [\varphi](\mathbf{w}) \vee [\psi](\mathbf{w})$,
- $[AX\varphi](\mathbf{w}) = forall(T(\mathbf{w}, \mathbf{v}) \Rightarrow [\varphi](\mathbf{w} \leftarrow \mathbf{v}), \mathbf{v})$,

[4]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 $i = 1, \ldots, n$.

[5]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.

- $[AG\varphi](\mathbf{w}) = fssm_{AG}([\varphi](\mathbf{w}))$,
- $[\mathrm{A}\varphi\mathrm{U}\psi](\mathbf{w}) = lfp_{\mathrm{AU}}([\varphi](\mathbf{w}), [\psi](\mathbf{w}))$.

The UMC method is based on the fact that the formula $\varphi$ holds in the initial state $q^\circ$ of $\mathcal{M}$ iff the propositional formula $[\varphi](\mathbf{w}) \wedge I_{q^\circ}(\mathbf{w})$ is satisfiable.

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

```
$\alpha_Z(\mathbf{w}) = \alpha_Q(\mathbf{w}) = [\varphi](\mathbf{w})$
while ($\alpha_Z \neq \mathbf{true}$) do
    $\alpha_Z(\mathbf{w}) = forall(\neg T(\mathbf{w},\mathbf{v}) \vee [\varphi](\mathbf{w} \leftarrow \mathbf{v}), \mathbf{v}) \downarrow \alpha_Q(\mathbf{w})$
    $\alpha_Q(\mathbf{w}) = \alpha_Q(\mathbf{w}) \wedge \alpha_Z(\mathbf{w})$
return $\alpha_Q(\mathbf{w})$
```

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

```
$\alpha_Q(\mathbf{w}) = \mathbf{false}$, $\alpha_Z(\mathbf{w}) = [\psi](\mathbf{w})$
while $SAT(\neg(\alpha_Z \implies \alpha_Q)) \neq UNSAT$ do
    $\alpha_Q(\mathbf{w}) = \alpha_Q(\mathbf{w}) \vee \alpha_Z(\mathbf{w})$
    $\alpha_Z(\mathbf{w}) = forall(\neg T(\mathbf{w},\mathbf{v}) \vee [\varphi](\mathbf{w} \leftarrow \mathbf{v}), \mathbf{v}) \wedge [\varphi](\mathbf{w})$
return $\alpha_Q(\mathbf{w})$
```

Notice that thanks to using the restriction in $fssm_{AG}([\varphi](\mathbf{w}))$ in each iteration it suffices to consider only the transitions from the states that have not been computed in the previous iterations.

## 4. Generalized Blocking Clauses in UMC

We have implemented the original algorithm $forall(\alpha, \mathrm{U})$ [13], where the blocking clauses are built over propositions of $\alpha$. Our experiments have confirmed its limited efficiency. 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 case of these resulting from UMC it produces clauses of maximal length.

The main idea of our paper, based on [20], consists in constructing blocking clauses not only over the propositions of $PV(\alpha)$, but also over propositions of $PV^C(\alpha)$ corresponding to the subformulas of $\alpha$. Consider the standard algorithm $forall(\alpha, U)$. We first discuss its modification in the general case and then its application to the timed UMC. To this aim, two steps 3 and 4, of Algorithm 1 are modified in the following way. The procedure $DFS_{forall-time-opt}(\alpha, U, A_\alpha)$ (Alg. 4) performs the DFS through $DAG(\alpha)$. It begins with the root $v_\alpha$ and returns a set of literals $L_1 \subseteq PV^C(\alpha)$[6], which imply the false value of $\alpha$ when assigned by $A_\alpha$. The resulting clause $c_b$ (shown to be a blocking clause in [20]) 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 $c_b$ is. In case of $U = PV(\alpha)$, no optimization is achieved. Formally, we have $c_b = genBlockingCl(L_1, A_\alpha)$, where

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

with $l'_\beta = \neg l_\beta$ if $A_\alpha(\beta) = 1$ and $l'_\beta = l_\beta$ if $A(\beta) = 0$.

