Towards Bounded Model Checking* for the Universal Fragment of TCTL

Wojciech Penczek[1], Bożena Woźna[2], and Andrzej Zbrzezny[2]

[1] Institute of Computer Science, PAS
Ordona 21, 01-237 Warsaw, Poland
and
Podlasie Academy
Institute of Informatics, Siedlce, Poland
penczek@ipipan.waw.pl
[2] Institute of Mathematics and Computer Science, PU
Armii Krajowej 13/15, 42-200 Częstochowa, Poland
{b.wozna,a.zbrzezny}@wsp.czest.pl

Abstract. Bounded Model Checking (BMC) based on SAT methods consists in searching for a counterexample of a particular length and to generate a propositional formula that is satisfiable iff such a counterexample exists. Our paper shows how the concept of bounded model checking can be extended to deal with TACTL (the universal fragment of TCTL) properties of Timed Automata.

1 Introduction

Model checking [13] consists in verifying that a finite state concurrent program P satisfies a property φ (denoted $P \models \varphi$). When P is represented by its model M_P and the property φ is given by a temporal logic formula, one checks that $M_P \models \varphi$. Generating M_P for P and checking that $M_P \models \varphi$ is automated. The complexity of model checking methods strongly depends on the translation from P to M_P and on the temporal logic to which the formula φ belongs to.

Recently, the interest in automated verification is moving towards concurrent real-time systems. The properties to be verified are usually expressed in either a standard temporal logic like LTL [13] and CTL [18,19], or in its timed version like MITL [5] and TCTL [1]. The practical applicability of model checking is strongly restricted by the state explosion problem. Therefore, many different reduction techniques have been introduced in order to alleviate the state explosion. The major methods include application of partial order reductions [7,14,26,29,30,31,37,38], symmetry reductions [20], abstraction techniques [15,16], BDD-based symbolic storage methods [11,27], and SAT-related algorithms [8,12,22,33,35].

Bounded model checking (BMC) based on SAT methods has been recently introduced as a complementary technique to BDD-based symbolic model checking

* Partly supported by the State Committee for Scientific Research under the grant No. 8T11C 01419.

W. Damm and E.-R. Olderog (Eds.): FTRTFT 2002, LNCS 2469, pp. 265–288, 2002.

for LTL [8,9]. An extension of the BMC method to verification of the properties expressed in ACTL has been shown for the first time in [34]. The main modification of the original algorithm for LTL consists in the translation of a model M to several symbolic paths (instead of one), which can start at arbitrary states of the model. Moreover, the translation for an ACTL formula is simpler than for LTL as there are no nested path modalities. The basic idea of BMC is to search for a counterexample of a particular length and to generate a propositional formula that is satisfiable iff such a counterexample exists. The efficiency of this method is based on an observation that if a system is faulty, then only a fragment of its state space is sufficient for finding an error [8,9,34].

In the standard approach to verification of real time systems, Timed Automata are used as model generators, whereas the properties are expressed as formulas of TCTL [1]. There are two alternative methods of building an abstract model for a Timed Automaton. The first one is based on the notion of a clock region graph [1], which is built by BFS-algorithm. The second method exploits the partitioning algorithm [2,3,36], which iteratively builds the minimal model w.r.t. the initial covering of the concrete state space.

The aim of our paper is to generalize the method of bounded model checking to verification of Timed Automata. Since the application of BMC to the whole language of CTL (so TCTL) seems to be very difficult, if not impossible, as one might need to search all the state space of the model, we consider the universal fragment of TCTL, called TACTL. Using a discretization method of [6], we translate the validity of a TACTL formula φ in the clock region model of a Timed Automaton $\mathcal{A}$ ($M_{\mathcal{A}} \models \varphi$) to a propositional formula. This formula is then tested for satisfiability using ZChaff [28,40] to verify whether $M_{\mathcal{A}} \models \varphi$. Since the length of the propositional formula is polynomial in the size of $\mathcal{A}$ and φ, the state explosion problem inherent in model checking is taken care by the efficient satisfiability solver.

The rest of the paper is organized as follows. The next section contains the discussion of the related work. Then, in section 3 Timed Automata are introduced. Logics TCTL and TACTL are defined in section 4. Section 5 defines region graphs. Bounded model checking for ACTL and TACTL is presented in the next two sections. Generalizations and further optimizations are discussed in Section 8. Section 9 contains experimental results, whereas final remarks are given in Section 10.

2 Related Work

Our paper is an extended and improved version of [33], where a general approach to applying BMC for a subset of TACTL was described.

The idea of BMC for a temporal logic is taken from the papers by Clarke et al. [8,9]. Timed Automata have been defined and investigated in many papers [1,4,36]. We adopt the definition given in [36]. Model checking for TCTL was considered by several authors using different approaches: over clock region models [1], on-the-fly [10], space-efficient [25], and over minimal models [36].

Our approach is closely related to [4] and [36], from which we draw the idea of the translation of model checking problem for TACTL to the model checking problem for *fair*-ACTL (ACTL^{F}). Motivation for considering only the universal fragment of CTL can be found in [21,32]. Similar arguments apply to TCTL.

3 Timed Automata

We start with introducing Timed Automata and their runs. Hereafter, $\mathbb{N} = \{0, 1, 2, \ldots\}$ denotes the set of natural numbers and $\mathbb{R}_+$ denotes the set of non-negative real numbers. Let $\mathrm{X} = \{x_1, \ldots, x_n\}$ be a finite set of variables, called *clocks*. A *clock valuation* is a function $v : \mathrm{X} \to \mathbb{R}_+$, assigning to each clock x a non-negative value $v(x)$. The set of all the valuations is denoted by $\mathbb{R}_+^n$. For a subset Y of X by $v[Y := 0]$ we mean the valuation v' such that $\forall x \in Y,\ v'(x) = 0$ and $\forall x \in \mathrm{X} \setminus Y,\ v'(x) = v(x)$. For $\delta \in \mathbb{R}_+$, $v + \delta$ denotes the valuation v'' such that $\forall x \in \mathrm{X}, v''(x) = v(x) + \delta$. The set Ψ_{X} of *clock constraints* over the set of clocks X is defined inductively as follows:

$$\psi := x_i \sim c \mid x_i - x_j \sim c \mid \psi \wedge \psi$$

where $x_i, x_j \in \mathrm{X}$, $i, j \in \{1, \ldots, n\}$, $\sim \in \{\leq, <, =, >, \geq\}$, and $c \in \mathbb{N}$.

A clock valuation v *satisfies* the clock constraint $\psi \in \Psi_{\mathrm{X}}$, if

$$\begin{array}{ll} v \models x_i \sim c & \text{iff } v(x_i) \sim c \\ v \models x_i - x_j \sim c & \text{iff } v(x_i) - v(x_j) \sim c \\ v \models \psi \wedge \psi' & \text{iff } v \models \psi \text{ and } v \models \psi' \end{array}$$

For each $\psi \in \Psi_{\mathrm{X}}$ by $[\![\psi]\!]$ we denote the set of all the clock valuations satisfying ψ, i.e., $[\![\psi]\!] = \{v \in \mathbb{R}_+^n \mid v \models \psi\}$. Similarly, $\mathbb{P}(\Psi_{\mathrm{X}}) = \{[\![\psi]\!] \subseteq \mathbb{R}_+^n \mid \psi \in \Psi_{\mathrm{X}}\}$. Now, we are ready to define Timed Automata.

Definition 1. *A* timed automaton $\mathcal{A}$ *is a 6-tuple* $(\Sigma, S, s^0, E, \mathrm{X}, \mathrm{I})$*, where* Σ *is a finite set of actions,* S *is a finite set of locations,* $s^0 \in S$ *is an initial location,* $E \subseteq S \times \Sigma \times \Psi_{\mathrm{X}} \times 2^{\mathrm{X}} \times S$ *is a transition relation,* X *is a finite set of clocks,* $\mathrm{I} : S \longrightarrow \Psi_{\mathrm{X}}$ *is a state invariant.*

Each element e *of* E *is denoted by* $e := s \xrightarrow{l,\psi,Y} s'$*. This represents a transition from the location* s *to the location* s' *on the input action* l*. The set* $Y \subseteq \mathrm{X}$ *gives the clocks to be reset with this transition, whereas* $\psi \in \Psi_{\mathrm{X}}$ *is the enabling condition for* e*.*

Given a transition $e := s \xrightarrow{l,\psi,Y} s'$, we write $source(e)$, $target(e)$, $guard(e)$ for s, s' and ψ, respectively. The clocks of a Timed Automaton allow to express the timing properties. An enabling condition constrains the execution of a transition without forcing it to be taken. An invariant condition allows an automaton to stay at a some state only as long as the constraint is satisfied.

A (concrete) state of $\mathcal{A}$ is a pair $q = (s, v)$, where $s \in S$ and $v \in \mathbb{R}_+^n$. The set of all the concrete states is denoted by Q, i.e., $Q = S \times \mathbb{R}_+^n$. The initial state of $\mathcal{A}$ is defined as $q^0 = (s^0, v^0)$ with $v^0(x_i) = 0$ for all $x_i \in \mathrm{X}$.

Let $l \in \Sigma$ and $\delta \in \mathbb{R}_+$. A timed consecution relation in $\mathcal{A}$ is defined by action- and time-successors as follows:

- $(s, v) \xrightarrow{l} (s', v')$ iff there is a transition $s \xrightarrow{l,\psi,Y} s' \in E$ such that $v \models \psi$ and $v' = v[Y := 0]$ and $v' \models \mathbb{I}(s')$,
- $(s, v) \xrightarrow{\delta} (s', v')$ iff $s = s'$ and $v' = v + \delta$ and $v' \models \mathbb{I}(s')$; ($(s', v')$ is denoted by $(s, v) + \delta$).

Let $q \in Q$. A *q-run* ρ of $\mathcal{A}$ is a maximal sequence of concrete states: $q_0 \xrightarrow{\delta_0} q_0 + \delta_0 \xrightarrow{l_0} q_1 \xrightarrow{\delta_1} q_1 + \delta_1 \xrightarrow{l_1} q_2 \xrightarrow{\delta_2} \cdots$, where $q_0 = q$, $l_i \in \Sigma$ and $\delta_i \in \mathbb{R}_+$ for each $i \geq 0$.

A state q of $\mathcal{A}$ is a *deadlock* if there is no delay $\delta \in \mathbb{R}_+$ and an action $l \in \Sigma$ such that $q \xrightarrow{\delta} q' \xrightarrow{l} q'$, for some $q', q'' \in Q$. A run ρ has a deadlock iff there is a deadlock state in ρ. A run ρ is said to be *progressive* iff $\Sigma_{i \in \mathbb{N}} \delta_i$ is unbounded. A run ρ is said to be *maximal* iff ρ is infinite and progressive or ρ has a deadlock. A Timed Automaton is *progressive* iff all its infinite runs are progressive. For simplicity of presentation, we consider only progressive Timed Automata. Progressiveness can be checked using the sufficient conditions of [36]. In Section 8 we discuss unrestricted Timed Automata.

4 Logics TCTL and TACTL

This section defines the logic TCTL [1] and two its sublogics: TACTL and TECTL interpreted over concrete models of Timed Automata. As usual let $\mathcal{PV} = \{p_1, p_2, \ldots\}$ be a set of propositional variables.

Definition 2. *A* (concrete) model *for a Timed Automaton $\mathcal{A}$ is a pair $M_{\mathcal{A}} = ((Q, f, q^0), \mathcal{V})$, where $Q = S \times \mathbb{R}_+^n$ is a set of the concrete states of $\mathcal{A}$, f is a function returning a set of all the maximal q-runs for each concrete state $q \in Q$, q^0 is the initial state, $\mathcal{V} : S \longrightarrow 2^{\mathcal{PV}}$ is a valuation function.*

The formulas of TCTL are defined inductively as follows:

$$\alpha, \beta := p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \wedge \beta \mid \mathrm{E}(\alpha \mathrm{U}_I \beta) \mid \mathrm{EG}_I \alpha,$$

where $p \in \mathcal{PV}$ and I is an interval in $\mathbb{R}_+$ with integer bounds of the form $[n, m]$, $[n, m)$, $(n, m]$, (n, m), (n, ∞), and $[n, \infty)$, for $n, m \in \mathbb{N}$.

Intuitively, E means there exists a run, $\alpha \mathrm{U}_I \beta$ is true in a run if β is true for some state within interval I and always earlier α holds.

