Checking MTL Properties of Discrete Timed Automata via Bounded Model Checking[⋆]

Extended Abstract

Bożena Woźna-Szcześniak and Andrzej Zbrzezny

IMCS, Jan Długosz University.
Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland.
{b.wozna,a.zbrzezny}@ajd.czest.pl

Abstract. We investigate a SAT-based bounded model checking (BMC) method for MTL (metric temporal logic) that is interpreted over linear discrete infinite time models generated by discrete timed automata. In particular, we translate the existential model checking problem for MTL to the existential model checking problem for a variant of linear temporal logic (called HLTL), and we provide a SAT-based BMC technique for HLTL. We show how to implement the BMC technique for HLTL and discrete timed automata, and as a case study we apply the technique in the analysis of TGPP, a Timed Generic Pipeline Paradigm modelled by a network of discrete timed automata.

1 Introduction

Nowadays the interest in model checking [5] is focused not only on standard concurrent systems, but also on soft real-time systems, i.e., systems the goal of which is to ensure a certain subset of deadlines in order to optimize some application specific criteria. A number of formalisms, which use a discrete time domain, have been proposed in the literature to model the behaviour of these systems, e.g. discrete timed automata [2] and discrete timed Petri nets [7]. To express the requirements of the systems mostly standard temporal logics are used: *computation tree logic* (CTL) [4], *the soft real-time CTL* (RTCTL) [6], *linear temporal logic* (LTL) [13], and *metric temporal logic* (MTL) [8, 10, 14].

Bounded model checking (BMC) [3, 11, 12] is a symbolic verification method that uses only a portion of the considered model that is truncated up to some specific depth. It exploits the observation that we can infer some properties of the model using only its fragments. This approach can be combined with symbolic techniques based on decision diagrams or with techniques which involve translation of the verification problem either to the boolean satisfiability problem (SAT) or to the satisfiability modulo theories (SMT) problem.

The original contributions of the paper are as follows. First, we define a SAT-based BMC for soft real-time systems, which are modelled by discrete

[⋆] Partly supported by National Science Center under the grant No. 2011/01/B/ST6/05317.

timed automata, and for properties expressible in MTL. Next, we report on the implementation of the proposed BMC method as a new module of VerICS [9]. Finally, we evaluate the BMC method experimentally by means of a *timed generic pipeline paradigm* (TGPP), which we model by a network of discrete timed automata.

The rest of the paper is structured as follows. In Section 2 we brief the basic notion used through the paper. In Section 3 we define the BMC method for HLTL. In Section 4 we discuss our experimental results. In Section 5 we conclude the paper.

2 Preliminaries

We assume familiarity with the notion of *discrete timed automaton* (DTA) and their semantics in terms of the Kripke structure (called *model*). We refer the reader to the body of the paper [15] for details; note that a discrete timed automaton is basically a timed automaton with the restriction that clocks are positive integer variables. Further, we assume the following syntax of MTL. Let $p \in \mathcal{PV}$, and I be an interval in $\mathbb{N} = \{0, 1, 2, \ldots\}$ of the form: $[a, b)$ or $[a, \infty)$, for $a, b \in \mathbb{N}$ and $a \neq b$; note that the remaining forms of intervals can be defined by means of $[a, b)$ and $[a, \infty)$. The MTL formulae in the negation normal form are defined by the following grammar:

$$\alpha := \top \mid \bot \mid p \mid \neg p \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \mathrm{U}_I \alpha \mid \alpha \mathrm{R}_I \alpha$$

We refer the reader to the body of the paper [15] for the semantics of MTL; note that to get the discrete time semantics for MTL from the dense time semantics for MITL, in all the definitions presented in [15] at page 5, it is enough to replace the set of positive real numbers with the set of positive integer numbers, and to drop the assumption about single intervals.

Determining whether an MTL formula φ is existentially (resp. universally) valid in a given model is called an *existential* (resp. *universal*) *model checking problem*.

In order to define a SAT-based BMC method for MTL, we first translate the existential model checking problem for MTL that is interpreted over the region graph (i.e., a standard finite model defined for (discrete) timed automata) to the existential model checking problem for HLTL that is also interpreted over the region graph. For the details on this translation and the semantics of the HLTL language we refer the reader to [15]; note that the single intervals do not affect this translation. Here we only provide the syntax of HLTL and the translation scheme.