Now, let's examine an application of $forall(\alpha, U)$ to computing $[\mathrm{AX}\varphi](\mathbf{w})$. We use $\alpha_{\mathrm{AX}}(\mathbf{w}, \mathbf{v})$ for denoting $T(\mathbf{w}, \mathbf{v}) \Rightarrow [\varphi](\mathbf{w} \leftarrow \mathbf{v})$ and $\chi(\mathbf{w})$ for the formula returned by $forall(\alpha_{\mathrm{AX}}(\mathbf{w}, \mathbf{v}), \mathbf{v})$.

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

Unfortunately, the optimization of generalized blocking clauses is likely not to make any improvement here. The reason is that all the state variables of $\mathbf{v}$ are quantified, so each blocking clause would describe only one state of the model. Clearly, this reduces to describing the discretized states one by one, which in case of the full region graph is not feasible in practice[7]. Therefore, we have to 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 $\mathbf{v}$ to a subset of the state variables of $\mathbf{v}^C$, encoding the locations of the components that do not participate 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 $A_\alpha(\mathbf{w},\mathbf{v})$ be a *blocking assignment*, i.e., an assignment for which $\alpha_{\mathrm{AX}}(\mathbf{w},\mathbf{v})$ evaluates to $0$. Since $A_\alpha((\alpha_{\mathrm{AX}})(\mathbf{w},\mathbf{v})) = 0$ and $\alpha_{\mathrm{AX}}(\mathbf{w},\mathbf{v})$ is an implication, the formula $T(\mathbf{w},\mathbf{v})$ is true in $A_\alpha$ and it determines the transition in the model. Recall that $A_\alpha(T(\mathbf{w},\mathbf{v})) = 1$ implies that for the states $q_\alpha = (l_\alpha, v_\alpha)$ and $q'_\alpha = (l'_\alpha, v'_\alpha)$ such that $A_{q_\alpha}(\mathbf{w}) = A_\alpha(\mathbf{w})$ and $A_{q'_\alpha}(\mathbf{v}) = A_\alpha(\mathbf{v})$ there is a transition $t = q_\alpha \xrightarrow{a} q'_\alpha$ for some $a \in \Sigma$. The transition $t$ is called the *blocked transition* whereas $a$ - the *blocked action* for $A_\alpha$.

Consider the clause directly blocking $q_\alpha$:

- its location part $c_b^{CONTR}(\mathbf{w}^C) = genBlockingCl(\overline{\mathbf{w}^C}, A_\alpha)$ blocks $l_\alpha$, whereas
- its timed part $c_b^{DBM}(\mathbf{w}^t) = genBlockingCl(\overline{\mathbf{w}^t}, A_\alpha)$ blocks $v_\alpha$.

Now, both the clauses above are generalized. The general framework of the optimizations is shown in the procedure $blocking\_timed\_clause()$ (Algorithm 4), which is replacing line 3 and 4 in $forall(\alpha, U)$. We call the resulting algorithm $forall_{opt}$. First, the blocked transition and the blocked action is identified. Then, the input formula is searched by $DFS_{forall\_time\_opt}()$. The search identifies the sets $L_0$, $L_1$, and $L_2$ of subformulas over the subvectors $\mathbf{v}^C(a)$, $\mathbf{v}^C \setminus \mathbf{v}^C(a)$, and $\mathbf{v}^t$, respectively, which imply the false value of $[\varphi](\mathbf{v})$. Finally, the location subclause is calculated on the basis of the set $L_1$[8], and the timed subclause is computed using the set $L_2$. More details of the construction are given below.

**Algorithm 4** $blocking\_timed\_clause(\alpha_{\mathrm{AX}}(\mathbf{w},\mathbf{v}), \mathbf{v}, A_\alpha)$

```
1: Determine the blocked transition $t = q_\alpha \xrightarrow{a} q'_\alpha$ and the blocked action $a$.
2: Search the formula: $(L_0, L_1, L_2) = DFS_{forall\_time\_opt}([\varphi](\mathbf{w} \leftarrow \mathbf{v}), \mathbf{v}(a), A_\alpha)$.
3: Compute the control part $c_b^{CONTR}(\mathbf{w}^C)$ by means of $a$ and $L_1$.
4: Compute the timed part $c_b^{DBM}(\mathbf{w}^t)$ by means of $a$ and $L_2$.
5: Return $c_b(\mathbf{w}) = c_b^{CONTR}(\mathbf{w}^C) \vee c_b^{DBM}(\mathbf{w}^t)$.
```

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 [20] for untimed systems can as well be applied here. The idea consists

[7]The cube reduction (identifying subsets of $\mathbf{w}^t$ which suffice to represent a given constraint) cannot efficiently describe a zone corresponding to a constraint involving the difference of two clocks.