The logic TECTL is the restriction of TCTL such that the negation can be applied only to propositions, i.e., $\neg\alpha$ is replaced by $\neg p$ above. The set of all the TECTL formulas is denoted by $\mathcal{TF}$. TACTL is the language, which can be defined by replacing each occurrence of the path quantifier E by A in the set $\mathcal{TF}$.

Let $\rho = q_0 \xrightarrow{\delta_0} q_0 + \delta_0 \xrightarrow{l_0} q_1 \xrightarrow{\delta_1} q_1 + \delta_1 \xrightarrow{l_1} q_2 \xrightarrow{\delta_2} \cdots$ be a q_0-run, $q_0 = (s_0, v_0)$, and $|\rho|$ be the length of ρ, i.e., the number of δ_i's if finite and ∞, otherwise.

$q_0 \models p$ iff $p \in \mathcal{V}(s_0)$, $q_0 \models \neg\alpha$ iff $q_0 \not\models \alpha$,
$q_0 \models \alpha \vee \beta$ iff $q_0 \models \alpha$ or $q_0 \models \beta$, $q_0 \models \alpha \wedge \beta$ iff $q_0 \models \alpha$ and $q_0 \models \beta$,
$q_0 \models \mathrm{E}(\alpha \mathrm{U}_I \beta)$ iff $\exists\, \rho \in f(q_0) \exists_{i<|\rho|+1}$ $[\Sigma_{j\leq i}\delta_j \in I$ and $(q_i + \delta_i) \models \beta$ and
$(\forall j \leq i)(\forall \delta \leq \delta_j)\ q_j + \delta \models \alpha \vee \beta]$,
$q_0 \models \mathrm{E}G_I\alpha$ iff $\exists\, \rho \in f(q_0)\ (\forall_{i<|\rho|+1})\ (\forall \delta \leq \delta_i)$with $\Sigma_{j\leq i}\delta_j \in I,\ (q_i + \delta) \models \alpha$.

Note that without a loss of generality we can define the length of a run ρ taking into account the number of time delays (δ_i) only. The action step (l_i) can be seen as a time step with $\delta_i = 0$, for $i \in \mathbb{N}$.

For model $M_{\mathcal{A}}$ and a TCTL formula φ, we say that $M_{\mathcal{A}} \models \varphi$ iff $M_{\mathcal{A}}, q^0 \models \varphi$. The *model checking* problem for TCTL is defined as follows: given a TCTL formula φ and a Timed Automaton $\mathcal{A}$ together and a valuation function $\mathcal{V}$, determine whether $M_{\mathcal{A}} \models \varphi$.

Since $M_{\mathcal{A}}$ is usually infinite, we need to define its finite abstraction, which preserves TCTL or TACTL. Such an abstraction is a region graph, which is defined in the next section.

5 Region Graphs

Given a Timed Automaton $\mathcal{A}$. Let $\Psi \subseteq \Psi_{\mathrm{X}}$ be a non-empty set containing all the clock constrains occurring in any enabling condition used in the transition relation E or in a state invariant of $\mathcal{A}$. Moreover, let c_{max} be the largest constant appearing in Ψ. For $\sigma \in \mathbb{R}_+$, $frac(\sigma)$ denotes the fractional part of σ, and $\lfloor\sigma\rfloor$ denotes its integral part.

Definition 3 (Equivalence of clock valuations). *For two clock valuations v and v' in $\mathbb{R}^n_+$, $v \simeq_\Psi v'$ iff for all $x, y \in \mathrm{X}$ the following conditions are met:*

1. *$v(x) > c_{max}$ implies $v'(x) > c_{max}$*
2. *if $v(x) \leq c_{max}$ then*
   a.) $\lfloor v(x)\rfloor = \lfloor v'(x)\rfloor$, and
   b.) $frac(v(x)) = 0$ implies $frac(v'(x)) = 0$,
3. *for all clock constraints of the form $x - y \sim c$ with $c \in \mathbb{N}$ and $c \leq c_{max}$, $v \models x - y \sim c$ implies $v' \models x - y \sim c$.*

We use $[v]$ to denote the equivalence class of the relation $\simeq_\Psi$ to which v belongs. Such a class is called a *zone*. The set of all the zones is denoted by $Z(n)$. A zone $[v']$ is *final* iff $v(x) > c_{max}$ for all $v \in [v']$ and $x \in \mathrm{X}$. A zone $[v']$ is *open* if there is a clock $x \in \mathrm{X}$ such that $v(x) > c_{max}$ for all $v \in [v']$.
A zone $[v]$ *satisfies* the clock constraint $\psi \in \Psi_{\mathrm{X}}$, if $[v] \models \psi$ iff $\forall v' \in [v],\ v' \models \psi$.
A *region* is a pair $(s, [v])$, where $s \in S$ and $[v] \in Z(n)$. Note that the set of all the regions is finite.

Definition 4 (Time successor). *Let $[v]$ and $[w]$ be two distinct zones. The zone $[w]$ is said to be the* time successor *of $[v]$, denoted $\tau([v])$, iff for each $v' \in [v]$ there exists $\delta \in \mathbb{R}_+$ such that $v' + \delta \in [w]$ and $v' + \delta' \in [v] \cup [w]$ for all $\delta' \leq \delta$.*

The time successor exists for every zone except for the *final* one.

Definition 5 (Action successor). *The zone* $[w]$ *is said to be the* action successor *of* $[v]$ *by a transition* $e : s \xrightarrow{l,\psi,Y} s' \in E$*, denoted* $e([v])$*, iff* $[v] \models \psi$ *and* $[w] = [v[Y := 0]]$.

Definition 6. *The* region graph *of a Timed Automaton* $\mathcal{A}$ *is a finite structure* $\mathcal{RG}(\mathcal{A}) = (\mathcal{W}, \rightarrow, w^0)$*, where* $\mathcal{W} = S \times Z(n)$*,* $w^0 = (s^0, [v^0])$ *and* $\rightarrow \subseteq \mathcal{W} \times (E \cup \{\tau\}) \times \mathcal{W}$ *is defined as follows:*

1. $(s, [v]) \xrightarrow{e} (s', [v'])$ *iff* $[v'] = e([v])$*,* $s = source(e)$*,* $s' = target(e)$*, and* $[v'] \models \mathbb{I}(s')$*, for* $e \in E$*,*
2. $(s, [v]) \xrightarrow{\tau} (s, [v'])$ *iff* $[v'] \models \mathbb{I}(s)$ *and* ($[v'] = \tau([v])$ *or* $[v'] = [v]$ *if* $[v]$ *is final).*

The model based on the region graph of $\mathcal{A}$ *and a valuation function* $\mathcal{V}$ *is defined as* $(\mathcal{RG}(\mathcal{A}), \mathcal{V}')$*, where* $\mathcal{V}' : \mathcal{W} \longrightarrow 2^{\mathcal{PV}}$ *such that* $\mathcal{V}'((s, [v])) = \mathcal{V}(s)$.

6 Bounded Model Checking for TACTL

In this section we describe a method of BMC for TACTL, and show its correctness. The main idea of our method consists in translating the TACTL model checking problem to the ACTL model checking problem [1,36] and applying the bounded model checking for ACTL [34].

In general, the model checking problem for TACTL can be translated to the model checking problem for ACTL^{F} [1]. However, since we have assumed that we deal with progressive Timed Automata only, this translation can be made to the ACTL model checking [36]. The idea is as follows. Given a Timed Automaton $\mathcal{A}$, a valuation function $\mathcal{V}$, and a TACTL formula ψ. First we extend $\mathcal{A}$ with a set of new clocks, to obtain a new automaton $\mathcal{A}'$. Then we construct the region graph for $\mathcal{A}'$ and ψ, and augment the valuation function $\mathcal{V}$. Finally we transform ψ to the ACTL formula ψ'.

More precisely, let X be the set of clocks of $\mathcal{A}$, and $\{I_1, \ldots, I_m\}$ be a set of the non-trivial intervals appearing in ψ. The automaton $\mathcal{A}'$ is like $\mathcal{A}$ except for the set of clocks X', which is defined as follows. $\text{X}' = \text{X} \cup \{x_{n+1}, \ldots, x_{n+m}\}$, i.e., $\text{X}' = \{x_1, \ldots, x_{n+m}\}$. Let $\mathcal{RG}(\mathcal{A}', \psi) = (\mathcal{W}, \rightarrow, w^0)$ be a region graph for $\mathcal{A}'$ and ψ. The set of propositional variables $\mathcal{PV}'$ and the valuation function $\mathcal{V}'$ augments $\mathcal{PV}$ and $\mathcal{V}$ as follows. By $p_{x_{n+i}=0}$ and $p_{x_{n+i} \in I_i}$ we denote the new propositions for every interval I_i appearing in ψ and by $\mathcal{PV}_X$ the set of these propositions. The proposition $p_{x_{n+i}=0}$ is true at a state $(s, [v])$ of $\mathcal{RG}(\mathcal{A}', \psi)$ if $(\exists v' \in [v]) v'(x_{n+i}) = 0$. The proposition $p_{x_{n+i} \in I_i}$ is true at a state $(s, [v])$ of $\mathcal{RG}(\mathcal{A}', \psi)$ if $(\exists v' \in [v])\ v'(x_{n+i}) \in I_i$. Let $\mathcal{V}_X$ be the function labeling each state of the graph $\mathcal{RG}(\mathcal{A}', \psi)$ with the set of propositions from $\mathcal{PV}_X$ true at that state. Let $\mathcal{PV}' = \mathcal{PV} \cup \mathcal{PV}_{\text{X}}$. We define the valuation function $\mathcal{V}' : \mathcal{W} \to 2^{\mathcal{PV}'}$ as follows: $\mathcal{V}'((s, [v])) = \mathcal{V}(s) \cup \mathcal{V}_{\text{X}}((s, [v]))$.

Since the bounded model checking method consists in searching for counterexamples, in practice we check the validity of the negation of a tested formula. Therefore, here we give a translation of a TECTL formula φ to an ECTL

formula (instead of the TACTL formula $\psi = \neg\varphi$). Then, we show the semantics of the formula φ defined over a model based on the region graph $\mathcal{RG}(\mathcal{A}', \varphi)$.

A TECTL formula φ is translated inductively to the ECTL formula φ' as follows:

- $p \in \mathcal{PV}$ is translated to p,
- $\alpha \vee \beta$ is translated to $\alpha' \vee \beta'$,
- $\alpha \wedge \beta$ is translated to $\alpha' \wedge \beta'$,
- $\mathrm{EG}_{I_r}\alpha$ is translated to $\mathrm{EG}(\alpha' \vee \neg p_{x_{n+r} \in I_r})$,
- $\mathrm{E}(\alpha \mathrm{U}_{I_r} \beta)$ is translated to $\mathrm{E}((\alpha' \vee (\beta' \wedge p_{x_{n+r} \in I_r}))\mathrm{U}(\beta' \wedge p_{x_{n+r} \in I_r}))$.

Definition 7 (Semantics). *Let $M = ((\mathcal{W}, \to, w^0), \mathcal{V}')$ be a model based on the region graph $\mathcal{RG}(\mathcal{A}', \varphi)$ and φ be* TECTL *formulas. A* path *in M is a maximal sequence $\pi = (w_0, w_1, \cdots)$ of states such that $w_i \to w_{i+1}$ for each $i < |\pi|$, where $|\pi|$ is the length of π, i.e., the number of its states when finite and ∞ when infinite. For a path $\pi = (w_0, w_1, \cdots)$, let $\pi(i) = w_i$, for each $i \in |\pi|$. $M, (s, [v]) \models \varphi$ denotes that φ is true at the state $(s, [v])$ of M. M is omitted if it is implicitly understood. Let $[v_r]$ denote $[v[x_{n+r} := 0]]$, for $r \in \{1, \ldots, m\}$. The relation $\models$ is defined inductively as follows:*

$(s, [v]) \models p$ *iff* $p \in \mathcal{V}'((s, [v]))$, $(s, [v]) \models \alpha \vee \beta$ *iff* $(s, [v]) \models \alpha$ *or* $(s, [v]) \models \beta$,
$(s, [v]) \models \neg p$ *iff* $p \notin \mathcal{V}'((s, [v]))$, $(s, [v]) \models \alpha \wedge \beta$ *iff* $(s, [v]) \models \alpha$ *and* $(s, [v]) \models \beta$,
$(s, [v]) \models \mathrm{EG}_{I_r}\alpha$ *iff* $\exists \pi \big[\pi(0) = (s, [v_r])$ *and* $\forall_{j<|\pi|} \pi(j) \models (\alpha \vee \neg p_{x_{n+r} \in I_r})\big]$,
$(s, [v]) \models \mathrm{E}(\alpha \mathrm{U}_{I_r} \beta)$ *iff* $\begin{cases} \exists \pi \big[\pi(0) = (s, [v_r])$ *and* $\exists_{j<|\pi|} \big(\pi(j) \models (\beta \wedge p_{x_{n+r} \in I_r})$ \\ *and* $\forall_{0 \le i < j} \pi(i) \models (\alpha \vee (\beta \wedge p_{x_{n+r} \in I_r}))\big)\big]$.