Let φ be an MTL formula, n the number of intervals in φ, $p \in \mathcal{PV}$ a propositional variables, and $h = 0, \ldots, n-1$. The HLTL formulae in release positive normal form are given by the following grammar:

$$\alpha := \top \mid \bot \mid p \mid \neg p \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \mathrm{H}_h \alpha \mid \alpha \mathrm{U} \alpha \mid \alpha \mathrm{R} \alpha$$

where the symbols U and R denote the *until* and *release* modalities, respectively. The indexed symbol H_h denotes the *reset* modality representing the reset of the clock number h. In addition, we introduce some useful derived temporal modalities: $\mathrm{G}\alpha \overset{def}{=} \bot\mathrm{R}\alpha$ (*always*), $\mathrm{F}\alpha \overset{def}{=} \top\mathrm{U}\alpha$ (*eventually*).

Let φ be a MTL formula. We translate the formula φ inductively into the HLTL formula $\mathcal{H}(\varphi)$ in the following way:

$$\begin{aligned}
&\mathcal{H}(\top) = \top, \quad \mathcal{H}(\bot) = \bot, \quad \mathcal{H}(p) = p, \quad \mathcal{H}(\neg p) = \neg p, \text{ for } p \in \mathcal{PV},\\
&\mathcal{H}(\alpha \vee \beta) = \mathcal{H}(\alpha) \vee \mathcal{H}(\beta), \qquad \mathcal{H}(\alpha \wedge \beta) = \mathcal{H}(\alpha) \wedge \mathcal{H}(\beta),\\
&\mathcal{H}(\alpha \mathrm{U}_{I_h} \beta) = \mathrm{H}_h(\mathcal{H}(\alpha)\mathrm{U}(\mathcal{H}(\beta) \wedge p_{y_h \in I_h} \wedge (p_{nf} \vee \mathcal{H}(\alpha)))),\\
&\mathcal{H}(\alpha \mathrm{R}_{I_h} \beta)) = \mathrm{H}_h(\mathcal{H}(\alpha)\mathrm{R}(\neg p_{y_h \in I_h} \vee \mathcal{H}(\beta))).
\end{aligned}$$

Observe that the translation of literals as well as logical connectives is straightforward. The translation of the U_{I_h} operator ensures that: (1) the translation of β holds in the interval I – this is expressed by the requirement $\mathcal{H}(\beta) \wedge p_{y_h \in I_h}$; (2) the translation of α holds always before the translation of β; and (3) if the value of the clock y_h belong to the final zone, i.e. the values of all the clocks are bigger then some maximal value (in this case the proposition p_{nf} is not true), then the translation of $\mathcal{H}(\alpha)$ is taken into account as well. The translation of the R_{I_h} operator makes use of the fact $\alpha\mathrm{R}_{I_h}\beta = \beta\mathrm{U}_{I_h}\alpha \wedge \beta \vee \mathrm{G}_{I_h}\beta$.

This translation preserves the existential model checking problem, i.e., the existential model checking of an MTL formula φ over the discrete model can be reduced to the existential model checking of $\mathcal{H}(\varphi)$ over the region graph.

The next step in defining a SAT-based BMC method for MTL relies on introducing a discretisation scheme for the region graph (defined for a given discrete timed automaton) that will represent zones (i.e. sets of equivalent clock valuations) of the region graph by exactly one specially chosen representative, and proving that a discretised model based on this scheme preserves the validity of the HLTL formulae - the discretised model constitutes the base for an implementation of our BMC method. We do not report on this step here in detail, since it requires introducing the huge mathematical machinery, but in fact it can be done in a way similar to the one presented in [16]. However, this will be provided in the full version of the paper.

The final step in defining a SAT-based BMC method for MTL relies on defining the BMC method for HLTL. This is described in the next section.

3 Bounded model checking for HLTL

Bounded semantics of a logic in question with existential interpretation is always used as the theoretical basis for the SAT-based bounded model checking. In the paper we have decided not to provide this semantics and not to show its equivalence to the unbounded semantics. This will be presented in the full version of the paper. Here, we only focus on the core of the BMC method, i.e. on the translation to SAT.

