Checking MTL Properties of Timed Automata with Dense Time using Satisfiability Modulo Theories (Extended abstract)

Agnieszka M. Zbrzezny[1] and Andrzej Zbrzezny[2]

[1] Faculty of Mathematics and Computer Science, University of Warmia and Mazury in Olsztyn, Poland, agnieszka.zbrzezny@matman.uwm.edu.pl
[2] Institute of Mathematics and Computer Science, Jan Długosz University in Częstochowa, Poland. a.zbrzezny@ujd.edu.pl

Abstract. We investigate a new SMT-based bounded model checking (BMC) method for the existential part of Metric Temporal Logic (MTL). The MTL logic is interpreted over linear dense infinite time models generated by timed automata with dense time. We implemented the new SMT-based bounded model checking technique for MTL and as a case study we applied the technique to the analysis of the Timed Generic Pipeline Paradigm and Timed Train Controller System, both modelled by a network of timed automata.

1 Introduction

Timed automata [2] are very convenient for modelling and reasoning about real-time systems: they combine a powerful formalism with advanced expressiveness and efficient algorithmic and tool support. The timed automata formalism is now applied to the analysis of timing analysis of software and asynchronous circuits [12] and real-time control programs [13]. Timed automata technology was also used to analysis of numerous real-time communication protocols [20].

Timed automata (TA) are adequate to represent systems, but not that much for representing properties of systems[3]. Temporal logics are a well known framework in the field of specification and verification of computer systems [18]. Linear Temporal Logic (LTL) allows to express properties about each individual execution of a system, such as the fact that any occurrence of a problem eventually triggers the alarm. Real-time constraints have naturally been added to temporal logics [16, 1] at the beginning of the 90s. The resulting logics allow to express e.g. that any occurrence of a problem in a system will trigger the alarm within at most 5 time units.

Metric Temporal Logic (MTL) [16] was discussed in the literature e.g on the verification of real-time systems [14, 11, 22, 19, 7]. MTL extends the until and globally

[3] Assume that one TA represents a system and the second TA represents a property of the system. We would like to check if TA representing the system satisfies the property. This problem is unfortunately undecidable for timed automata.

operators of classical temporal logic with an interval which specifies the time interval within which the formula must be satisfied. The MTL logic has traditionally been interpreted in two ways: the *pointwise* and the *continuous* semantics. In this paper we are focused on the pointwise version. The temporal assertions are interpreted only at time points where an action happens in the observed timed behaviour of a system.

Bounded model checking [8, 9, 17] (BMC) is one of the symbolic model checking technique designed for finding witnesses for existential properties or counterexamples for universal properties. Its main idea is to consider a model reduced to a specific depth. The method works by mapping a bounded model checking problem to the Boolean satisfiability problem (SAT) or to satisfiability modulo theories problem (SMT).

The satisfiability and model checking problems for MTL are undecidable over the interval-based semantics [5]. This has led to consider various restrictions on MTL to recover decidability [21, 22, 3]. The main steps of our new method for MTL and timed automata with dense time can be described as follows:

1. the timed model (infinite) is reduced to a finite model,
2. the MTL formula is translated to $\mathrm{LTL_q}$ formula [24, 25],
3. since the interval modalities in MTL appear as literals in $\mathrm{LTL_q}$ formula, the existential properties are reduced to a satisfiability modulo theories problem (SMT).

We evaluate the new BMC method experimentally by means of a timed generic pipeline paradigm (TGPP) and timed train controller system, which are modelled by a network of timed automata with dense time.

2 Timed Automata

The model of timed automata has been defined in the 90s by Alur and Dill as a model for representing systems with real-time constraints. A timed automaton is basically a finite automaton which manipulates finitely many variables called clocks.