Definition 8 (Validity). *A* TECTL *formula φ is valid in $M = ((\mathcal{W}, \to, w^0), \mathcal{V}')$ (denoted $M \models \varphi$) iff $M, w^0 \models \varphi$.*

It is easy to show that the validity over the model based on the region graph is equivalent to the validity over the concrete model, i.e., $M \models \varphi$ iff $M_{\mathcal{A}} \models \varphi$, for each $\varphi \in$ TECTL (see [36]).

In order to deal with the bounded model checking method, we have to define the bounded semantics, which allows us to interpret the formulas over a fragment of the considered model only. For that purpose we need the definitions of a $k-$path, a loop, and a $k-$model.

Definition 9 ($k-$path). *Let $M = ((\mathcal{W}, \to, w^0), \mathcal{V}')$ be a model based on the region graph $\mathcal{RG}(\mathcal{A}', \varphi)$ and $k \in \mathbb{N}$. A* k-path *is a finite sequence $\pi = (w_0, \cdots, w_k)$ of states of $\mathcal{W}$ such that $w_i \to w_{i+1}$ for each $0 \le i < k$.*

Though a $k-$path is finite, it still can represent an infinite path if there is a *back loop* from the last state of the $k-$path to any of the previous states.

Definition 10 (loop). *We call a $k-$path π a* loop *if $\pi(k) \to \pi(l)$ for some $0 \le l \le k$.*

Definition 11 ($k-$model). *Let $M = ((\mathcal{W}, \to, w^0), \mathcal{V}')$ be a model and $k \in \mathbb{N}$. The $k-$model for M is a structure $M_k = ((\mathcal{W}, Path_k, w^0), \mathcal{V}')$, where $Path_k$ is the set of all the $k-$paths of M.*

For a subset $Path' \subseteq Path_k$ we define

$$States(Path') = \{w \in \mathcal{W} \mid (\exists \pi \in Path')(\exists i \leq k)\ \pi(i) = w\}.$$

Definition 12. *Let $M_k = ((\mathcal{W}, Path_k, w^0), \mathcal{V}')$ be a k-model of M. A structure $M'_k = ((\mathcal{W}', Path'_k, w^0), \mathcal{V}'|_{\mathcal{W}'})$ is a* submodel *of M_k if $Path'_k \subseteq Path_k$, $W' = States(Path'_k)$.*

Similarly to the ACTL case, described in [35], we define the bounded semantics for TECTL formulas φ over k-models M_k of M (submodels M'_k of M_k).

Definition 13 (Bounded semantics). *Let M_k be a $k-$model and α, β be* TECTL *formulas. $M_k, (s,[v]) \models \alpha$ denotes that α is true at the state $(s,[v])$ of M_k. M_k is omitted if it is implicitly understood. Let $[v_r]$ denote $[v[x_{n+r} := 0]]$, $\alpha_r = (\alpha \vee \neg p_{x_{n+r} \in I_r})$, and $\beta_r = (\beta \wedge p_{x_{n+r} \in I_r})$, for $r \in \{1, \ldots, m\}$. The relation $\models$ is defined inductively as follows:*
$(s,[v]) \models p$ *iff* $p \in \mathcal{V}'((s,[v]))$, $(s,[v]) \models \alpha \vee \beta$ *iff* $(s,[v]) \models \alpha$ *or* $(s,[v]) \models \beta$,
$(s,[v]) \models \neg p$ *iff* $p \notin \mathcal{V}'((s,[v]))$, $(s,[v]) \models \alpha \wedge \beta$ *iff* $(s,[v]) \models \alpha$ *and* $(s,[v]) \models \beta$,

$$(s,[v]) \models \mathrm{EG}_{I_r}\alpha \quad \text{iff} \begin{cases} \exists \pi \in Path_k \big[\pi(0) = (s,[v_r]) \text{ and } \forall_{0 \leq j \leq k} \pi(j) \models \alpha_r\big], \\ if\ \pi \text{ is a loop or } \pi(k) \text{ is a deadlock}, \\ false, \quad otherwise, \end{cases}$$

$$(s,[v]) \models \mathrm{E}(\alpha \mathrm{U}_{I_r} \beta) \text{ iff} \begin{cases} \exists \pi \in Path_k \big[\pi(0) = (s,[v_r]) \text{ and } \exists_{0 \leq j \leq k} \big(\pi(j) \models \beta_r \text{ and} \\ \forall_{0 \leq i < j} \pi(i) \models (\alpha \vee \beta_r)\big)\big]. \end{cases}$$

Definition 14 (Validity for Bounded Semantics). *A* TECTL *formula φ is valid in a k-model M_k (denoted $M \models_k \varphi$) iff $M_k, w^0 \models \varphi$.*

Hereafter, let $|M|$ denote the number of states of the model M.
The main theorem of this section states that we can always find a bound k of M such that $M \models_k \varphi$ is equivalent to $M \models \varphi$.

Theorem 1. *Let M be a model and φ be a* TECTL *formula. $M \models \varphi$ iff there is $k \in \{0, \ldots, |M|\}$ such that $M \models_k \varphi$.*

Proof (Sketch). Follows from the following facts:

- The model checking problem of TECTL can be translated to the model checking problem for ECTL$^{\mathrm{F}}$ [1,36].
- Since we have assumed that we deal with progressive Timed Automata only, the above translation can be made to the ECTL model checking.
- By Theorem 1 of [35] we can always find a bound $k \leq |M|$ such that the bounded and unbounded validity for ECTL are equivalent.

Now, we define a function f_k, which determines the number of the k-paths of a submodel M'_k, which is sufficient for checking a TECTL formula.

Definition 15. *Define a function $f_k : \mathcal{TF} \to \mathbb{N}$ as follows:*

- $f_k(p) = f_k(\neg p) = 0$, *where* $p \in \mathcal{PV}'$

- $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$
- $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$
- $f_k(\mathrm{EG}_I\alpha) = (k+1) \cdot f_k(\alpha) + 1$
- $f_k(\mathrm{E}(\alpha \mathrm{U}_I \beta)) = k \cdot f_k(\alpha) + f_k(\beta) + 1$

Now, we can present a BMC method for TACTL.
Let $M = ((\mathcal{W}, \rightarrow, w^0), \mathcal{V}')$ be the model based on $\mathcal{RG}(\mathcal{A}', \psi)$ for a TACTL formula ψ. To answer the question whether $M \not\models \psi$, we use the following algorithm:

1. Let $\varphi = \neg\psi$ be a TECTL formula,
2. Iterate for $k := 1$ to $|M|$.
3. Select the $k-$model M_k of M.
4. Select the submodels M'_k of M_k with $|Path'_k| \leq f_k(\varphi)$, where f_k is a function of k and φ (see Definition 15).
5. Translate the transition relation of the $k-$paths of all of the submodels M'_k to a propositional formula $[M^{\varphi,w^0}]_k$.
6. Translate φ over all M'_k to a propositional formula $[\varphi]_{M_k}$.
7. Check the satisfiability of $[M, \varphi]_k := [M^{\varphi,w^0}]_k \wedge [\varphi]_{M_k}$.

Hence, checking φ over M_k is translated to checking the satisfiability of the propositional formula $[M^{\varphi,w^0}]_k \wedge [\varphi]_{M_k}$. So that one can use the most efficient algorithms for solving the SAT-problem.

We assume that $\mathcal{W} \subseteq \{0,1\}^n$ and $n = \lceil log_2(|\mathcal{W}|)\rceil$. So, each state can be represented by a vector of state variables $\boldsymbol{w} = (\boldsymbol{w}[1], \ldots, \boldsymbol{w}[n])$, where $\boldsymbol{w}[i]$ is a propositional variable for $i = 1, \ldots, n$. Moreover, a finite sequence $\boldsymbol{w}_0, \ldots, \boldsymbol{w}_k$ of vectors of state variables we call a *symbolic $k-$path*.

Hereafter, let $\mathcal{SV}$ be a set of the state variables containing the symbols $true$ and $false$, and let $\mathcal{SF}$ be a set of propositional formulas built over $\mathcal{SV}$. Moreover, let $LL^\varphi \subset \mathbb{N}_+$ be a finite set of natural numbers. The elements of LL^φ are used as the indices of the symbolic $k-$paths used for translating the transition relation of the submodels M'_k.

To construct $[M, \varphi]_k$, we first define a propositional formula $[M^{\varphi,w^0}]_k$ that constrains $|LL^\varphi|$ symbolic k-paths to represent k-paths of M_k. For $j \in LL^\varphi$, the j-th symbolic $k-$path is denoted by $\boldsymbol{w}_{0,j}, \ldots, \boldsymbol{w}_{k,j}$, where $\boldsymbol{w}_{i,j}$, for $i = 1, \ldots, k$, are vectors of state variables. Then, we translate the TECTL formula φ to a propositional formula that constrains the sets of $|LL^\varphi|$ symbolic k-paths that satisfy φ.

Let $lit : \mathcal{SV} \times \{0,1\} \rightarrow \mathcal{SF}$ be a function defined as follows: $lit(r,0) = \neg r$ and $lit(r,1) = r$. Furthermore, let $\boldsymbol{w}, \boldsymbol{w}'$ be vectors of state variables. We define the following propositional formulas:

- $dead(\boldsymbol{w})$ iff $\boldsymbol{w}$ is a deadlock,
- $I_w(\boldsymbol{w})$ iff $\bigwedge_{i=1}^{n} lit(\boldsymbol{w}[i], w[i])$; encodes a state w of the model M,
- $T(\boldsymbol{w}, \boldsymbol{w}')$ iff $\boldsymbol{w} \rightarrow \boldsymbol{w}'$ or $(dead(\boldsymbol{w}) \wedge \boldsymbol{w} = \boldsymbol{w}')$,
- $p(\boldsymbol{w})$ iff $p \in \mathcal{V}'(\boldsymbol{w})$, for $p \in \mathcal{PV}'$,
- $H(\boldsymbol{w}, \boldsymbol{w}')$ iff $\boldsymbol{w} = \boldsymbol{w}'$,

– $L_{k,j}(l)$ iff $T(\boldsymbol{w}_{k,j}, \boldsymbol{w}_{l,j})$; encodes a backward loop from the $k-$th state to the $l-$th state in the symbolic $k-$path j, for $0 \leq l \leq k$.

Definition 16. *Let M be a model, w be a state of M and φ be a* TECTL *formula. The propositional formula $[M^{\varphi,w}]_k$, for $|LL^{\varphi}| = f_k(\varphi)$, is defined as follows:*

$$[M^{\varphi,w}]_k := I_w(\boldsymbol{w}_{0,0}) \wedge \bigwedge_{j \in LL^{\varphi}} \Big(\bigwedge_{i=0}^{k-1} T(\boldsymbol{w}_{i,j}, \boldsymbol{w}_{i+1,j}) \Big)$$

where $\boldsymbol{w}_{0,0}$ and $\boldsymbol{w}_{i,j}$ for $i = 0, \ldots, k$ and $j \in LL^{\varphi}$ are vectors of state variables.

The above formula encodes all the k-paths generated by the transition relation T, as well as the initial vector $\boldsymbol{w}_{0,0}$ to be equal to w.
By $\boldsymbol{w}_{m_r,n_r}$ we denote the vector of state variables representing the states that are like the ones represented by $\boldsymbol{w}_{m,n}$ except for the value of the clock x_{n+r}, which is reset to 0. The translation of a TECTL formula is similar to the untimed case, since all the temporal subformulas of φ are interpreted at the initial states of the considered new k-paths starting at states $\boldsymbol{w}_{m_r,n_r}$. This means that no loop can return to a state preceding the present state.
We use $[\varphi]_k^{[m,n]}$ to denote the translation of a TECTL formula φ at $\boldsymbol{w}_{m,n}$ to a propositional formula.