Let $\mathcal{A}$ be a discrete timed automaton, φ an MTL formula, $\psi = \mathcal{H}(\varphi)$ the corresponding HLTL formula, $\mathcal{M}$ a discretized model for $\mathcal{A}_\varphi$ (this an extension of $\mathcal{A}$ – for the exact construction we refer to [15]), and $k \geq 0$ a bound. We propose a BMC method for HLTL, which is based on the BMC technique presented in [17]. More precisely, we construct a propositional formula

$$[\mathcal{M}, \psi]_k := [\mathcal{M}^{\psi,\iota}]_k \ \wedge \ [\psi]_{\mathcal{M},k} \tag{1}$$

that is satisfiable if and only if the underlying model $\mathcal{M}$ is a genuine model for ψ. The constructed Formula (1) is given to a satisfiability solving program (a SAT-solver), and if a satisfying assignment is found, that assignment is a witness for the checked property. If a witness cannot be found at a given depth, k, then the search is continued for larger k.

The definition of the formula $[\mathcal{M}, \psi]_k$ requires, among other, states of the model $\mathcal{M}$ to be encoded in a symbolic way. This encoding is possible, since the set of states of $\mathcal{M}$ is finite. In particular, we represent each state s by a vector $w = (\mathtt{w}_1, \ldots, \mathtt{w}_r)$ (called a *symbolic state*) of propositional variables (called *state variables*), whose length r depends on the number of locations and clocks in $\mathcal{A}_\varphi$. Further, we need to represent finite prefixes of paths in a symbolic way. We call this representation a *j-th symbolic k-path* $\boldsymbol{\pi}_j$ and define it as a pair $((w_{0,j}, \ldots, w_{k,j}), u_j)$, where $w_{i,j}$ are symbolic states for $0 \leq j < f_k(\psi)$ and $0 \leq i \leq k$, and u_j is a *symbolic number* for $0 \leq j < f_k(\psi)$. The *symbolic number* u_j is a vector $u_j = (\mathtt{u}_{1,j}, \ldots, \mathtt{u}_{t,j})$ of propositional variables (called *natural variables*), whose length t equals to $max(1, \lceil log_2(k+1) \rceil)$, and it is used to encode the looping conditions. Next, we need an auxiliary function $\widehat{f_k}$: HLTL $\rightarrow \mathbb{N}$ that gives a bound on the number of k-paths sufficient for validating a given HLTL formula. The function is defined as $\widehat{f_k}(\psi) = f_k(\psi) + 1$, where $f_k(\top) = f_k(\bot) = f_k(p) = f_k(\neg p) = 0$ for $p \in \mathcal{PV}$; $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$; $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$; $f_k(\mathrm{H}_h \alpha) = f_k(\alpha) + 1$; $f_k(\alpha \mathrm{U} \beta) = k \cdot f_k(\alpha) + f_k(\beta)$; $f_k(\alpha \mathrm{R} \beta) = (k+1) \cdot f_k(\beta) + f_k(\alpha)$.

The formula $[\mathcal{M}^{\psi,\iota}]_k$ – the 1st conjunct of Formula (1) – encodes $\widehat{f_k}(\psi)$-times unrolled transition relation, and it is defined in the following way:

$$[\mathcal{M}^{\psi,\iota}]_k := I_\iota(w_{0,0}) \wedge \bigvee_{j=1}^{\widehat{f_k}(\varphi)} H(w_{0,0}, w_{0,j}) \wedge \bigwedge_{j=1}^{\widehat{f_k}(\psi)} \bigwedge_{i=0}^{k-1} \mathcal{T}(w_{i,j}, w_{i+1,j}) \wedge \bigwedge_{j=0}^{\widehat{f_k}(\psi)} \bigvee_{l=0}^{k} B_l^{=}(u_j) \tag{2}$$

where $w_{i,j}$ are symbolic states, u_j is a symbolic number, $I_\iota(w_{0,0})$ and $\mathcal{B}_l^{=}(u_j)$ are formulae encoding the initial state, and the value l, respectively. $H(w, w')$ is a formula that encodes equality of two global states. The formula $\mathcal{T}(w_{i,j}, w_{i+1,j})$ is disjunction of three formulas: $T(w_{i,j}, w_{i+1,j})$, $TA(w_{i,j}, w_{i+1,j})$, and $A(w_{i,j}, w_{i+1,j})$ that encode respectively the time, time-action, and action successors of $\mathcal{M}$.