We assume a finite set $\mathcal{X} = \{x_0, \ldots, x_{n-1}\}$ of variables, called *clocks*. A *clock valuation* is a total function $v : \mathcal{X} \mapsto \mathbb{R}$ that assigns to each clock x a non-negative real value $v(x)$. The set of all the clock valuations is denoted by $\mathbb{R}^{|\mathcal{X}|}$. For $X \subseteq \mathcal{X}$, the valuation $v' = v[X := 0]$ is defined as: $\forall x \in X,\ v'(x) = 0$ and $\forall x \in \mathcal{X} \setminus X,\ v'(x) = v(x)$. For $\delta \in \mathbb{R}$, $v + \delta$ denotes the valuation v'' such that $\forall x \in \mathcal{X}, v''(x) = v(x) + \delta$. Let $x \in \mathcal{X}$, $c \in \mathbb{N}$, and $\sim \in \{<, \leqslant, =, \geqslant, >\}$. A *guard* over $\mathcal{X}$ is a finite conjunction of expressions of the form $x \sim c$. We denote by $\mathcal{C}(\mathcal{X})$ the set of guards over $\mathcal{X}$. A clock valuation v satisfies a clock guard $\mathfrak{cc}$, written as $v \models \mathfrak{cc}$, iff $\mathfrak{cc}$ evaluates to true using the clock values given by the valuation v.

Definition 1. *A* timed automaton *is a tuple* $\mathcal{A} = (Act, Loc, \ell^0, T, \mathcal{X}, Inv, \mathcal{AP}, V)$*, where*

- Act *is a finite set of actions,*
- Loc *is a finite set of locations,*
- $\ell^0 \in Loc$ *is an initial location,*
- $T \subseteq Loc \times Act \times \mathcal{C}(\mathcal{X}) \times 2^{\mathcal{X}} \times Loc$ *is a transition relation,*
- $\mathcal{X}$ *is a finite set of clocks,*
- $Inv : Loc \mapsto \mathcal{C}(\mathcal{X})$ *is a state invariant function,*

– *$\mathcal{AP}$ is a set of atomic proposition, and*
– *$V : Loc \mapsto 2^{\mathcal{AP}}$ is a valuation function assigning to each location a set of atomic propositions true in this location.*

Each element $t \in T$ is denoted by $\ell \overset{\sigma,\mathfrak{cc},X}{\longrightarrow} \ell'$, and it represents a transition from location ℓ to location ℓ' on the input action σ. $X \subseteq \mathcal{X}$ is the set of the clocks to be reset with this transition, and $\mathfrak{cc} \in \mathcal{C}(\mathcal{X})$ is the enabling condition for t.

2.1 Concrete model

The semantics of the timed automaton is defined by associating a transition system with it, which we call a *concrete model*.

Definition 2. *Let $\mathcal{A} = (Act, Loc, \ell^0, T, \mathcal{X}, Inv, \mathcal{AP}, V)$ a timed automaton, and v^0 a clock valuation such that $\forall x \in \mathcal{X}$, $v^0(x) = 0$.*

A concrete model *for $\mathcal{A}$ is a tuple $\mathcal{M}_{\mathcal{A}} = (Q, q^0, \longrightarrow, \mathcal{V})$, where*

– *$Q = Loc \times \mathbb{R}^n$ s a set of the concrete states,*
– *$q^0 = (\ell^0, v^0)$ is the initial state,*
– *$\longrightarrow \subseteq Q \times Q$ is a total binary relation on Q defined by action and time transitions as follows. For $a \in Act$ and $\delta \in \mathbb{R}_+$,*
  1. *Action transition: : $(\ell, v) \xrightarrow{a} (\ell', v')$ iff there is a transition $\ell \overset{a,\mathfrak{cc},X}{\longrightarrow} \ell' \in T$ such that $v \models \mathfrak{cc} \wedge Inv(\ell)$ and $v' = v[X := 0]$ and $v' \models Inv(\ell')$,*
  2. *Time transition: $(\ell, v) \xrightarrow{\delta} (\ell, v + \delta)$ iff $v \models Inv(\ell)$ and $v + \delta \models Inv(\ell)$.*
– *$\mathcal{V} : Q \mapsto 2^{\mathcal{AP}}$ is a valuation function such that $\mathcal{V}((\ell, v)) = V(\ell)$ for each state $(\ell, v) \in Q$.*

A *run* ρ in a concrete model for the timed automata $\mathcal{A}$ is a infinite sequence of concrete states $q_0 \overset{\delta_0,a_0}{\longrightarrow} q_1 \overset{\delta_1,a_1}{\longrightarrow} q_2 \overset{\delta_2,a_2}{\longrightarrow} \dots$ such that $q_i \in Q$, $a_i \in Act$ and $\delta_i \in \mathbb{R}_+$, for all $i \in \mathbb{N}$. An assumption that $\delta_i \in \mathbb{R}_+$ implies that runs are strongly monotonic. This is because the definition of the run does not permit two consecutive actions to be performed one after the other, i.e., between each two actions some time must pass.