[8]Note that 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 $\mathbf{w}_i$ for $i \in \Sigma(a)$.

[9]A similar optimization exists for the formula $\beta = \beta_1 \vee \beta_2$ and $A_\alpha(\beta) = 1$.

**Algorithm 5** $DFS_{forall\_time\_opt}(\alpha, U, A_\alpha)$

```
stack s, set $L_0$, $L_1$, $L_2$
s.push( $q_\alpha$ )
while s not empty do
  $v_\beta$ =s.pop();
  if ( $\beta$ encodes a constraint $\mathfrak{cc} \in \mathcal{C}^{\ominus}_{\mathcal{X}}$, i.e., $\beta = l_{\mathfrak{cc}}$ ) then
    if ( $A_\alpha(\beta) = 0$ ) then
      $L_2 = L_2 \cup \{l_{\mathfrak{cc}}\}$
    else
      $L_2 = L_2 \cup \{\neg l_{\mathfrak{cc}}\}$
  else if $PV(\beta) \cap U = \emptyset$ then
    $L_1 = L_1 \cup \{l_\beta\}$
  else if $PV(\beta) \subseteq U$ then
    $L_0 = L_0 \cup \{l_\beta\}$
  else if ($\beta = \beta_1 \wedge \beta_2$ and $A_\alpha(\beta) = 0$) /* an optimization[9] */ then
    for ($\gamma \in \{\beta_1, \beta_2\}$) do
      if ( $A_\alpha(\gamma) = 0$) then
        s.push($v_\gamma$); break
  else
    for ($v_\gamma \in succ(v_\beta)$) /* explore all the direct subformulas of $\beta$ */ do
      s.push($v_\gamma$)
return $(L_0, L_1, L_2)$
```

in restricting the range of quantification over the location part to the subvector encoding the blocked action, in order to exclude the variables unchanged by it. Let $L_{pre}$ be a set containing the literals of $I_{q_\alpha}(\mathbf{w})$ occurring in $\mathbf{w}^C(a)$, i.e., in the vectors $\mathbf{w}^C_i$ for $i \in \Sigma(a)$. Thus $L_{pre}$ encodes the location predecessor of $a$. Let $L_1(\mathbf{v} \leftarrow \mathbf{w})$ denote the set containing $\alpha(\mathbf{v} \leftarrow \mathbf{w})$ for each $\alpha \in L_1$. Finally, $c_b^{CONTR}(\mathbf{w}^C) = genBlockingCl(L_1(\mathbf{v} \leftarrow \mathbf{w}) \cup L_{pre}, A_\alpha)$, i.e., the location part of the blocking clause is built of the negated literals corresponding to the source locations of the blocked transition, and to the formulas (found by the search algorithm) encoding sets of locations of the components in which this transition does not occur.

Now, we show how to compute timed subclauses by operating on time constraints. Our approach is based on 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 on DBMs. Finally, the resulting constraints are encoded in the propositional logic and added directly to the blocking clause.

First, we define some useful operations on zones. Let $v, v' \in \mathbb{R}_+^{n_\mathcal{X}}$ and $Z, Z' \in Z(n_\mathcal{X})$. Let $v \leq v'$ iff $\exists \delta \in \mathbb{R}_+$ s.t. $v' = v + \delta$. The operations listed below preserve zones:

(1) $Z \cap Z' = \{v \in Z \mid v \in Z'\}$ (intersection of zones),

(2) $Z \swarrow = \{v' \in \mathbb{R}_+^{n_\mathcal{X}} \mid (\exists v \in Z)\ v' \leq v\}$ (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 $\mathcal{X}$ is extended with an additional fictitious clock $x_0 \notin \mathcal{X}$, which represents the constant 0. The set $\mathcal{X} \cup \{x_0\}$ is denoted with $\mathcal{X}^+$. Then, each constraint $\mathfrak{cc}'$ over $\mathcal{X}^+$ can be generated by the following grammar:

$$\mathfrak{cc}' = x_i - x_j \sim c \mid x_i - x_j \sim \infty \mid x_i - x_j \sim -\infty \mid \mathfrak{cc}' \wedge \mathfrak{cc}',$$

where $x_i, x_j \in \mathcal{X}^+$ and $\sim \in \{<, \leq\}$. The standard conversion of the constraints in $\mathcal{C}_{\mathcal{X}}^{\ominus}$ to the normalized form is described in [16]. Now, we formally introduce DBMs:

**Definition 4.1. (Difference Bounds Matrix)**
A difference bounds matrix (DBM) in $\mathbb{R}^{n_{\mathcal{X}}}$ is a $(n_{\mathcal{X}} + 1) \times (n_{\mathcal{X}} + 1)$ matrix of bounds, with rows and columns indexed from 0 to $n_{\mathcal{X}}$. The DBM $D = (\mathbf{d}_{ij})$, where for each $i, j \in \{0, \ldots, n_{\mathcal{X}}\}$ $\mathbf{d}_{ij} = (d_{ij}, \sim_{ij})$, represents the zone $Z = [\![\bigwedge_{i=0}^{n_{\mathcal{X}}} \bigwedge_{j=0}^{n_{\mathcal{X}}} (x_i - x_j \sim_{ij} d_{ij})]\!]$. 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 $Z = [\![D]\!]$. It is assumed that an implementation of DBMs is available together with the operations to calculate union, clock resets, time predecessor, and the canonical form. The details can be found in [16].

Now, we are ready to define generalized timed subclauses. Recall that $t$ is the blocked transition, $a$ - the blocked action (labelling the blocked transition), $\mathfrak{cc}_g$ - the guard of $t$, and $L_2$ - some subformulas of $\alpha_{AX}(\mathbf{w}, \mathbf{v})$ over $\mathbf{v}^t$. Next, let $L'$ be a set of the constraints encoded by the literals in $L_2$, defined as $L' = \{\mathfrak{cc} \mid l_{\mathfrak{cc}} \in L_2\}$. 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 $\mathfrak{cc} = x_i - x_j \sim c$ and $\mathfrak{cc}' = x_i - x_j \sim' c'$ we have $\mathfrak{cc} \preceq \mathfrak{cc}'$ if either $c < c'$ or $c = c'$ and $\sim \leq \sim'$. 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 transition $t$:

$$Z = (\bigcap_{i \in \Sigma(a)} [\![\mathcal{I}_i((l_\alpha)_i)]\!]) \cap \big([\![\mathfrak{cc}_g]\!] \cap [Y = 0](Z' \cap (\bigcap_{i \in \Sigma(a)} [\![\mathcal{I}_i((l'_\alpha)_i)]\!])\big) \swarrow$$

Then, $c_b^{DBM}(\mathbf{w}^t) = genBlockingCl(L_Z, A_\alpha)$, where $L_Z = \{l_{\mathfrak{cc}} \mid \mathfrak{cc} \in Z'\}$.

In order to show correctness of our optimizations we formulate two conditions $C1$ and $C2$ (below). If they are satisfied by the clauses of $\chi(\mathbf{w})$ returned by $forall_{opt}()$, then they guarantee the clauses to be blocking clauses and this way $\chi(\mathbf{w})$ to properly characterize $[AX\varphi](\mathbf{w})$.

**Definition 4.2.** Consider the algorithm $forall_{opt}()$ called for the formula $\alpha_{AX}(\mathbf{w}, \mathbf{v})$ and the variables of $\mathbf{v}$. For the clause $c_b$ in the blocking assignment $A_\alpha(\mathbf{w}, \mathbf{v})$ of the blocked transition $q_\alpha \longrightarrow q'_\alpha$ define the following two conditions:

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

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

Then, the following theorem holds:

**Theorem 4.1.** Let $\chi(\mathbf{w})$ be the formula computed by $forall_{opt}(\alpha_{\mathrm{AX}}(\mathbf{w}, \mathbf{v}), \mathbf{v})$. If each clause $c_b$ of $\chi(\mathbf{w})$ satisfies the conditions $C1$ and $C2$, then $A_q(\chi(\mathbf{w})) = 0$ iff $q \not\models \mathrm{AX}\varphi$, for each state $q$ of the model $\mathcal{M}(\mathcal{TA})$.