The formula $[\psi]_{\mathcal{M},k} := [\psi]_k^{[0,1,F_k(\psi)]}$ – the 2nd conjunct of Formula (1) – encodes the translation of a HLTL formula ψ along a k-path, whose number belongs to the set $F_k(\psi) = \{j \in \mathbb{N} \mid 1 \leq j \leq \widehat{f_k}(\psi)\}$. The main idea of this translation consists in translating every subformula α of ψ using only $f_k(\alpha)$ k-paths. More precisely, given a formula ψ and a set $F_k(\psi)$ of indices of k-paths, following [17], we divide the set $F_k(\psi)$ into subsets needed for translating the subformulae of

ψ. We assume that the reader is familiar with this division process, and here we only recall definitions of the functions we use in the definition of the formula $[\psi]_k^{[0,1,F_k(\psi)]}$.

First, we recall the relation $\prec$ that is defined on the power set of $\mathbb{N}$ as: $A \prec B$ iff for all natural numbers x and y, if $x \in A$ and $y \in B$, then $x < y$. Now, let $A \subset \mathbb{N}$ be a finite nonempty set, and $n, d \in \mathbb{N}$, where $d \leq |A|$. Then,

- $g_l(A, d)$ denotes the subset B of A such that $|B| = d$ and $B \prec A \setminus B$.
- $g_r(A, d)$ denotes the subset C of A such that $|C| = d$ and $A \setminus C \prec C$.
- $g_s(A)$ denotes the set $A \setminus \{min(A)\}$.
- if n divides $|A| - d$, then $hp(A, d, n)$ denotes the sequence $(B_0, \ldots, B_n)$ of subsets of A such that $\bigcup_{j=0}^{n} B_j = A$, $|B_0| = \ldots = |B_{n-1}|$, $|B_n| = d$, and $B_i \prec B_j$ for every $0 \leq i < j \leq n$.

Now let $h_k^{\mathrm{U}}(A, d) \overset{df}{=} hp(A, d, k)$ and $h_k^{\mathrm{R}}(A, d) \overset{df}{=} hp(A, d, k+1)$. Note that if $h_k^{\mathrm{U}}(A, d) = (B_0, \ldots, B_k)$, then $h_k^{\mathrm{U}}(A, d)(j)$ denotes the set B_j, for every $0 \leq j \leq k$. Similarly, if $h_k^{\mathrm{R}}(A, d) = (B_0, \ldots, B_{k+1})$, then $h_k^{\mathrm{R}}(A, d)(j)$ denotes the set B_j, for every $0 \leq j \leq k+1$. Further, the function g_s is used in the translation of subformulae of the form $\mathrm{H}_h\alpha$, if a set A is used to translate this formula, then the path of the number $min(A)$ is used to translate the operator H_h and the set $g_s(A)$ is used to translate the subformula α. For more details on the remaining functions we refer to [17].

Now we are ready to define the formula $[\psi]_k^{[0,1,F_k(\psi)]}$. Let ψ be a HLTL formula, and $k \geq 0$ a bound. We can define inductively the translation $[\psi]_k^{[m,n,A]}$ of ψ along the n-th symbolic k−path $\boldsymbol{\pi}_n$ ($n \in F_k(\psi)$) with starting point m by using the set A as shown below. Let $cl(w_{m,n}, h)$ denote the fragment of the symbolic state $w_{m,n}$ that encodes the h-th clock from the set $\mathbb{Y}$, $n' = min(A)$, $h_k^{\mathrm{U}} = h_k^{\mathrm{U}}(g_s(A), f_k(\beta))$, and $h_k^{\mathrm{R}} = h_k^{\mathrm{R}}(g_s(A),\ f_k(\alpha))$. Then,