(1) Translation of a TECTL **formula**

$$[p]_k^{[m,n]} := p(\boldsymbol{w}_{m,n}), \quad [\alpha \wedge \beta]_k^{[m,n]} := [\alpha]_k^{[m,n]} \wedge [\beta]_k^{[m,n]},$$
$$[\neg p]_k^{[m,n]} := \neg p(\boldsymbol{w}_{m,n}), \; [\alpha \vee \beta]_k^{[m,n]} := [\alpha]_k^{[m,n]} \vee [\beta]_k^{[m,n]},$$
$$[\mathrm{E}(\alpha \mathrm{U}_{I_r} \beta)]_k^{[m,n]} := (let \;\; \beta' := \beta \wedge p_{x_{n+r} \in I_r}) \bigvee_{i \in LL^{\varphi}} \Big(H(\boldsymbol{w}_{m_r,n_r}, \boldsymbol{w}_{0,i}) \wedge \bigvee_{j=0}^{k} ([\beta']_k^{[j,i]} \wedge \bigwedge_{t=0}^{j-1} [\alpha \vee \beta']_k^{[t,i]}) \Big),$$
$$[\mathrm{EG}_{I_r} \alpha]_k^{[m,n]} := (let \;\; \alpha' := \alpha \vee \neg p_{x_{n+r} \in I_r}) \;\; \bigvee_{i \in LL^{\varphi}} \Big(H(\boldsymbol{w}_{m_r,n_r}, \boldsymbol{w}_{0,i}) \wedge \big(\bigvee_{l=0}^{k} L_{k,i}(l) \vee dead(w_{k,i}) \big) \;\wedge\; \bigwedge_{j=0}^{k} [\alpha']_k^{[j,i]} \Big).$$

Let $[\varphi]_{M_k}$ denote $[\varphi]_k^{[0,0]}$.
The following theorem shows the correctness of our translation.

Theorem 2. *Let $M = ((\mathcal{W}, \rightarrow, w^0), \mathcal{V}')$ be a model, M_k be a $k-$model of M, and φ be a* TECTL *formula. Then, $M \models_k \varphi$ iff $[\varphi]_{M_k} \wedge [M^{\varphi,w^0}]_k$ is satisfiable.*

Proof (Sketch). Follows from the following facts:

– The model checking problem of TECTL can be translated to the model checking problem for ECTL$^{\mathrm{F}}$ [1,36].
– Since we deal with progressive Timed Automata only, the above translation can be made to ECTL model checking. Our definition of the Translation (1) is based on this translation.
– Theorem 2 of [35] shows that the translation of the model checking problem for ECTL to the SAT problem is correct.

Corollary 1. *$M \models_k \neg\varphi$ iff $[\varphi]_{M_k} \wedge [M^{\varphi,w^0}]_k$ is unsatisfiable for $k = |M|$.*

7 Implementation of BMC for Timed Automata

In this section we show how BMC can be applied to verification of *Timed Automata*. Following [6], we consider Timed Automata satisfying the following two conditions (**): the space of clocks valuations is bounded by some $c \in \mathbb{N}$, (i.e., range over $C^n = [0, c)^n$) and there is at most one transition associated with every pair of states. The readers can convince themselves that it costs few states to convert any Timed Automaton into one satisfying these properties [6].

Two clock valuations $v = (\lfloor v_1 \rfloor + frac(v_1), \ldots, \lfloor v_n \rfloor + frac(v_n))$ and $v' = (\lfloor v'_1 \rfloor + frac(v'_1), \ldots, \lfloor v'_n \rfloor + frac(v'_n))$ are *zone-equivalent*, written $v \simeq v'$, if

$$\begin{array}{l} \bigwedge_{i=1}^{n} \big((\lfloor v'_i \rfloor = \lfloor v_i \rfloor) \wedge ((frac(v'_i) = 0) \Leftrightarrow (frac(v_i) = 0))\big) \wedge \\ \bigwedge_{i,j \in \{1,\ldots,n\}} \big((frac(v_i) \leq frac(v_j)) \Leftrightarrow (frac(v'_i) \leq frac(v'_j))\big) \end{array}$$

Since the clock valuations are bounded by c, it is obvious that the above definition is equivalent to the Definition 3.

To implement BMC for Timed Automata we have to encode both the transition relation of the region graph of a Timed Automaton and the TECTL formula by propositional formulas. The encoding of the formula was discussed in the previous section. This section shows the encoding of the transition relation.

The method is based on the *discretization* scheme of [6], which preserves the qualitative behaviour of the automaton. Below, we present the above scheme consisting in representing each region of the region graph by one or more appropriately chosen representative states. The set of the representatives of a region is obtained by discretizing the space of clock valuations as follows.

Let $\Delta = 1/(2n)$ be a discretization step and $\widetilde{C} = \{m\Delta \mid 0 \leq m < 2nc\}$. In other words, we cut every unit interval into $2n$ equal segments and pick the endpoints. The discretized clock space (the domain over which discretized clocks range) is $\widetilde{\mathbb{B}^n} = \widetilde{C^n} \cap \{(v_1, \ldots, v_n) \mid \forall_{i,j \in \{1,\ldots,n\}} |v_i - v_j| = 2m\Delta,\ m \geq 0\}$.

Note that we take from $\widetilde{C^n}$ only points such that the difference between any pair of clock valuations is an even multiple of Δ (see Figure 1).

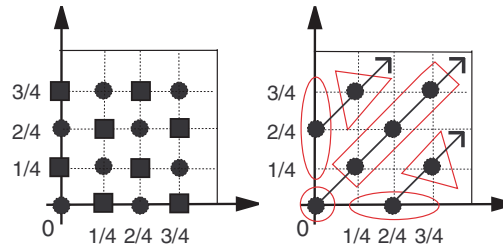

Fig. 1. Left: discretizing $[0, 1)^2$: the circle points are the elements of $\widetilde{\mathbb{B}^n}$ while the squared points belong to $\widetilde{C^n} \setminus \widetilde{\mathbb{B}^n}$. Right: illustration of the Definition 17 and Lemma 2.

For any zone Z, its discretization is defined as $\widetilde{Z} = Z \cap \widetilde{\mathbb{B}^n}$. A discretized zone is called a *d−zone*. It is not hard to see that, for every zone Z, we have $Z \neq \emptyset$ iff $\widetilde{Z} \neq \emptyset$. Another important property of this scheme is the following:

Lemma 1 ([6]). *Let $v = \widetilde{v} + \epsilon$, for some $\widetilde{v} \in \widetilde{\mathbb{B}^n}$ and $|\epsilon| < \Delta$. Then, $v \in Z \Rightarrow (\widetilde{v} \in Z \vee \widetilde{v} + \Delta \in Z)$. (Hence at least one of them belongs also to $\widetilde{Z}$).*

Note that this fails to be true for points outside $\widetilde{\mathbb{B}^n}$. Consider $v = (0, \frac{3}{4})$ and zone $Z = (0 < x < 1) \wedge (0 < y < 1) \wedge (y > x)$. Here $v + \epsilon \in Z$ but neither v nor $v + \Delta = (\frac{1}{4}, 1)$ belong to Z.

The *Discrete Time Successor* of $\widetilde{Z}$, denoted by $\tau(\widetilde{Z})$, is the restriction of $\tau(Z)$ to points in $\widetilde{\mathbb{B}^n}$ and times values in $\widetilde{C}$.

Definition 17 (Discrete Time Successor). *Let $\widetilde{Z}$, $\widetilde{Z'}$ be two distinct d-zones. The d-zone $\widetilde{Z'}$ is the* discrete time successor *of $\widetilde{Z}$ iff for each $v \in \widetilde{Z}$ there exists $\delta \in \widetilde{C}$ such that $v + \delta \in \widetilde{Z'}$ and $v + \delta' \in \widetilde{Z} \cup \widetilde{Z'}$ for all $\delta' \in \widetilde{C}$ with $\delta' \leq \delta$.*

Lemma 2 ([6]. Discretization Preserves Time Successor). *For every zone Z and Z' if $Z' = \tau(Z)$, then $\widetilde{Z'} = \tau(\widetilde{Z})$.*

Obviously, the reverse of Lemma 2 also holds.
Below we show that it is sufficient to represent each zone by one representative.

Lemma 3. *Let $\widetilde{Z}$, $\widetilde{Z'}$ be two distinct d-zones and $v \in \widetilde{Z}$ be any representative of $\widetilde{Z}$. $\widetilde{Z'} = \tau(\widetilde{Z})$ iff there exists $\delta \in \widetilde{C}$ such that $v + \delta \in \widetilde{Z'}$ and $v + \delta' \in \widetilde{Z} \cup \widetilde{Z'}$ for all $\delta' \in \widetilde{C}$ with $\delta' \leq \delta$.*

Proof. (=>) The proof is obvious.
(<=) Let $v = (v_1, \ldots, v_n) \in \widetilde{Z}$ and there exists $\delta \in \widetilde{C}$ such that $v + \delta \in \widetilde{Z'}$ and $v + \delta' \in \widetilde{Z} \cup \widetilde{Z'}$ for all $\delta' \in \widetilde{C}$ with $\delta' \leq \delta$. We show, that $\widetilde{Z'} = \tau(\widetilde{Z})$.

Let $u = (u_1, \ldots, u_n) \in \widetilde{Z}$ and $u \neq v$. Since u and v belong to the same d-zone $\widetilde{Z}$, then their integral parts are equal and also the ordering of the fractional parts is the same. Moreover, for all $i \in \{1, \ldots, n\}$ we have $0 \leq | v_i - u_i | < 1$ (i.e., $0 \leq | frac(v_i) - frac(u_i) | < 1$) and $frac(v_i) - frac(u_i) = m\Delta$ for $m \in \{-2n + 1, \ldots, 2n-1\}$. Hence, $frac(v_i) = frac(u_i) + m\Delta$ for all $i \in \{1, \ldots, n\}$. Therefore, $v + \delta = (v_1 + \delta, \ldots, v_n + \delta) = (u_1 + m\Delta + \delta, \ldots, u_n + m\Delta + \delta) = u + (m\Delta + \delta)$. Since $v + \delta \in \widetilde{Z'}$, then $u + (m\Delta + \delta) \in \widetilde{Z'}$. Let $\delta_1 = (m\Delta + \delta)$. It is obvious, that $u + \delta_1' \in \widetilde{Z} \cup \widetilde{Z'}$ for all $\delta_1' \in \widetilde{C}$ with $\delta_1' \leq \delta_1$. Since u has been an arbitrary valuation different from v, we conclude that $\widetilde{Z'} = \tau(\widetilde{Z})$.

This discretization is not closed under the reset of the values of the clocks - for example, resetting the first coordinate of $(\Delta, \Delta) \in \widetilde{\mathbb{B}^2}$ we obtain $(0, \Delta) \in \widetilde{C^2} \backslash \widetilde{\mathbb{B}^2}$. This is important, because Lemma 2 does not hold on $\widetilde{C^n}$, but only on $\widetilde{\mathbb{B}^n}$. In order to calculate the *Discrete Action Successors* of a d-zone $\widetilde{Z}$ we delete points of $\widetilde{Z}$ that went out of $\widetilde{\mathbb{B}^n}$ and add one or more zone-equivalent points from $\widetilde{\mathbb{B}^n}$.

Before we give the definition of the discrete action successors, we define the following operations on zones and its discretization:
$Z[Y := 0] = \{v[Y := 0] \mid v \in Z\}, \quad \widetilde{Z}[Y := 0] = \{v[Y := 0] \mid v \in \widetilde{Z}\}.$

Note that $\widetilde{Z}[Y := 0]$ is a subset of $\widetilde{C^n}$, but not necessarily a subset of $\widetilde{\mathbb{B}^n}$.

Definition 18 (Discrete Action Successor). *Let $\widetilde{Z}$, $\widetilde{Z'}$ be two d-zones. The d-zone $\widetilde{Z'}$ is said to be the discrete action successor of d-zone $\widetilde{Z}$ by transition $e : s \xrightarrow{l,\psi,Y} s' \in E$ (denoted $e(\widetilde{Z})$) iff $\widetilde{Z} \subseteq \widetilde{\mathbb{p}(\psi)}$ and $e(\widetilde{Z}) = \{v' \in \widetilde{\mathbb{B}^n} \mid v' \simeq v$ and $v \in \widetilde{Z}[Y := 0]\}$.*

Note that $e(\widetilde{Z}) = e(Z) \cap \widetilde{\mathbb{B}^n}$.