3 MTL logic

MTL logic [16] (metric LTL) is a timed temporal logic with interval operators. It has received much attention in the literature on the verification of real-time systems. MTL can express many time constraints. For example we can express a system property: if a system is in the state q, than it will be in the state q' exactly 3 time units later.

Syntax Let $p \in \mathcal{AP}$ be a propositional variable and I be an interval in $\mathbb{N}$ of the form $[a, b)$ or $[a, \infty)$, where $a, b \in \mathbb{N}$ and $a < b$. The MTL logic in release positive normal form is defined in the following way:

$$\alpha := \mathbf{true} \mid \mathbf{false} \mid p \mid \neg p \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \mathbf{U}_{\mathrm{I}} \alpha \mid \mathbf{G}_{\mathrm{I}} \alpha.$$

Intuitively, $\mathbf{U}_{\mathrm{I}}$ and $\mathbf{G}_{\mathrm{I}}$ are the operators for, respectively, *bounded until* and *bounded always* and they are ridden as, respectively, „until in the interval I" and „always in the interval I". The operators $\mathbf{F}_{\mathrm{I}}$ and $\mathbf{R}_{\mathrm{I}}$ are defined in the standard way:

$$\mathbf{F}_{\mathrm{I}}\alpha \stackrel{df}{=} \mathbf{true}\,\mathbf{U}_{\mathrm{I}}\,\alpha \qquad\qquad \alpha\mathbf{R}_{\mathrm{I}}\beta \stackrel{df}{=} \mathbf{G}_{\mathrm{I}}\beta \vee \beta\,\mathbf{U}_{\mathrm{I}}(\alpha \vee \beta).$$

Semantics Over dense time the logic has traditionally been interpreted in either of two ways which have come to be known as the "pointwise" and the "continuous" semantics [10]. In the pointwise approach temporal assertions are interpreted only at time points where action happens in the observed timed behaviour of a system. In the continuous one is allowed to assert formulae at arbitrary time points between actions as well.

In the presented method we use the pointwise semantics.

Let $\mathcal{A}$ be a time automata, $\mathcal{M}_{\mathcal{A}} = (Q, q^0, \longrightarrow, \mathcal{V})$ be a concrete model for the timed automata $\mathcal{A}$, $\rho : q_0 \xrightarrow{\delta_0} q'_0 \xrightarrow{a_0} q_1 \xrightarrow{\delta_1} q'_1 \xrightarrow{a_1} q_2 \xrightarrow{\delta_2} q'_2 \xrightarrow{a_2} \ldots$ be a run in $\mathcal{A}$, which can be written in the shorter way: $\rho : q_0 \xrightarrow{\delta_0,a_0} q_1 \xrightarrow{\delta_1,a_1} q_2 \xrightarrow{\delta_2,a_2} \ldots$. Each of the runs determines the path λ_ρ: $q_0, q'_0, q_1, q'_1, q_2, q'_2 \ldots$ in an unambiguous way. Given a run ρ one can define a function $\zeta_\rho : \mathbb{N} \mapsto \mathbb{R}_+$ such that, for all the position $j \geqslant 0$, $\zeta_\rho(j)$ is a sum of all the time transitions along the run ρ till the position j. For all $j \geqslant m$, the function $\zeta_\rho(j)$ returns the value of the global time (in [10] called „*duration*").

To make the definition more clear we use a notation $(\lambda_\rho, m) \models_{\mathrm{EMTL}} \varphi$ instead of $\mathcal{M}, (\lambda_\rho, m) \models_{\mathrm{EMTL}} \varphi$, for each MTL formula φ.