$$
\begin{aligned}
&[\top]_k^{[m,n,A]} := \top,\ [\bot]_k^{[m,n,A]} := \bot,\ [p]_k^{[m,n,A]} := p(w_{m,n}),\ [\neg p]_k^{[m,n,A]} := \neg p(w_{m,n}),\\
&[\alpha \wedge \beta]_k^{[m,n,A]} := [\alpha]_k^{[m,n,g_l(A,f_k(\alpha))]} \wedge [\beta]_k^{[m,n,g_r(A,f_k(\beta))]},\\
&[\alpha \vee \beta]_k^{[m,n,A]} := [\alpha]_k^{[m,n,g_l(A,f_k(\alpha))]} \vee [\beta]_k^{[m,n,g_l(A,f_k(\beta))]},\\
&[\mathrm{H}_h(\alpha \mathrm{U} \beta)]_k^{[m,n,A]} := \textstyle\bigwedge_{j=0}^{m-1} H(w_{j,n}, w_{j,n'}) \wedge H_{h=0}(w_{m,n}, w_{m,n'}) \wedge \bigwedge_{j=m+1}^{k} H_{\neq h}\\
&\quad (w_{j,n}, w_{j,n'}) \wedge \big(\textstyle\bigvee_{j=m}^{k}([\beta]_k^{[j,n',h_k^{\mathrm{U}}(k)]} \wedge \bigwedge_{i=m}^{j-1}[\alpha]_k^{[i,n',h_k^{\mathrm{U}}(i)]})\big) \vee\\
&\quad \textstyle\bigwedge_{j=m+1}^{k} H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge H_{h=0}(w_{m,n}, w_{m,n'}) \wedge\\
&\quad \big(\textstyle\bigvee_{l=0}^{m-1}(\mathcal{L}_k^l(\boldsymbol{\pi}_{n'}) \wedge \bigwedge_{j=0}^{l-1} H(w_{j,n}, w_{j,n'}) \wedge H(w_{l,n'}, w_{k,n'}) \wedge\\
&\quad \textstyle\bigwedge_{j=l+1}^{m-1} H_{\neq h}(w_{j,n}, w_{j,n'}))\big) \wedge \big(\bigvee_{j=0}^{m-1}(\mathcal{B}_j^{>}(u_{n'}) \wedge [\beta]_k^{[j,n',h_k^{\mathrm{U}}(k)]}\\
&\quad \wedge \textstyle\bigwedge_{i=0}^{j-1}(\mathcal{B}_i^{>}(u_{n'}) \rightarrow [\alpha]_k^{[i,n',h_k^{\mathrm{U}}(i)]}))\big) \wedge \bigwedge_{i=m}^{k}[\alpha]_k^{[i,n',h_k^{\mathrm{U}}(i)]},\\
&[\mathrm{H}_h(\alpha \mathrm{R} \beta)]_k^{[m,n,\,A]} := \textstyle\bigwedge_{j=0}^{m-1} H(w_{j,n}, w_{j,n'}) \wedge H_{h=0}(w_{m,n}, w_{m,n'}) \wedge \bigwedge_{j=m+1}^{k} H_{\neq h}\\
&\quad (w_{j,n}, w_{j,n'}) \wedge \big(\textstyle\bigvee_{j=m}^{k}([\alpha]_k^{[j,n',h_k^{\mathrm{R}}(k+1)]} \wedge \bigwedge_{i=m}^{j}[\beta]_k^{[i,n',h_k^{\mathrm{R}}(i)]})\big)\\
&\quad \vee \textstyle\bigwedge_{j=m+1}^{k} H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge H_{h=0}(w_{m,n}, w_{m,n'}) \wedge\\
&\quad \big(\textstyle\bigvee_{l=0}^{m-1}(\mathcal{L}_k^l(\boldsymbol{\pi}_{n'}) \wedge \bigwedge_{j=0}^{l-1} H(w_{j,n}, w_{j,n'}) \wedge H(w_{l,n'}, w_{k,n'}) \wedge\\
&\quad \textstyle\bigwedge_{j=l+1}^{m-1} H_{\neq h}(w_{j,n}, w_{j,n'}))\big) \wedge \big(\bigvee_{j=0}^{m}(\mathcal{B}_j^{>}(u_n') \wedge [\alpha]_k^{[j,n',h_k^{\mathrm{R}}(k+1)]}
\end{aligned}
$$