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

($\Leftarrow$) Assume that $q \not\models \mathrm{AX}\varphi$. Since the algorithm has terminated, there are two cases to consider.
Case 1. A blocking assignment $A_\alpha(\mathbf{w}, \mathbf{v})$ was found such that it agrees with $A_q(\mathbf{w})$ on $\mathbf{w}$. Then, $\chi(\mathbf{w})$ contains a clause $c_b(\mathbf{w})$ such that $A_q(c_b(\mathbf{w})) = 0$ (due to $C2$). So, we have $A_q(\chi(\mathbf{w})) = 0$ (because $\chi(\mathbf{w})$ is the conjunction of blocking clauses).
Case 2. No blocking assignment $A_\alpha(\mathbf{w}, \mathbf{v})$ was found such that it agrees with $A_q(\mathbf{w})$ on $\mathbf{w}$. Then, since the algorithm has terminated (which follows from $C2$), a blocking clause $c'_b(\mathbf{w})$ must have been generated s.t. $A_q(c'_b(\mathbf{w})) = 0$ as otherwise there would be another blocking assignment found (for example such that it agrees with $A_q(\mathbf{w})$ on $\mathbf{w}$). If $A_q(c'_b(\mathbf{w})) = 0$, then, clearly, we have $A_q(\chi(\mathbf{w})) = 0$. ∎

Notice that for the simplest blocking clause composed of all the state variables of $\mathbf{w}$ (i.e., $c_b = genBlockingCl(\overline{\mathbf{w}}, A_\alpha)$), both the conditions $C1$ and $C2$ are satisfied.

**Lemma 4.1.** The blocking clause $c_b(\mathbf{w})$ generated by the algorithm $forall_{opt}()$ satisfies the condition $C1$ and $C2$.

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

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

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

Note that the upper invariants are forbidden for efficiency reasons: some components unrelevant 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.

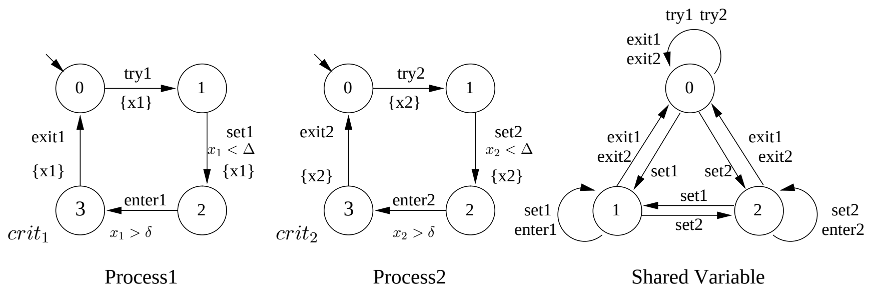

Figure 1. Fischer's mutual exclusion

| System | | | | Time [s] | | | System | | | | Time [s] | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n | $\Delta$ | $\delta$ | T/F | U | R | V | n | $\Delta$ | $\delta$ | T/F | U | R | V |
| 12 | 1 | 2 | T | 580 | 304 | 59 | 10 | 3 | 4 | T | 33 | 53 | 50 |
| 13 | 1 | 2 | T | - | 657 | 88 | 11 | 3 | 4 | T | 125 | 133 | 71 |
| 15 | 1 | 2 | T | - | - | 154 | 10 | 2 | 1 | F | 7 | 49 | 97 |
| 18 | 1 | 2 | T | - | - | 376 | | | | | | | |
| 20 | 1 | 2 | T | - | - | 491 | | | | | | | |

Table 1. Testing $\varphi = \mathrm{EF}\big(\bigvee_{1\leq i,j\leq n, i\neq j} crit_i \wedge crit_j\big)$, U - Uppaal 4.0.6, R - RED 5.0, V - Verics; T/F - formula true/false

## 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 [10, 16] of `Verics` [6, 15], and the SAT solver ZChaff. In order to evaluate the performance, we have examined the well-known Fischer's Mutual Exclusion protocol. The example models a system consisting of $n$ independent processes $(n \geq 2)$ and the controlling process. The processes indexed with $1, \ldots, n$ compete for an exclusive access to the shared resource (indexed with $0$).

The experimental results comparing UMC of `Verics` [11] to RED [21] and UppAal[10] [17] 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.