Lemma 4 (Discretization for Action Successor). *For every zone Z and Z' if $Z' = e(Z)$, then $\widetilde{Z'} = e(\widetilde{Z})$.*

Proof. Let $Z' = e(Z)$. By the definition of the Action Successor, Z' is the action successor of Z by the transition $e : s \xrightarrow{l,\psi,Y} s' \in E$, iff $Z \subseteq \mathbb{p}(\psi)$ and $Z' = Z[Y := 0]$. Since, $\widetilde{Z} \subseteq Z$ and $\widetilde{\mathbb{p}(\psi)} \subseteq \mathbb{p}(\psi)$, it is obvious that $\widetilde{Z} \subseteq \widetilde{\mathbb{p}(\psi)}$. Now, we show that $\widetilde{Z'} = e(\widetilde{Z})$.
($e(\widetilde{Z}) \subseteq \widetilde{Z'}$) Let $v' \in e(\widetilde{Z})$. By Definition 18 there is $v \in \widetilde{Z}[Y := 0]$ such that $v \simeq v'$. Hence, by discretization $v \in \widetilde{Z'}$.
($\widetilde{Z'} \subseteq e(\widetilde{Z})$) Let $v' \in \widetilde{Z'}$. By discretization and by assumption $Z' = e(Z)$, we have that $v' \in e(Z) \cap \widetilde{\mathbb{B}^n}$. Thus, $v' \in e(Z)$ and $v' \in \widetilde{\mathbb{B}^n}$. Since $v' \in e(Z)$, $v' = v[Y := 0]$ for some $v \in Z$. So, it is clear that $v' \in Z[Y := 0]$. Therefore, from $v' \in \widetilde{\mathbb{B}^n}$ and $v' \in Z[Y := 0]$, it follows that $v' \in Z[Y := 0] \cap \widetilde{\mathbb{B}^n}$. So, $v' \in e(\widetilde{Z})$.

Obviously, the reverse of Lemma 4 also holds.

We are given a Timed Automaton $\mathcal{A} = (\Sigma, S, s^0, E, \mathrm{X}, \mathbb{I})$, a valuation function $\mathcal{V}$, and a TECTL formula φ. We assume that $\mathcal{A}$ is progressive and satisfies the conditions (**). Let $\mathcal{A}''$ be the Timed Automaton like $\mathcal{A}'$ defined in Section 6, but converted to satisfy the conditions (**) [1].

Hereafter, we assume that $\mathcal{A}''$ has altogether n clocks and $\widetilde{\mathcal{A}''}$ denotes the automaton $\mathcal{A}''$ with the discretized clock space. Let M be the model based on the region graph of $\mathcal{A}''$ and φ.

Definition 19. *The* region graph *for the Timed Automaton $\widetilde{\mathcal{A}''}$ and φ is a finite structure $\widetilde{\mathcal{RG}}(\widetilde{\mathcal{A}''}, \varphi) = (\widetilde{\mathcal{W}}, \rightarrow, \widetilde{w^0})$, where $\widetilde{\mathcal{W}} = \{(s, \widetilde{Z}) \mid (s, Z) \in S \times Z(n)\}$, $\widetilde{w^0} = (s^0, \widetilde{Z^0})$, where $Z^0 = [v^0]$ and $\rightarrow \subseteq \widetilde{\mathcal{W}} \times (E \cup \{\tau\}) \times \widetilde{\mathcal{W}}$ is defined as follows:*

- $(s, \widetilde{Z}) \xrightarrow{e} (s', \widetilde{Z'})$ *iff* $\widetilde{Z'} = e(\widetilde{Z})$, $s = source(e)$, $s' = target(e)$, *and* $\widetilde{Z'} \subseteq \widetilde{\mathbb{p}(\mathbb{I}(s'))} \cap \widetilde{\mathbb{B}^n}$, *for* $e \in E$,
- $(s, \widetilde{Z}) \xrightarrow{\tau} (s, \widetilde{Z'})$ *iff* $\widetilde{Z'} \subseteq \widetilde{\mathbb{p}(\mathbb{I}(s))} \cap \widetilde{\mathbb{B}^n}$ *and* $\widetilde{Z'} = \tau(\widetilde{Z})$.

[1] This conversion is usually not necessary as we state in Section 8

The discretized model based on the region graph of $\widetilde{\mathcal{A}''}$ *and a* TECTL *formula* φ *is defined as* $\widetilde{M} = (\widetilde{\mathcal{RG}}(\widetilde{\mathcal{A}''}, \varphi), \widetilde{\mathcal{V}''})$*, where* $\widetilde{\mathcal{V}''} : \widetilde{\mathcal{W}} \longrightarrow 2^{\mathcal{P}\mathcal{V}'}$ *with* $\widetilde{\mathcal{V}''}((s, \widetilde{Z})) = \mathcal{V}''((s, Z))$*, where* $\mathcal{V}''$ *is the valuation function for* $\mathcal{A}''$*, defined like* $\mathcal{V}'$ *for* $\mathcal{A}'$ *in Section 6.*

Since the discretized model $\widetilde{M}$ is isomorphic with the model M for $\mathcal{A}''$ and φ, we have $\widetilde{M} \models_k \varphi$ iff $M \models_k \varphi$, for TECTL formula φ.

Now, we can construct a propositional formula $[\widetilde{M}, \varphi]_k$ that is satisfiable iff $\widetilde{M} \models_k \varphi$. An implementation of $[\widetilde{M}, \varphi]_k$ is shown in the appendix.

8 Generalizations and Further Optimizations

As to unrestricted Timed Automata we can reduce the verification of TECTL formulas to progressive runs only, by defining the translation from TECTL to ECTL^{F}, i.e., the fair version of ECTL, and adapting the translation from ECTL to SAT to deal with the fair paths.

While we construct the automaton $\mathcal{A}'$ we add the new m clocks, which correspond to the non-trivial intervals appearing in φ. Since these clocks are never reset the valuations of them are unbounded. It is however not necessary to convert $\mathcal{A}'$ to an automaton with the bounded clock space. Note that in our bounded semantics the valuations of the clocks are always bounded. However, dealing directly with the unconverted $\mathcal{A}'$ has a problem of not closing loops by time successors. This can be compensated by using one of the following two methods.

The first method relies on freezing the value of each new clock x_i when x_i exceeds the maximal integer bound of the interval I_i. An alternative solution can be applied when checking formulas indexed with intervals of the form $(a, b), (a, b], [a, b)$ or $[a, b]$. To each location of $\mathcal{A}'$ we add the self-loop with the enabling conditions ψ defined as follows. Let b_i be the right bound of the interval I_i, and $N_{\text{X}'\setminus\text{X}}$ be a set of indices of the clocks from $\text{X}' \setminus \text{X}$. Then, $\psi := \bigwedge_{i \in N_{\text{X}'\setminus\text{X}}} x_i \geq b_i$. For our example, we have used the latter solution.

In the final version of the implementation we are going to use the former method, which allows to encode and discretize open zones. Then, we would not need to convert Timed Automata to satisfy the conditions (**), avoiding an increase in the number of state variables.

It is known that some aggregation of time successors can be made in the region graph model [25] without influencing the validity of a tested formula. This allows to find counterexamples on shorter paths.

9 Experimental Results

We have implemented a model checker BBMC based on bounded model checking for TACTL. For input consisting of a description of a Timed Automaton $\mathcal{A}$ and a TACTL formula ψ expressing some desired specification, BBMC first negates the formula ψ obtaining the TECTL formula φ. Then, it builds the propositional

formula $\mu = [\varphi]_{M_k} \wedge [M^{\varphi,w^0}]_k$, which is satisfiable iff the formula φ is valid in the model of $\mathcal{A}'$. Given the formula μ, the model checker outputs the set of clauses C. Each clause is a set of literals, where each literal is either a positive or a negative propositional variable. Notice, that the size of the set C can be exponential with respect to the size of μ. In order to avoid the exponential explosion we use a structure preserving transformation, such that the resulting set C is satisfiable iff the formula μ is satisfiable, although C is not logically equivalent to the original formula μ.

The output format for the set of clauses C generated by our model checker is in the DIMACS format [23] for satisfiability problems. We have made use of the satisfiability solver ZChaff [28,39,40], which uses the DIMACS format. It is an implementation of the Chaff solver, maintained by Lintao Zhang. ZChaff is known to solve problems with more than one million variables and 10 million clauses.

We have performed our experiments on the IBM PC compatible computer equipped with the processor AMD Duron 800 MHz, 256 MB main memory and the operating system Windows 98. We have tested the Timed Automaton describing the standard *railroad crossing system* (RCS, for short).

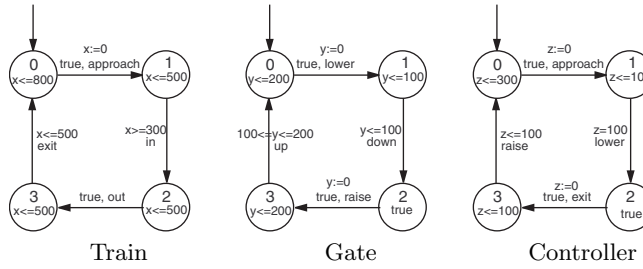

Fig. 2. Timed Automaton for Train, Gate, and Controller

This is a well-known example in the literature of real-time verification (see [24]). The system consists of three components, Train, Gate and Controllers, as shown in Figure 2, which execute in parallel and synchronize through the events: *approach, exit, lower* and *down*. When a train approaches the crossing, Train sends an *approach* signal to Controller and enters the crossing at least 300 seconds later. When a train leaves the crossing, Train sends an *exit* signal to Controller. The *exit* signal is sent within 500 seconds after the *approach* signal. Controller sends a signal *lower* to Gate exactly 100 seconds after the *approach* signal and sends a *raise* signal within 100 seconds after *exit*. Gate responds to *lower* by moving *down* within 100 seconds and responds to *raise* by moving *up* between 100 and 200 seconds. The product Timed Automaton composed of Train, Gate and Controllers is shown in Figure 3. A node (i, j, k) represents that Train, Gate, and Controllers are at nodes i, j and k, respectively. The invariant of a node is the conjunction of the invariants of the three components. For input, our model checker takes the encoded transition relation of the product Timed Automaton of Figure 3.

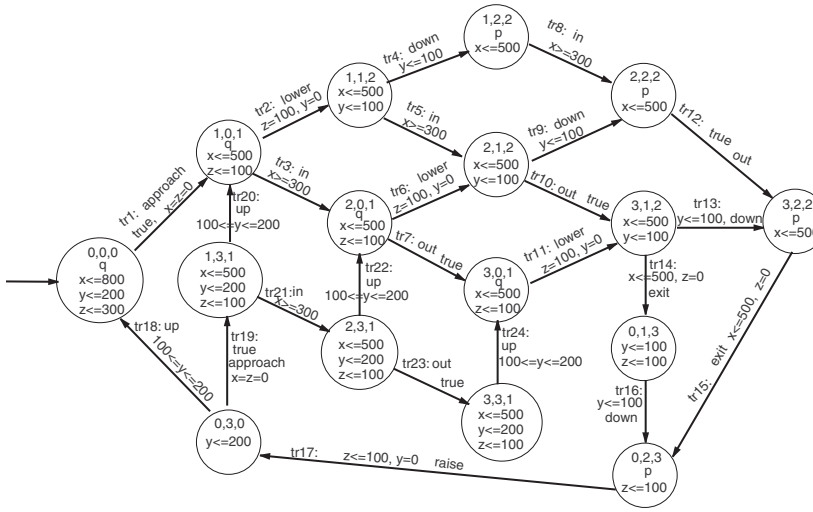

Fig. 3. Timed Automaton for Railroad Crossing System (RCS)

Let M_{RCS} be the model of the Timed Automaton for RCS. For the railroad crossing system, we want to verify the following utility property: whenever the gate is down, it is moved back up within K seconds for some K. This property is given by the TACTL formula: $\psi = \mathrm{AG}(p \Rightarrow \mathrm{AF}_{<K} q)$ with the proposition p true at the states of M_{RCS} when the gate is down, and the proposition q true at the states of M_{RCS} when the gate is moved back up.