$$\begin{aligned}
&\wedge \bigwedge_{i=0}^{j-1}(\mathcal{B}_i^{>}(u_{n'}) \rightarrow [\beta]_k^{[i,n',h_k^{\mathrm{R}}(i)]}))) \wedge \bigwedge_{i=m}^{k}[\beta]_k^{[i,n',h_k^{\mathrm{R}}(i)]}\\
&\vee \bigwedge_{j=0}^{m-1} H(w_{j,n}, w_{j,n'}) \wedge H_{h=0}(w_{m,n}, w_{m,n'}) \wedge \bigwedge_{j=m+1}^{k}\\
&H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge \bigwedge_{j=m}^{k}[\beta]_k^{[j,n',h_k^{\mathrm{R}}(j)]} \wedge \mathcal{B}_{right(h)}^{\leq}(cl(w_{k,n'}, h))\\
&\vee \mathcal{B}_{right(h)}^{>}(cl(w_{k,n'}, h)) \wedge \bigwedge_{j=0}^{m-1} H(w_{j,n}, w_{j,n'}) \wedge H_{h=0}\\
&(w_{m,n}, w_{m,n'}) \wedge \bigwedge_{j=m+1}^{k} H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge \bigwedge_{j=m}^{k}[\beta]_k^{[j,n',h_k^{\mathrm{R}}(j)]}\\
&\wedge(\bigvee_{l=m}^{k-1}(\mathcal{L}_k^{l}(\boldsymbol{\pi}_{n'}))\vee \mathcal{B}_{right(h)}^{>}(cl(w_{k,n'}, h)) \wedge H_{h=0}(w_{m,n}, w_{m,n'})\\
&\wedge \bigwedge_{j=m+1}^{k} H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge \bigwedge_{j=m}^{k}[\beta]_k^{[j,n',h_k^{\mathrm{R}}(j)]} \wedge\\
&\big(\bigvee_{l=0}^{m-1}(\mathcal{L}_k^{l}(\boldsymbol{\pi}_{n'}) \wedge \bigwedge_{j=0}^{l-1} H(w_{j,n}, w_{j,n'}) \wedge H(w_{l,n'}, w_{k,n'}) \wedge\\
&\bigwedge_{j=l+1}^{m-1} H_{\neq h}(w_{j,n}, w_{j,n'}) \wedge \bigwedge_{j=l+1}^{m-1}[\beta]_k^{[j,n',h_k^{\mathrm{R}}(j)]})\big).
\end{aligned}$$

where $p(w)$ is a formula that encodes a set of states of M in which $p \in \mathcal{PV}$ holds; $H(w, w')$ is a formula that encodes equality of two global states; $H_{h=0}(w, w')$ is a formula that for two global states encodes the equality of their locations, the equality of values of the original clocks (i.e., clocks from $\mathbb{X}$), and the equality of values of the new clocks (i.e., clocks from $\mathbb{Y}$) but the value of clock y_h. For clock y_h the formula guarantees that its value in the 2nd global state is equal to zero; $H_{\neq h}(w, w')$ is a formula that for two global states encodes the equality of their locations, the equality of values of the original clocks, and the equality of the values of the new clocks with the potential exception of clock y_h. For clock y_h the formula guarantees that its value in the 2nd global state is greater than zero; $H_{\mathbb{X}}(w, w')$ is a formula that encodes equality of two global states on locations and values of the original clocks; $\mathcal{B}_j^{\sim}(\mathbf{v})$ is a formula that encodes that the value represented by the vector of propositional variables $\mathbf{v}$ is in arithmetic relation $\sim$ with the value j, where $\sim \in \{<, \leqslant, =, \geqslant, >\}$; $\mathcal{L}_k^{l}(\boldsymbol{\pi}_j) := \mathcal{B}_k^{>}(u_j) \wedge H_{\mathbb{X}}(w_{k,j}, w_{l,j})$.

The following theorem, whose proof will be provided in the full version of the paper, guarantees that the bounded model checking problem can be reduced to the SAT-problem.