Definition 3. *Let α and β be* MTL *formulae. The satisfaction relation $\models_{\mathrm{EMTL}}$, which indicates truth of an* MTL *formula in the concrete model $\mathcal{M}_{\mathcal{A}}$ along a path λ_ρ starting at position $m \in \mathbb{N}$, is defined inductively as follows:*

$(\lambda_\rho, m) \models_{\mathrm{EMTL}} \mathbf{true}$,
$(\lambda_\rho, m) \not\models_{\mathrm{EMTL}} \mathbf{false}$,
$(\lambda_\rho, m) \models_{\mathrm{EMTL}} p$ *iff* $p \in \mathcal{V}(\rho(m))$,
$(\lambda_\rho, m) \models_{\mathrm{EMTL}} \neg p$ *iff* $p \notin \mathcal{V}(\rho(m))$,
$(\lambda_\rho, m) \models_{\mathrm{EMTL}} \alpha \wedge \beta$ *iff* $(\lambda_\rho, m) \models_{\mathrm{EMTL}} \alpha$ *and* $(\lambda_\rho, m) \models_{\mathrm{EMTL}} \beta$,
$(\lambda_\rho, m) \models_{\mathrm{EMTL}} \alpha \vee \beta$ *iff* $(\lambda_\rho, m) \models_{\mathrm{EMTL}} \alpha$ *or* $(\lambda_\rho, m) \models_{\mathrm{EMTL}} \beta$,
$(\lambda_\rho, m) \models_{\mathrm{EMTL}} \alpha \mathbf{U}_{\mathrm{I}} \beta$ *iff* $(\exists j \geq m)(\zeta_\rho(j) - \zeta_\rho(m) \in \mathrm{I}$ *and* $(\lambda_\rho, m + j) \models_{\mathrm{EMTL}} \beta$ *and* $(\forall m \leqslant j' < j)(\lambda_\rho, m + j') \models_{\mathrm{EMTL}} \alpha)$,
$(\lambda_\rho, m) \models_{\mathrm{EMTL}} \mathbf{G}_{\mathrm{I}} \beta$ *iff* $(\forall j \geq m)(\zeta_\rho(j) - \zeta_\rho(m) \in \mathrm{I}$ *implies* $(\lambda_\rho, m + j) \models_{\mathrm{EMTL}} \beta)$.

As $(\lambda_\rho, 0) = \lambda_\rho$, we shall write $\mathcal{M}_{\mathcal{A}}, \lambda_\rho \models_{\mathrm{EMTL}} \varphi$ for $\mathcal{M}_{\mathcal{A}}, (\lambda_\rho, 0) \models_{\mathrm{EMTL}} \varphi$. An MTL formula φ is *existentially valid* in the model $\mathcal{M}_{\mathcal{A}}$, denoted $\mathcal{M}_{\mathcal{A}} \models_{\mathrm{EMTL}} \mathbf{E}\varphi$, if, and only if $\mathcal{M}_{\mathcal{A}}, \lambda_\rho \models_{\mathrm{EMTL}} \varphi$ for some path λ_ρ starting in the initial state of $\mathcal{M}_{\mathcal{A}}$. Determining whether an MTL formula φ is existentially valid in a given model is called an *existential model checking problem*.

4 SMT-based bounded model checking

The verification method presented in this paper is based on the translation of MTL formulae to $\mathrm{LTL}_{\mathrm{q}}$ formulae defined in [24, 25]. 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 [24]. However, this will be provided in the full version of the paper. We refer the reader to the chapter 4 of the thesis [24] for details.

The SMT-based bounded model checking can be described in the following steps:

1. Since the concrete model is infinite we have to abstract the model to be able to applied bounded model checking technique. It is a well-known technique [15, 4].
2. We use the EMTL semantics in the abstract model [24] and translate the EMTL formula into an $\mathrm{LTL_q}$ formula.
3. The next step is standard. It consists of a translation of the transition relation in the depth k in the abstract model and the $\mathrm{LTL_q}$ formula into satisfiability modulo theories problem. The difference between the BMC method for classic LTL and our method lies in encoding of a finite prefix $(\zeta_{\widetilde{\pi}_l}(0), \zeta_{\widetilde{\pi}_l}(1), \ldots)$.
4. After the translation to SMT, the SMT-solver checks the satisfiability of the $\mathrm{LTL_q}$ formula in the abstract model.

5 Experimental results

In this section we experimentally evaluate the performance of our new translation. We have conducted the experiments using the slightly modified timed generic pipeline paradigm (TGPP) and timed train controller system (TTCS) [23].