Without loss of generality we can assume that the time unit is equal to 100 seconds. So, this means that we test $\psi = \mathrm{AG}(p \Rightarrow \mathrm{AF}_{<\frac{K}{100}} q)$ instead of $\psi = \mathrm{AG}(p \Rightarrow \mathrm{AF}_{<K} q)$, and each value c, which appears in the guards and the invariants, is divided by 100.

Note that, for each $k > 0$ the value of the function f_k for the formula $\varphi = \neg\psi$ (i.e., $\mathrm{EF}(p \wedge \mathrm{EG}_{<K} \neg q)$) is equal to 2. This means that a counterexample, if exists, can be found on two symbolic $k-$paths. Moreover, it is easy to see, that the length of a counterexample, i.e., the value of k, depends on K. In Table 1 we show the experimental results for two symbolic $k-$paths. The first column of Table 1 shows the right bound of the interval appearing in the subscript of our formula, (i.e., the interval $[0, K)$); the second shows the length of the symbolic $k-$paths, (i.e. the value k); the third and fourth show the numbers of propositional variables and clauses generated by BBMC respectively; the fifth and sixth show the time and memory consumed by BBMC to generate the set of clauses; the next two columns show the time and the memory consumed by ZChaff, and the last shows the solution of our problem, i.e., satisfiable iff $M_{RCS} \not\models_k \psi$ (i.e., satisfiable if $M_{RCS} \not\models \psi$).

Since our method is devoted for finding an error, for the subscript $K >= 7$ we do not present the results, because the property: *whenever the gate is down, it is moved back up within 700 seconds, i.e., 7 time unit*, is true [24].

Table 1. Experimental results for RCS system on two symbolic $k-$paths

		BBMC				ZChaff		
K	**k**	**variables**	**clauses**	**sec**	**MB**	**sec**	**MB**	Solution
1	1	9697	28828	4.8	3.88	0.6	2.75	unsatisfiable
1	4	26275	79060	40.6	33.06	6.8	9.06	unsatisfiable
1	5	31801	95804	64.5	57.06	24.3	11.06	satisfiable
2	1	9701	28840	4.8	5.31	0.3	2.63	unsatisfiable
2	4	26285	79090	40.6	26.50	23.7	9.81	unsatisfiable
2	5	31813	95840	64.7	49.38	7.5	9.75	satisfiable
3	1	9685	28792	4.8	5.25	0.3	2.63	unsatisfiable
3	5	31765	95696	64.8	53.50	54.0	14.62	unsatisfiable
3	6	37285	112422	97.0	69.31	63.0	15.69	satisfiable
4	1	9703	28846	4.8	4.94	0.3	2.63	unsatisfiable
4	6	37353	112626	97.3	61.38	312.6	39.75	unsatisfiable
4	8	48413	146138	191.0	106.90	178.0	42.62	satisfiable
5	1	9689	28804	4.8	6.38	0.4	2.69	unsatisfiable
5	6	37299	112464	97.1	62.88	234.7	39.88	unsatisfiable
5	12	70431	212856	622.3	175.70	3593.0	144.00	satisfiable
6	1	9693	28816	4.8	5.38	0.4	2.69	unsatisfiable
6	8	48361	145982	203.3	108.60	502.7	74.69	unsatisfiable
6	16	92553	279886	1527.0	154.80	5054.0	147.80	satisfiable

10 Final Remarks

We have shown that the BMC method for the logic TACTL is feasible. Our method has been implemented and checked on several examples. In Section 9 we gave experimental results for the system RCS finding counterexamples on paths of length up to 16. This experiment is maybe not sufficiently convincing to the reader about the efficiency of our method since the model of RCS is only of degree 3, i.e., the maximal branching of its states is equal to 3. However, we believe that using our method will allow us to search for errors in timed systems, for which complete minimal bisimulation state spaces cannot be generated (e.g. *the mutual exclusion protocol* for 7 processes [36]). This is motivated by the fact that we do not need to generate the whole model and can encode the transition relation in a very efficient way.

So far the transition relation of the product automaton has been an input for our implementation. This solution is obviously not efficient for Timed Automata composed of many components, like the above mentioned MUTEX, since the explosion in states and transitions is already present in the product automaton itself. Therefore, we are planning to change the input so that the translation of the transition relation of the product automaton is obtained from the local transition relations of the components.

Obviously exploiting region graph models for model checking of TCTL is not the best possible option, even if only parts of models are used, as we do it. Therefore, we are working on replacing region graphs with minimized bisim-

ulation graphs. This is, however, far non-trivial due to the fact that minimal bisimulation models cannot be built by the DFS or BFS algorithm on which the translation of the transition relation is based. Moreover, it seems likely that our method can be efficiently used for detecting deadlocks and checking reachability in Timed Automata using surjective (simulating) models.

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
2. R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proc. of RTSS'92*, p. 157–166. IEEE Comp. Soc. Press, 1992.
3. R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. Minimization of timed transition systems. In *Proc. of CONCUR'92*, volume 630 of *LNCS*, p. 340–354. Springer-Verlag, 1992.
4. R. Alur and D. Dill. A theory of Timed Automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994.
5. R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.
6. E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of Timed Automata. In *Proc. of HART'97*, volume 1201 of *LNCS*, p. 346–360. Springer-Verlag, 1997.
7. J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *Proc. of CONCUR'98*, volume 1466 of *LNCS*, p. 485–500. Springer-Verlag, 1998.
8. A. Biere, A. Cimatti, E. Clarke, M.Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. of DAC'99*, p. 317–320, 1999.
9. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. of TACAS'99*, volume 1579 of *LNCS*, p. 193–207. Springer-Verlag, 1999.
10. A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model checking for real-time systems. In *Proc. of RTSS'97*, p. 232 – 243. IEEE Comp. Soc. Press, 1997.
11. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(8):677–691, 1986.
12. E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. In *Formal Methods in System Designe*, volume 19(1), p. 7–34, 2001.
13. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
14. D. Dams, R. Gerth, B. Knaack, and R. Kuiper. Partial-order reduction techniques for real-time model checking. In *Proc. of the 3rd Int. Workshop on Formal Methods for Industrial Critical Systems*, p. 157 – 169, 1998.
15. D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving ACTL*, ECTL* and CTL*. In *Proc. of PROCOMET'94*. Elsevier Science Publishers, 1994.
16. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *Proc. of TACAS'98*, volume 1384 of *LNCS*, p. 313–329. Springer-Verlag, 1998.

17. Richard H. Eckhouse. *Minicomputer systems. Organization and programming.* Prentice-Hall, 1975.
18. E. A. Emerson. *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, chapter Temporal and Modal Logic, p. 995–1067. Elsevier Science Publishers, 1990.
19. E. A. Emerson and E. M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.
20. E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9:105–131, 1995.
21. O. Grumberg and D. E. Long. Model checking and modular verification. In *Proc. of CONCUR'91*, volume 527 of *LNCS*, p. 250–265. Springer-Verlag, 1991.
22. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. In *Proc. of LPNMR'2001*, volume 2173 of *LNCS*, p. 200–212. Springer-Verlag, 2001.
23. D. S. Johnson and M. A. Trick, editors. *Cliques, Coloring and Satisfiability: The Second DIMACS Implementation Challenge*, volume 26 of *ACM/AMS DIMACS Series*. Amer. Math. Soc., 1996.
24. I. Kang and I. Lee. An Efficient State Space Generation for the Analysis of Real-time Systems. In *Proc. of Int. Symposium on Software Testing and Analysis*, 1996.
25. O. Kupferman, T. A. Henzinger, and M. Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In *Proc. of CONCUR'96*, volume 1119 of *LNCS*, p. 514–529. Springer-Verlag, 1996.
26. J. Lilius. Efficient state space search for Time Petri Nets. In *Proc. of MFCS*, volume 18 of *ENTCS*. Elsevier Science Publishers, 1999.
27. K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.
28. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proc. of DAC'01*, June 2001.
29. F. Pagani. Partial orders and verification of real-time systems. In *Proc. of FTRTFT'96*, volume 1135 of *LNCS*, p. 327–346. Springer-Verlag, 1996.
30. D. Peled. Partial order reduction: Linear and branching temporal logics and process algebras. In *Proc. of POMIV'96*, volume 29 of *ACM/AMS DIMACS Series*, p. 79–88. Amer. Math. Soc., 1996.
31. W. Penczek and A. Półrola. Abstractions and partial order reductions for checking branching properties of Time Petri Nets. In *Proc. of ICATPN'01*, volume 2075 of *LNCS*, p. 323–342. Springer-Verlag, 2001.
32. W. Penczek, M. Szreter, R. Gerth, and R. Kuiper. Improving partial order reductions for universal branching time properties. *Fundamenta Informaticae*, 43:245–267, 2000.
33. W. Penczek and B. Woźna. Towards bounded model checking for Timed Automata. In *Proc. of CS&P'01*, p. 195–209, 2001.
34. W. Penczek, B. Woźna, and A. Zbrzezny. Bounded Model Checking for the Universal Fragment of CTL. *Fundamenta Informaticae*, 2002. to appear.
35. W. Penczek, B. Woźna, and A. Zbrzezny. Branching Time Bounded Model Checking for Elementary Net Systems. *Report ICS PAS*, 940, January 2002.
36. S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.
37. P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In *Proc. of CONCUR'93*, volume 715 of *LNCS*, p. 233–246. Springer-Verlag, 1993.
38. T. Yoneda and B.H. Schlingloff. Efficient verification of parallel real-time systems. *Formal Methods in System Design*, 11(2):197–215, 1997.

39. L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proc. of ICCAD'01*, Nov. 2001.
40. Lintao Zhang. Zchaff. http://www.ee.princeton.edu/∼chaff/zchaff.php, 2001.

11 Appendix

Here, we give a detailed description of the implementation of the propositional formula $[\widetilde{M,\varphi}]_k$. Let's start with introducing the following auxiliary functions.

- $\Theta_t : [0,\ldots,2^t{-}1] \to \{0,1\}^t$ is a function, which converts each natural number smaller than 2^t to the bit-vector of the length t.
- $BP : \{0,1\} \to \mathcal{SV}$ is an injective function, which converts a bit to the (const) state proposition such that $BP(0) = false$ and $BP(1) = true$.
- $sum_t : \mathcal{SV}^t \times \{0,1\}^t \to \mathcal{SF}^t$ is a function that adds a bit-vector to a vector of state variables in the binary way, defined as follows.
  Let $u = (u[1],\ldots,u[t])$ be a vector of state variables, $b = (b[1],\ldots,b[t])$ be a bit-vector and $a = (a[1],\ldots,a[t])$ be an auxiliary vector of propositional formulas, which encodes the move-bits for the vectors u and b, defined as:
  - $a[1] := false$,
  - $a[i] := (u[i{-}1] \wedge BP(b[i{-}1])) \vee (u[i{-}1] \wedge a[i{-}1]) \vee (BP(b[i{-}1]) \wedge a[i{-}1])$, for $i = 2,\ldots,t$.

  Let $p, q \in \mathcal{SF}$. Define the operator $\oplus$ as $p \oplus q := \neg(p \Leftrightarrow q)$. Then,

$$sum_t(u,b) = (u[1] \oplus BP(b[1]) \oplus a[1],\ldots,u[t] \oplus BP(b[t]) \oplus a[t])$$

- $neg_t : \{0,1\}^t \to \{0,1\}^t$ is an one-to-one function encoding the negation of a bit-vector, defined as follows: $neg_t((b[1],\ldots,b[t])) = (\neg b[1],\ldots,\neg b[t])$, where $\neg 1 = 0$ and $\neg 0 = 1$.
- $minus_t : \mathcal{SF}^t \times \{0,1\}^t \to \mathcal{SF}^t$ is a function encoding the binary subtraction, defined as follows. Let u be a vector of propositional formulas, b be a bit-vector, and let the value encoded by u be greater than the value encoded by b.

$$minus_t(u,b) = sum'_t(u, sum'_t(neg_t(b), \Theta_t(1)))$$

  where sum'_t is like sum_t except for the t'th move-bit, which is removed. The definition of the function $minus_t$ is written on the basis of the standard algorithm of the subtraction of the binary numbers, which is used by digital circuits (see [17]).
- Let $\Phi \subseteq \Psi_{X'}$ be a set of clock constraints only of the form $x \sim c$ and $x{-}y \sim c$, where $x, y \in X'$ and $\sim \in \{\leq, <, =, >, \geq\}$, and $c \in \mathbb{N}$. Define the function $scc : \Psi_{X'} \to 2^{\Phi}$ (a *s*traight *c*ondition *c*alculating) as follows:
  - $scc(\text{"}x \sim c\text{"}) = \{\text{"}x \sim c\text{"}\}$
  - $scc(\text{"}x - y \sim c\text{"}) = \{\text{"}x - y \sim c\text{"}\}$
  - $scc(\phi_1 \wedge \phi_2) = scc(\phi_1) \cup scc(\phi_2)$

  The function scc returns the set of clock constraints of the form $x \sim c$ and $x - y \sim c$, (i.e., from the set Φ) for all the clock' constraints.