Theorem 1. *Let $\mathcal{M}$ be a discrete abstract model, and ψ a HLTL formula. Then for every $k \in \mathbb{N}$, ψ is existentially valid in $\mathcal{M}$ with the bound k if, and only if, the propositional formula $[\mathcal{M}, \psi]_k$ is satisfiable.*

4 Experimental results

Our SAT-based BMC method for MTL, interpreted over the discrete time models, and discrete timed automata is, to our best knowledge, the first one formally presented in the literature, and moreover there is no any other model checking technique for the considered MTL language. Further, our implementation of the presented BMC method uses Reduced Boolean Circuits (RBC) [1] to represent the propositional formula $[\mathcal{M}, \psi]_k$. An RBC represents subformulae of $[\mathcal{M}, \psi]_k$ by fresh propositions such that each two identical subformulae correspond to the same proposition.

For the tests we have used a computer with Intel Core i3-2125 processor, 8 GB of RAM, and running Linux 2.6. We set the time limit to 900 seconds, and memory limit to 8GB, and we used the state of the art SAT-solver MiniSat 2. The specifications for the described benchmark are given in the universal form, for which we verify the corresponding counterexample formula, i.e., the formula which is negated and interpreted existentially.

To evaluate the performance of our SAT-based BMC algorithms for the verification of several properties expressed in MTL, we have analysed a Timed Generic Pipeline Paradigm (TGPP) discrete timed automata model shown in Figure 1. It consists of Producer producing data ($ProdReady$) or being inactive, Consumer receiving data ($ConsReady$) or being inactive, and a chain of n intermediate Nodes which can be ready for receiving data ($Node_iReady$), processing data ($Node_iProc$), or sending data ($Node_iSend$). The example can be scaled by adding intermediate nodes or by changing the length of intervals (i.e., the parameters a, b, c, d, e, f, g, h) that are used to adjust the time properties of Producer, Consumer, and of the intermediate Nodes.

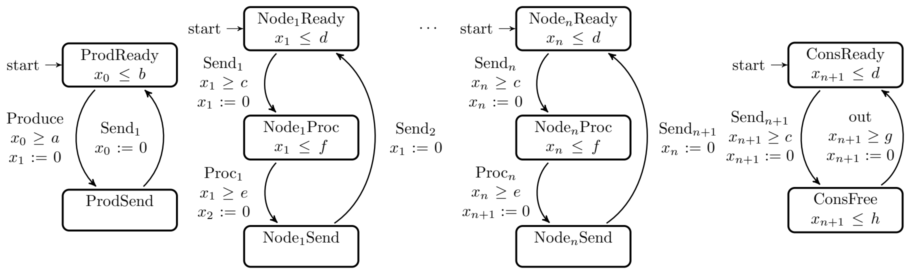
start
ProdReady $x_0 \leq b$
Produce $x_0 \geq a$ $x_1 := 0$
Send$_1$ $x_0 := 0$
ProdSend
start
Node$_1$Ready $x_1 \leq d$
Send$_1$ $x_1 \geq c$ $x_1 := 0$
Node$_1$Proc $x_1 \leq f$
Proc$_1$ $x_1 \geq e$ $x_2 := 0$
Node$_1$Send
Send$_2$ $x_1 := 0$
start
Node$_n$Ready $x_n \leq d$
Send$_n$ $x_n \geq c$ $x_n := 0$
Node$_n$Proc $x_n \leq f$
Proc$_n$ $x_n \geq e$ $x_{n+1} := 0$
Node$_n$Send
Send$_{n+1}$ $x_n := 0$
start
ConsReady $x_{n+1} \leq d$
Send$_{n+1}$ $x_{n+1} \geq c$ $x_{n+1} := 0$
out $x_{n+1} \geq g$ $x_{n+1} := 0$
ConsFree $x_{n+1} \leq h$

Fig. 1. A Generic Timed Pipeline Paradigm discrete timed automata model

We have tested the TGPP discrete timed automata model, scaled in the number of intermediate nodes and with all the intervals set to $[1, 3]$, on the following MTL formulae:

φ_1 $= \mathrm{G}_{[0,\infty)}(ProdSend \Rightarrow \mathrm{F}_{[2n+1,2n+2)}ConsFree)$, where n is the number of nodes. It expresses that each time Producer produces data, then Consumer receives this data in $2n + 1$ time units.