5.1 Timed Generic Pipeline Paradigm

The Timed Generic Pipeline Paradigm (TGPP) timed automata model shown in Figure 1 consists of a Producer producing data within the certain time interval ($[a, b]$) or being inactive, a Consumer receiving data within the certain time interval ($[c, d]$) or being inactive within the certain time interval ($[g, h]$), and a chain of n intermediate Nodes which can be ready for receiving data within the certain time interval ($[c, d]$), processing data within the certain time interval ($[e, f]$) or sending data. We assume that $a = c = e = g = 1$ and $b = d = f = h = 2 \cdot n + 2$, where n represents a number of nodes in the TGPP.

To compare our experimental results with [23], we have tested the TGPP timed automata model, scaled in the number of intermediate nodes on the following MTL formulae that existentially hold in the model of TGPP (n is the number of nodes):

- $\varphi_1 = \mathbf{F}_{[0,2\cdot n+3)}(ConsReceived)$. It expresses that Consumer receives the data in at most $2 \cdot n + 3$ time units.
- $\varphi_2 = \mathbf{G}_{[0,2\cdot n+2)}(ConsReady)$. It states that the Consumer is always forced to receive the data in $2 \cdot n + 2$ time units.
- $\varphi_3 = \mathbf{G}_{[0,\infty)}(ProdReady \vee ConsReady)$. It states that always either the Producer has sent the data or the Consumer has received the data.
- $\varphi_4 = \mathbf{F}_{[0,2\cdot n+3)}(\mathbf{G}_{[0,\infty)}(ProdSend \vee ConsReceived))$. It states that eventually in time less then $2 \cdot n + 3$ it is always the case that the Producer is ready to send the data or the Consumer has received the data.
- $\varphi_5 = \mathbf{G}_{[0,\infty)}(\mathbf{F}_{[0,2\cdot n+3)}(ConsReceived))$. It states that the Consumer infinitely often is receiving the data in time less then $2 \cdot n + 3$.

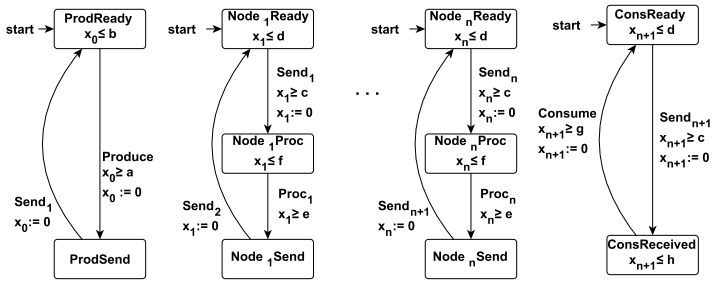

Fig. 1: The TGPP system.

5.2 Timed Train Controller System

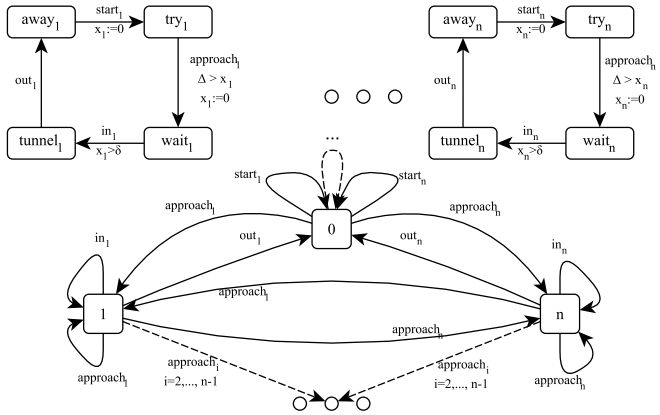

Fig. 2: The TTCS system.

The Timed Train Controller System (also known as Fischer's mutual exclusion protocol) consists of n (for $n \geq 2$) trains $T_1, \ldots, T_n$, each one using its own circular track for travelling in one direction and containing its own clock x_i, together with controller C used to coordinate the access of trains to the tunnel through which all trains have to pass at certain point. There is only one track in the tunnel, so trains arriving from each direction cannot use it in this same time. There are signals on both sides of the tunnel, which can be either red or green. All trains notify the controller when they request entry to the tunnel or when they leave the tunnel. The controller controls the colour of the displayed signal, and the behaviour of the scenario depends on the values δ and Δ ($\Delta \geq \delta$ makes it incorrect - the mutual exclusion does not hold).