– Let $\Xi \subseteq \Psi_{\mathrm{X}'}$ be a set of clock constraints of the form $x \sim c$ only, where $x \in \mathrm{X}'$, $\sim \in \{\leq, <, =, >, \geq\}$, and $c \in \mathbb{N}$. Moreover, let INT be the set of the intervals of the form $[a, b]$, $[a, b)$, $(a, b]$, (a, b), (a, ∞), and $[a, \infty)$, for $a, b \in \mathbb{N}$. We define the function $inter : INT \times \mathrm{X}' \to 2^{\Xi}$ as follows:

  - $inter((a, \infty), x) = \{"x > a"\}$,
  - $inter([a, \infty), x) = \{"x \geq a"\}$,
  - $inter([a, b], x) = \{"x \geq a",\ "x \leq b"\}$,
  - $inter([a, b), x) = \{"x \geq a",\ "x < b"\}$,
  - $inter((a, b], x) = \{"x > a",\ "x \leq b"\}$,
  - $inter((a, b), x) = \{"x > a",\ "x < b"\}$.

  The function *inter* returns the set of clock constraints of the form $x \sim c$, (i.e., from the set Ξ) for each interval and clock $x \in \mathrm{X}'$.

Let a, b be vectors of propositional formulas over $\mathcal{SF}$. Define the following auxiliary operator:

– $a = b := \forall_{1 \leq i \leq t}\ a[i] = b[i]$,

and the following auxiliary propositional formulas:

– $zero(a) := \bigwedge_{i=1}^{t} \neg a[i]$
– $gez(a) := \bigvee_{i=1}^{t} a[i]$
– $eq(a, b) := \bigwedge_{i=1}^{t} a[i] \Leftrightarrow b[i]$
– $ge(a, b) := \bigvee_{i=1}^{t} \left(a[i] \wedge \neg b[i] \wedge \bigwedge_{j=i+1}^{t} a[j] \Leftrightarrow b[j]\right)$
– $geq(a, b) := eq(a, b) \vee ge(a, b)$
– $le(a, b) := \neg geq(a, b)$
– $leq(a, b) := \neg ge(a, b)$

Note that above definitions are also correct when a or b are bit-vectors.

To construct the formula $[\widetilde{M}, \varphi]_k$ we have to implement the following propositional formulas:

– I encoding the initial state of M,
– T encoding the transition relation of M, and
– $[\varphi]_{M_k}$ encoding the translation of the formula φ to a propositional formula, which has been already defined in Section 6.

Let $\widetilde{M}$ be a model based on the region graph of $\widetilde{\mathcal{A}''}$ and the TECTL formula φ. By $m = log_2(|S|)$ we denote the number of state variables sufficient to encode the locations of $\widetilde{\mathcal{A}''}$, where $|S|$ is the number of locations of $\widetilde{\mathcal{A}''}$. We encode the elements of $\widetilde{\mathcal{W}}$ by subsets of $\{0,1\}^m \times \{0,1\}^{n \cdot (\lceil log_2(c) \rceil + \lceil log_2(4n) \rceil)}$. This means that a state of $\widetilde{\mathcal{W}}$ can be represented by a pair of vectors of state variables $(\boldsymbol{s}, \boldsymbol{v}) = ((\boldsymbol{s}[1], \ldots, \boldsymbol{s}[m]), (\boldsymbol{v}[1], \ldots, \boldsymbol{v}[r]))$, where $r = n \cdot (\lceil log_2(c) \rceil + \lceil log_2(4n) \rceil)$ and $\boldsymbol{s}[i]$, $\boldsymbol{v}[j]$ are propositional variables, for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, r\}$. Moreover, we require that the values of clocks are given in the binary system as the sum of the integral and the fractional part, but we encode only the integral part and the numerator of the fractional part.

Hereafter, by $\boldsymbol{s}$, $\boldsymbol{s}1$, and $\boldsymbol{s}2$ we mean vectors of state variables representing locations of $\mathcal{A}''$, and by $\boldsymbol{v}j$ (i.e., $\boldsymbol{v}$, $\boldsymbol{v}1$,$\boldsymbol{v}2$) we mean a vector of state variables representing clock valuations.

Each vector $\boldsymbol{v}j$ consists of n "subvectors" $\mathcal{I}_i^{vj}$ with $\lceil log_2(c) \rceil$ state variables representing the integral part of the value of the clock x_i, and n "subvectors" $\mathcal{F}_i^{vj}$ with $\lceil log_2(4n) \rceil$ state variables representing the fractional part of the value of the clock x_i, for $i \in \{1, \ldots, n\}$.

By $\boldsymbol{u}$ we mean a vector of propositional formulas over $\mathcal{SF}$. The vector $\boldsymbol{u}$ like $\boldsymbol{v}$ consists of n "subvectors" $\mathcal{I}_i^u$ with $\lceil log_2(c) \rceil$ propositional formulas representing the integral part of the value of the clock x_i, and n "subvectors" $\mathcal{F}_i^u$ with $\lceil log_2(4n) \rceil$ propositional formulas representing the fractional part of the value of the clock x_i, for $i \in \{1, \ldots, n\}$.

Moreover, let $idx : S \times \{0,1\} \to 2^{\{1,\ldots,m\}}$ be a function defined as follows. $idx(s,i)$ returns the set of indices of the bits of the bit-vector representing s, which are set to i, for $i \in \{0,1\}$.

Now, we define the following propositional formulas:

- It follows from Lemma 3 that we can assume each $d-$zone to be represented by its arbitrary point. So, let $z \in \widetilde{Z}$ be any representative of $\widetilde{Z}$. The formula I is implemented as follows:

  $I_{(s,\widetilde{Z})}((\boldsymbol{s},\boldsymbol{v})) := \bigwedge_{i \in idx(s,1)} \boldsymbol{s}[i] \wedge \bigwedge_{i \in idx(s,0)} \neg \boldsymbol{s}[i] \wedge eq(\boldsymbol{v}, vec(z))$, where

  $vec : \widetilde{\mathbb{B}^n} \to \{0,1\}^r$ is an injective function converting each clock valuation $(v_1, \ldots, v_n)$ to the bit-vector $\boldsymbol{b} = (\boldsymbol{b}_{I_1}, \boldsymbol{b}_{F_1}, \ldots, \boldsymbol{b}_{I_n}, \boldsymbol{b}_{F_n})$ of the length r, where $\boldsymbol{b}$ consists of $2n$ bit-vectors: $\boldsymbol{b}_{I_i}$, $\boldsymbol{b}_{F_i}$ for $i \in \{1, \ldots, n\}$. Each $\boldsymbol{b}_{I_i}$ is the binary representation of the integral part of v_i and $\boldsymbol{b}_{F_i}$ is the binary representation of the numerator of the fractional part of v_i.

To implement the formula T, we first implement the formula AS encoding the discrete action successor relation, and then implement the formula TS encoding the discrete time successor relation. To implement the formula AS we have to define the function *reset* and the formulas *Guard*, *Inv*, and *adjust*. To simplify the notation of the above formulas we start with defining the formulas *Bool* and *even*.

- Let $\phi \in \Phi$, $t = \lceil log_2(c) \rceil$ and $\boldsymbol{a}$ denote the bit-vector of the length t, which is the binary representation of $a \in \{0, \ldots, c\}$. $Bool(\phi, \boldsymbol{u})$:=

$$
\begin{cases}
le(\mathcal{I}_i^u, \boldsymbol{a}), & \text{if } \phi = x_i < a \\
le(\mathcal{I}_i^u, \boldsymbol{a}) \vee \big(eq(\mathcal{I}_i^u, \boldsymbol{a}) \wedge zero(\mathcal{F}_i^u)\big), & \text{if } \phi = x_i \leq a \\
ge(\mathcal{I}_i^u, \boldsymbol{a}) \vee \big(eq(\mathcal{I}_i^u, \boldsymbol{a}) \wedge gez(\mathcal{F}_i^u)\big), & \text{if } \phi = x_i > a \\
geq(\mathcal{I}_i^u, \boldsymbol{a}), & \text{if } \phi = x_i \geq a \\
eq(\mathcal{I}_i^u, \boldsymbol{a}) \wedge zero(\mathcal{F}_i^u), & \text{if } \phi = (x_i = a) \\
le(\mathcal{I}_i^u, sum_t(\mathcal{I}_j^u, \boldsymbol{a})) \vee \big(eq(\mathcal{I}_i^u, sum_t(\mathcal{I}_j^u, \boldsymbol{a})) \wedge le(\mathcal{F}_i^u, \mathcal{F}_j^u)\big), & \text{if } \phi = x_i - x_j < a \\
le(\mathcal{I}_i^u, sum_t(\mathcal{I}_j^u, \boldsymbol{a})) \vee \big(eq(\mathcal{I}_i^u, sum_t(\mathcal{I}_j^u, \boldsymbol{a})) \wedge leq(\mathcal{F}_i^u, \mathcal{F}_j^u)\big), & \text{if } \phi = x_i - x_j \leq a \\
ge(\mathcal{I}_i^u, sum_t(\mathcal{I}_j^u, \boldsymbol{a})) \vee \big(eq(\mathcal{I}_i^u, sum_t(\mathcal{I}_j^u, \boldsymbol{a})) \wedge ge(\mathcal{F}_i^u, \mathcal{F}_j^u)\big), & \text{if } \phi = x_i - x_j > a \\
ge(\mathcal{I}_i^u, sum_t(\mathcal{I}_j^u, \boldsymbol{a})) \vee \big(eq(\mathcal{I}_i^u, sum_t(\mathcal{I}_j^u, \boldsymbol{a})) \wedge geq(\mathcal{F}_i^u, \mathcal{F}_j^u)\big), & \text{if } \phi = x_i - x_j \geq a \\
eq(\mathcal{I}_i^u, sum_t(\mathcal{I}_j^u, \boldsymbol{a})) \wedge eq(\mathcal{F}_i^u, \mathcal{F}_j^u), & \text{if } \phi = x_i - x_j = a
\end{cases}
$$

– $even(\boldsymbol{u}) := \bigwedge_{i,j\in\{1,\ldots,n\}}(\mathcal{F}_i^u[1] \Leftrightarrow \mathcal{F}_j^u[1])$.
The formula *even* encodes all the clocks valuations satisfying the conditions $\forall_{i,j\in\{1,\ldots,n\}}|v_i - v_j| = 2m$ for some $m \geq 0$.
– $Guard(e, \boldsymbol{u}) := \bigwedge_{\phi\in scc(guard(e))} Bool(\phi, \boldsymbol{u}) \wedge even(\boldsymbol{u})$.
Guard is a propositional formula, which encodes the enabling conditions for a transition e of $\widetilde{\mathcal{A}''}$.
– $Inv(s, \boldsymbol{u}) := \bigwedge_{\phi\in scc(\mathbb{I}(s))} Bool(\phi, \boldsymbol{u}) \wedge even(\boldsymbol{u})$.
The above formula encodes the invariant of a location s of $\widetilde{\mathcal{A}''}$.
Let $Y_e \subseteq \{1, \ldots, n\}$ be a finite set of the indices of the clocks to be reset with the transition e. For $e \in E$, let $reset_e : \mathcal{SV}^r \to \mathcal{SV}^r$ be a function defined as follows: $reset_e(\boldsymbol{v}) = \boldsymbol{v}1$, where: $\forall_{i\in Y_e}\big(\forall_{j=1}^{\lceil log_2(c)\rceil}\mathcal{I}_i^{v1}[j] = false \; and \; \forall_{j=1}^{\lceil log_2(4n)\rceil}\mathcal{F}_i^{v1}[j] = false\big)$ and $\forall_{i\in\{1,\ldots,n\}\setminus Y_e}\ \big(\mathcal{I}_i^{v1} = \mathcal{I}_i^v \; and \; \mathcal{F}_i^{v1} = \mathcal{F}_i^v\big)$.
The function $reset_e$ returns the vector of the state variables that constrains all the clock valuations reset with the transition $e \in E$, to be valid valuations in $\widetilde{C^n}$, for each vector of state variables.
In order to calculate the action successors we need to define a propositional formula, called *adjust*, based on the "adjustment" function α from [6]. After applying the reset function, *adjust* replaces each vector of state variables $\boldsymbol{v}1$, representing values of clocks went out of $\widetilde{\mathbb{B}^n}$, by the vector of state variables $\boldsymbol{v}2$, representing values of clocks from $\widetilde{\mathbb{B}^n}$, equivalent to $\boldsymbol{v}1$.