φ_2 $= \mathrm{G}_{[0,\infty)}(ProdSend \Rightarrow \mathrm{F}_{[2n+1,2n+2)}(ConsFree \wedge \mathrm{F}_{[1,2)}ConsReady))$, where n is the number of nodes. It expresses that each time Producer produces data, then Consumer receives this data in $2n + 1$ time units and one unit after that it will be ready to receive another data.

Since there is no model checker that supports the MTL properties of systems modelled by discrete timed automata, we were not able to compare results of the application of our method to a TGPP system with others.

We provide a preliminary evaluation of our method by means of the running time and the consumed memory. We have observed that for both formulae φ_1 and

φ_2, we managed to compute the results for 5 nodes in the time of 900 seconds. The exact data for the mentioned maximal number of nodes are the following:

φ_1: $k = 20$, $f_k(\varphi_1) = 3$, bmcT is 6.50, bmcM is 19.54, satT is 25.24, satM is 43.00, bmcT+satT is 31.74, max(bmcM,satM) is 43.00;

φ_2: $k = 20$, $f_k(\varphi_2) = 24$, bmcT is 89.96, bmcM is 163.40, satT is 610.23, satM is 310.00, bmcT+satT is 700.19, max(bmcM,satM) is 310.00;

where k is the bound, $f_k(\varphi)$ is the number of symbolic k-paths, *bmcT* is the encoding time, *bmcM* is memory use for encoding, *satT* is the satisfiability checking time, *satM* is memory use for the satisfiability checking.

The preliminary results are very promising and indicate that the method is worthy of further investigation for which purpose especially designed benchmarks will be developed.

5 Conclusions

We have introduced a SAT-based approach to bounded model checking of discrete timed automata and properties expressed in MTL with discrete semantics. The method is based on a translation of the existential model checking for MTL to the existential model checking for HLTL, and then on the translation of the existential model checking for HLTL to the propositional satisfiability problem. The two translations have been implemented and tested on the benchmark, which has been carefully selected in such a way as to reveal the advantages and disadvantages of the presented approaches.

References

1. P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT-solvers. In *Proceedings of the 6th International Conference 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. A theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. In *Highly Dependable Software*, volume 58 of *Advances in Computers*, pages 118–149. Academic Press, 2003.
4. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using Branching Time Temporal Logic. In *Proceedings of the Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.
5. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* The MIT Press, Cambridge, Massachusetts, 1999.
6. E. A. Emerson, A.K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Systems*, 4(4):331–352, December 1992.
7. M. Felder, D. Mandrioli, and A. Morzenti. Proving Properties of Real-Time Systems Through Logical Specifications and Petri Net Models. *IEEE Transaction on Software Engineering*, 20(2):127–141, 1994.

8. C. A. Furia and P. Spoletini. Tomorrow and all our yesterdays: MTL satisfiability over the integers. In *Proceedings of the Theoretical Aspects of Computing - ICTAC 2008*, volume 5160 of *LNCS*, pages 253–264. Springer-Verlag, 2008.
9. M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M.Szreter, B. Woźna, and A. Zbrzezny. VerICS 2007 - a model checker for knowledge and real-time. *Fundamenta Informaticae*, 85(1-4):313–328, 2008.
10. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
11. A. Lomuscio, W. Penczek, and B. Woźna. Bounded model checking for knowledge and real time. *Artificial Intelligence*, 171:1011–1038, 2007.
12. W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.
13. A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE International Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Presss, 1977.
14. M. Pradella, A. Morzenti, and P. San Pietro. A metric encoding for bounded model checking. In *Proceedings of the 2nd World Congress on Formal Methods (FM 2009)*, volume 5850 of *LNCS*, pages 741–756. Springer-Verlag, 2009.
15. B. Woźna-Szcześniak and A. Zbrzezny. A translation of the existential model checking problem from MITL to HLTL. *Fundamenta Informaticae*, 122(4):401–420, 2013.
16. A. Zbrzezny. A new discretization for timed automata. In *Proceedings of the International Workshop on Concurrency, Specification and Programming (CS&P'04)*, volume 170 of *Informatik-Berichte*, pages 178–189. Humboldt University, 2004.
17. A. Zbrzezny. A new translation from ECTL* to SAT. *Fundamenta Informaticae*, 120(3–4):377–397, 2012.