Controller C has $n + 1$ states, denoting that all trains are away (state 0), and the numbers of trains, i.e., $1, \ldots, n$. Controller C is initially at state 0. The action $Start_i$

of train T_i denotes the passage from state away to the state where the train wishes to obtain access to the tunnel. This is allowed only if controller C is in state 0. Similarly, train T_i synchronises with controller C on action $approach_i$, which denotes setting C to state i, as well as out_i, which denotes setting C to state 0. Finally, action in_i denotes the entering of train T_i into the tunnel.

- $\psi_1 = \mathbf{F}_{[0,7)}(\bigvee_{i=1}^{n-1} \bigvee_{j=i+1}^{n} (tunnel_i \wedge tunnel_j))$. It states that the mutual exclusion property is violated, i.e. for some path some two trains are in the tunnel at the same time. The formula is existentially valid in the model if and only if $\Delta \geq \delta$.
- $\psi_2 = \mathbf{G}_{[0,\infty)}(\mathbf{F}_{[0,9)}(\bigvee_{i=1}^{n}(tunnel_i)))$. It states that always eventually some of the trains enters the tunnel. The formula is existentially valid in the model regardless of the values of Δ and δ.
- $\psi_3 = \mathbf{F}_{[0,\infty)}(\bigvee_{i=1}^{n}(tunnel_i))$. It states that eventually all the trains are in the tunnel. The formula is existentially valid in the model if $ndonlyif \Delta \geq \delta$.

5.3 Performance evaluation

We have performed our experimental results on a computer equipped with I7-5500U processor, 12 GB of RAM, and the operating system Linux with the kernel 5.2.9. Our SMT-based BMC algorithm is implemented as standalone programs written in the programming language C++. We used the state of the art SMT-solver Z3 (program version 8.4.5).

TGPP

The number of considered k-paths (f_k) for the translation is always equal to 1. The length of the witness for the formula φ_1 is equal to $4 \cdot (n + 1)$; for the formula φ_2 is equal to $8 \cdot n + 6$; for the formula φ_3 and is equal to $8 \cdot n + 6$; for the formula φ_4 is equal to $8 \cdot n + 15$; for the formula φ_5 is equal to $8 \cdot n + 6$.

The experimental results presented on the Fig. 3 show total time usage. We can observe that the SMT-based method is sensitive to scaling up the size of the benchmarks. For more complicated formulae time usage grows very fast with the size of the benchmark.

The maximum memory usage for all the formulae showed on the Fig. 4 is not very big. The biggest memory usage was for the formula φ_5 and it amounts 375 MB.

Fig. 5 shows time usage for bounded model checking algorithm (translation of the model and the formula to SMT format) and for the SMT-solver. We can observe that almost the whole time were consumed by the SMT-solver (the y axis has different scale on both figures). Generating a quantifier-free first-order formula in every case took less than 45 sec.

Fig. 6 shows memory usage for the BMC algorithm and SMT-solver. We can observe that for the formulae φ_1 and φ_3 the BMC algorithm did not use a lot of memory and the SMT-solver consumed almost the whole memory in this case. For other formulae the memory usage for BMC and SMT-solver is almost the same.

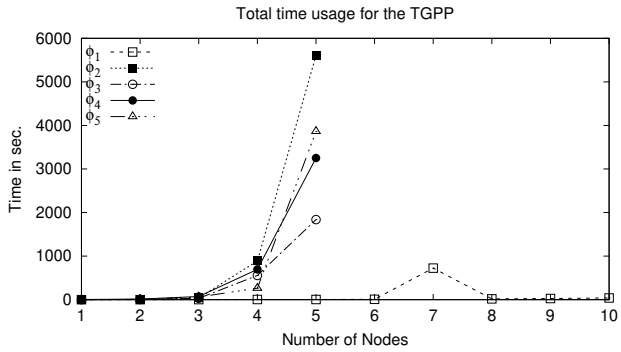

Fig. 3: Total time and usage for TGPP.