– $adjust(\boldsymbol{v}1, \boldsymbol{v}2) := \begin{cases} \big(eq(\boldsymbol{v}1, \boldsymbol{v}2) \wedge even(\boldsymbol{v}1)\big)\vee \\ \big(\big(\bigvee_{m=0}^{n-1} adjust_m(\boldsymbol{v}1, \boldsymbol{v}2)\big) \wedge \neg even(\boldsymbol{v}1)\big), \end{cases}$
where $adjust_m$ is defined as follows. Let $\boldsymbol{l} = \Theta_{\lceil log_2(4n)\rceil}(l)$, for $l \in \{0, \ldots, 2n\}$.

$$adjust_m(\boldsymbol{v}1, \boldsymbol{v}2) := \begin{cases} \bigwedge_{i=1}^{n} eq(\mathcal{I}_i^{v1}, \mathcal{I}_i^{v2}) \wedge \Big(zero(\mathcal{F}_i^{v1}) \wedge zero(\mathcal{F}_i^{v2})\vee \\ \bigvee_{l=0}^{m-1}\big(eq(\mathcal{F}_i^{v1}, \boldsymbol{2l+1}) \wedge eq(\mathcal{F}_i^{v2}, \boldsymbol{2l+2})\big)\vee \\ \bigvee_{l=m+1}^{n-1}\big(eq(\mathcal{F}_i^{v1}, \boldsymbol{2l+1}) \wedge eq(\mathcal{F}_i^{v2}, \boldsymbol{2l})\big)\Big) \end{cases}$$

– $AS((\boldsymbol{s}1, \boldsymbol{v}1), (\boldsymbol{s}2, \boldsymbol{v}2)) := \begin{cases} \bigvee_{e\in E}\Big(\bigwedge_{i\in idx(source(e),1)} \boldsymbol{s}1[i] \wedge \\ \bigwedge_{i\in idx(source(e),0)} \neg\boldsymbol{s}1[i] \wedge \\ \bigwedge_{i\in idx(target(e),1)} \boldsymbol{s}2[i] \wedge \\ \bigwedge_{i\in idx(target(e),0)} \neg\boldsymbol{s}2[i] \ \wedge \ Guard(e, \boldsymbol{v}1) \wedge \\ Inv(target(e), \boldsymbol{v}2) \ \wedge \ adjust(reset_e(\boldsymbol{v}1), \boldsymbol{v}2)\Big). \end{cases}$

To implement the formula TS, encoding the discrete time successor relation, we have to first define the formulas *equiv* and *cleverSum*.

– $equiv(\boldsymbol{u}, \boldsymbol{v}) := \begin{cases} \bigwedge_{i=1}^{n}\Big(eq(\mathcal{I}_i^v, \mathcal{I}_i^u) \wedge \big(zero(\mathcal{F}_i^v) \Leftrightarrow zero(\mathcal{F}_i^u)\big)\Big)\wedge \\ \bigwedge_{i,j\in\{1,\ldots,n\}}\big(leq(\mathcal{F}_i^u, \mathcal{F}_j^u) \Leftrightarrow leq(\mathcal{F}_i^v, \mathcal{F}_j^v)\big). \end{cases}$
The formula *equiv* encodes the equivalence relation of the clock valuations, i.e., the relation $\simeq$.

Let $t_1 = \lceil log_2(c)\rceil$, $t_2 = \lceil log_2(4n)\rceil$, $\boldsymbol{d} = \Theta_{\lceil log_2(4n)\rceil}(d)$ for $d \in \{0, \ldots, 2n-1\}$, $\boldsymbol{0} = \Theta_{\lceil log_2(4n)\rceil}(0)$, $\boldsymbol{2n} = \Theta_{\lceil log_2(4n)\rceil}(2n)$, $\boldsymbol{1} = \Theta_{\lceil log_2(c)\rceil}(1)$.

– $cleverSum(\boldsymbol{v}1, \boldsymbol{d}, \boldsymbol{v}2) := \begin{cases} \bigwedge_{i=1}^{n} \Big(\big(eq(sum_{t_2}(\mathcal{F}_i^{v1}, \boldsymbol{d}), \boldsymbol{2n}) \wedge eq(\mathcal{F}_i^{v2}, \boldsymbol{0}) \wedge \\ eq(sum_{t_1}(\mathcal{I}_i^{v1}, \boldsymbol{1}), \mathcal{I}_i^{v2})\big) \vee \big(le(sum_{t_2}(\mathcal{F}_i^{v1}, \boldsymbol{d}), \boldsymbol{2n}) \wedge \\ eq(\mathcal{I}_i^{v1}, \mathcal{I}_i^{v2}) \wedge eq(sum_{t_2}(\mathcal{F}_i^{v1}, \boldsymbol{d}), \mathcal{F}_i^{v2})\big) \Big) \end{cases}$

The formula $cleverSum$ encodes the addition of the bit-vector $\boldsymbol{d}$ to all vectors $\mathcal{F}_i^{v1}$ and $\mathcal{I}_i^{v1}$ representing the value of the clock $x_i \in \mathrm{X}'$, for $i = 1, \ldots, n$. The result of the addition is put in the vector $\boldsymbol{v}2$, i.e., $\boldsymbol{v}2$ contains either the value zone-equivalent to the value $\boldsymbol{v}1$ or its time successor.

– $TS((\boldsymbol{s}1, \boldsymbol{v}1), (\boldsymbol{s}2, \boldsymbol{v}2)) := \begin{cases} eq(\boldsymbol{s}1, \boldsymbol{s}2) \wedge \Big(\bigvee_{d=0}^{2n-1} cleversum(\boldsymbol{v}1, \boldsymbol{d}, \boldsymbol{v}2) \Big) \wedge \\ \neg equiv(\boldsymbol{v}1, \boldsymbol{v}2) \wedge \Big(\bigvee_{s \in S} \big(eq(\boldsymbol{s}2, s) \wedge Inv(s, \boldsymbol{v}2)\big) \Big). \end{cases}$

Now, we are ready to implement the formula T.

– $T((\boldsymbol{s}1, \boldsymbol{v}1), (\boldsymbol{s}2, \boldsymbol{v}2)) := AS((\boldsymbol{s}1, \boldsymbol{v}1), (\boldsymbol{s}2, \boldsymbol{v}2)) \vee TS((\boldsymbol{s}1, \boldsymbol{v}1), (\boldsymbol{s}2, \boldsymbol{v}2))$.

Next, we implement the formulas to be used in the definition of $[\varphi]_{M_k}$.

– $H((\boldsymbol{s}1, \boldsymbol{v}1), (\boldsymbol{s}2, \boldsymbol{v}2)) := eq(\boldsymbol{s}1, \boldsymbol{s}2) \wedge eq(\boldsymbol{v}1, \boldsymbol{v}2)$.
– Let $\mathcal{V}_S : \mathcal{PV} \to 2^S$ be a function returning the set of locations of which a proposition p is true, for each $p \in \mathcal{PV}$, defined as $\mathcal{V}_S(p) = \{s \in S \mid p \in \mathcal{V}(s)\}$. The formula encoding all the states labeled by the proposition $p_i \in \mathcal{PV}$ is defined as $p_i((\boldsymbol{s}, \boldsymbol{v})) := \bigvee_{s \in \mathcal{V}_S(p_i)} eq(\boldsymbol{s}, s)$.
– The formula encoding all the states labeled by the proposition $p_{x_i \in I} \in \mathcal{PV}_X$ is defined as $p_{x_i \in I}((\boldsymbol{s}, \boldsymbol{v})) := \bigwedge_{\xi_i \in inter(I, x_i)} Bool(\xi_i, \boldsymbol{v})$, where ξ_i denotes an inequality over $x_i \in \mathrm{X}'$.
– $p((\boldsymbol{s}, \boldsymbol{v}))$ iff $\begin{cases} p_i((\boldsymbol{s}, \boldsymbol{v})), & \text{if } p_i \in \mathcal{PV} \\ p_{x_i \in I}((\boldsymbol{s}, \boldsymbol{v})), & otherwise. \end{cases}$

Below, we implement the formula $dead$ encoding the deadlock.

– Let $add : \mathcal{SV}^r \times \{0,1\}^{\lceil log_2(4n) \rceil} \to \mathcal{SF}^r$ be a function, which returns a vector of propositional formulas such that each vector $\boldsymbol{u}_i$ (i.e., the vector consisting of $\mathcal{I}_i^u$ and $\mathcal{F}_i^u$) encodes the sum of the values represented by $\boldsymbol{d}$ and $\boldsymbol{v}_i$ (i.e., the vector consisting of $\mathcal{I}_i^v$ and $\mathcal{F}_i^v$) as the proper fractions, defined as follows. Let $d \in \{0, \ldots, 2n-1\}$. Then, $add(\boldsymbol{v}, \boldsymbol{d}) = \boldsymbol{u}$, where for all $i = 1, \ldots, n$.

$\mathcal{I}_i^u[j] := \begin{cases} \big(le(sum_{t_2}(\mathcal{F}_i^v, \boldsymbol{d}), \boldsymbol{2n}) \wedge \mathcal{I}_i^v[j]\big) \vee \big(eq(sum_{t_2}(\mathcal{F}_i^v, \boldsymbol{d}), \boldsymbol{2n}) \wedge \\ \big(sum_{t_1}(\mathcal{I}_i^v, \boldsymbol{1})\big)[j]\big) \vee \big(ge(sum_{t_2}(\mathcal{F}_i^v, \boldsymbol{d}), \boldsymbol{2n}) \wedge \big(sum_{t_1}(\mathcal{I}_i^v, \boldsymbol{1})\big)[j]\big) \end{cases}$

for all $j = 1, \ldots, \lceil log_2(c) \rceil$,

$\mathcal{F}_i^u[j] := \begin{cases} \big(le(sum_{t_2}(\mathcal{F}_i^v, \boldsymbol{d}), \boldsymbol{2n}) \wedge (sum_{t_2}(\mathcal{F}_i^v, \boldsymbol{d}))[j]\big) \vee \\ \big(eq(sum_{t_2}(\mathcal{F}_i^v, \boldsymbol{d}), \boldsymbol{2n}) \wedge (zero(\mathcal{F}_i^v))[j]\big) \vee \\ \big(ge(sum_{t_2}(\mathcal{F}_i^v, \boldsymbol{d}), \boldsymbol{2n}) \wedge (minus_{t_2}(sum_{t_2}(\mathcal{F}_i^v, \boldsymbol{d}), \boldsymbol{2n}))[j]\big) \end{cases}$

for all $j = 1, \ldots, \lceil log_2(4n) \rceil$.

$dead((\boldsymbol{s}, \boldsymbol{v})) := \begin{cases} \neg \Big[\bigvee_{s \in S} \Big(eq(\boldsymbol{s}, s) \ \wedge \ \bigvee_{d=1}^{2n-1} \big(Inv(s, add(\boldsymbol{v}, \boldsymbol{d})) \wedge \\ cleversum(\boldsymbol{v}, \boldsymbol{d}, add(\boldsymbol{v}, \boldsymbol{d})) \wedge \neg equiv(add(\boldsymbol{v}, \boldsymbol{d}), \boldsymbol{v})\big) \Big) \vee \\ \bigvee_{e \in E} \big(eq(\boldsymbol{s}, source(e)) \wedge Guard(e, \boldsymbol{v}) \wedge Inv(source(e), \boldsymbol{v})\big) \Big] \end{cases}$