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`.

[10]Verics: `verics.ipipan.waw.pl`, RED: `www.iis.sinica.edu.tw/~farn/red`, Uppaal: `www.uppaal.com`; The default set of options is used in Uppaal, excluding the symmetry reductions.

## 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 canonical. It should be possible to use optionally BDD graphs for a formula representation, which would significantly improve the performance. Contrary to the current representation, adding blocking clauses would reduce the size of a formula. Another important aim is to relax the restriction of the upper invariants. This is quite straightforward 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

[1] P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT-solvers. In *Proc. of the 6th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 411–425. Springer-Verlag, 2000.

[2] R. Alur and D. Dill. Automata-theoretic verification of real-time systems. In *Formal Methods for Real-Time Computing, Trends in Software Series*, pages 55–82. John Wiley & Sons, 1996.

[3] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using Clock Difference Diagrams. In *Proc. of the 11th Int. Conf. on Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 341–353. Springer-Verlag, 1999.

[4] D. Beyer. Rabbit: Verification of real-time systems. In *Proc. of the Workshop on Real-Time Tools (RT-TOOLS'01)*, pages 13–21, 2001.

[5] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[6] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS: A tool for verifying timed automata and Estelle specifications. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 278–283. Springer-Verlag, 2003.

[7] E. A. Emerson and E. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. of the 7th Int. Colloquium on Automata, Languages and Programming (ICALP'80)*, volume 85 of *LNCS*, pages 169–181. Springer-Verlag, 1980.

[8] M. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In *Proc. of the Int. Conf. on Computer-Aided Design (ICCAD'04)*, pages 510–517, 2004.

[9] M. Kacprzak, A. Lomuscio, and W. Penczek. From bounded to unbounded model checking for temporal epistemic logic. *Fundamenta Informaticae*, 63(2-3):221–240, 2004.

[10] Magdalena Kacprzak, Alessio Lomuscio, Artur Niewiadomski, Wojciech Penczek, Franco Raimondi, and Maciej Szreter. Comparing bdd and sat based techniques for model checking chaum's dining cryptographers protocol. *Fundam. Inform.*, 72(1-3):215–234, 2006.

[11] W. Nabiałek M. Kacprzak, A. Niewiadomski, W. Penczek, M. Szreter A. Półrola, and B. Woźna. VerICS 2006: A model checker for real time and multi-agent systems. In *Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'07)*, pages 345–356. Warsaw University, 2007.

[12] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[13] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proc. of the 14th Int. Conf. on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 250–264. Springer-Verlag, 2002.

[14] J. Møller, J. Lichtenberg, H. Andersen, and H. Hulgaard. Difference Decision Diagrams. In *Proc. of the 13th Int. Workshop Computer Science Logic (CSL'99)*, volume 1683 of *LNCS*, pages 111–125. Springer-Verlag, 1999.

[15] W. Nabiałek, A. Niewiadomski, W. Penczek, A. Półrola, and M. Szreter. VerICS 2004: A model checker for real time and multi-agent systems. In *Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'04)*, volume 170(1) of *Informatik-Berichte*, pages 88–99. Humboldt University, 2004.

[16] W. Penczek and A. Półrola. *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*, volume 20 of *Studies in Computational Intelligence*. Springer-Verlag, 2006.

[17] P. Pettersson and K. G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000.

[18] Mukul R. Prasad, Armin Biere, and Aarti Gupta. A survey of recent advances in sat-based formal verification. *STTT*, 7(2):156–173, 2005.

[19] S. Seshia and R. Bryant. Unbounded, fully symbolic model checking of timed automata using boolean methods. In *Proc. of the 15th Int. Conf. on Computer Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 154–166. Springer-Verlag, 2003.

[20] Maciej Szreter. *SAT-based model checking of distributed systems*. PhD thesis, Instytut Podstaw Informatyki PAN, 2006.

[21] F. Wang. RED: Model checker for timed automata with clock-restriction diagram. In *Proc. of the Int. Workshop on Real-Time Tools (RT-TOOLS'01)*, 2001.

[22] B. Woźna. *Ograniczona weryfikacja modelowa dla logik czasu rozgałęzionego: Szybka metoda falsyfikacji*. PhD thesis, IPI PAN, June 2003. In Polish.