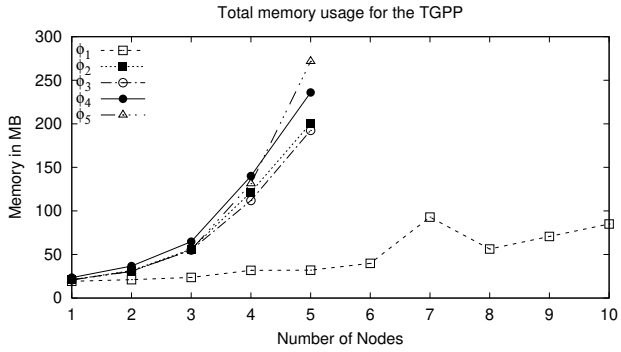

Fig. 4: Total memory usage for TGPP.

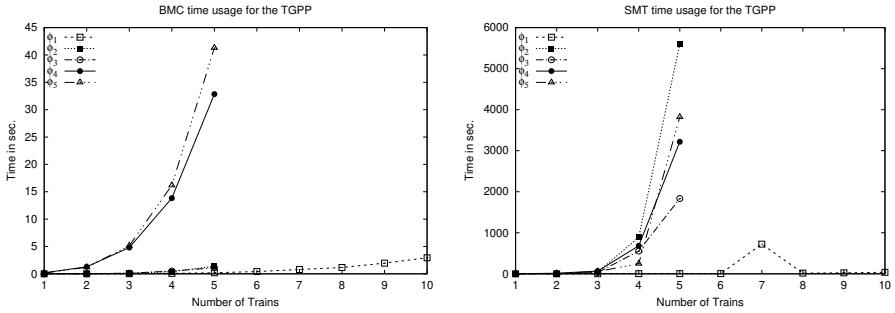

Fig. 5: BMC and SMT time usage for TGPP.

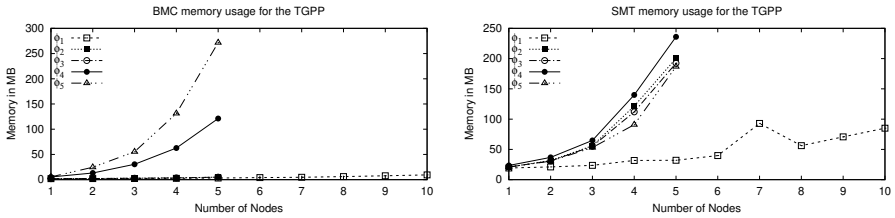

Fig. 6: BMC and SMT memory usage for TGPP.

TTCS

As we mentioned before, the number of considered k-paths (f_k) for the translation is always equal to 1. The length of the witness for the formula ψ_1 is equal to 12 and for the formula ψ_2 is equal to 7. For the formula ψ_3 the length of the witness is equal to 12 for two trains, 18 for three trains, 32 for four trains, and 38 for five trains.

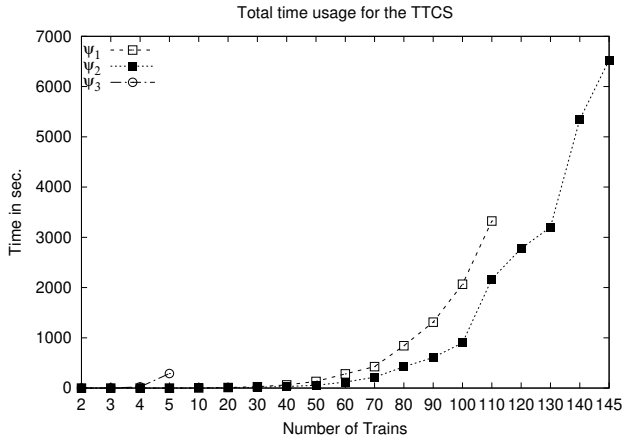

Fig. 7: Total time usage for TTCS.

Fig. 7 shows the time usage for the verification process for all the formulae. We can observe that the third formula could be verified only for the TTCS with 5 trains. We tried to verify it for the system with 6 trains but we gave up after 6 hours. Fig. 8 shows the memory usage for all the formulae.

Fig. 9 shows the time usage for BMC and SMT for all the formulae and TTCS. For the formula ψ_2 we can observe that BMC algorithm uses more time than the SMT-solver. However, the plot for this formula and SMT looks a bit weird. We expect that with the scaling up the benchmark the time usage would be bigger. For 120 and 130

trains we have the opposite situation. The reason of this situation is the heuristic of the SMT-solver. Z3 solver found the valuation faster. We think that the interesting future work will be finding the optimal SMT file for SMT-solvers [6].

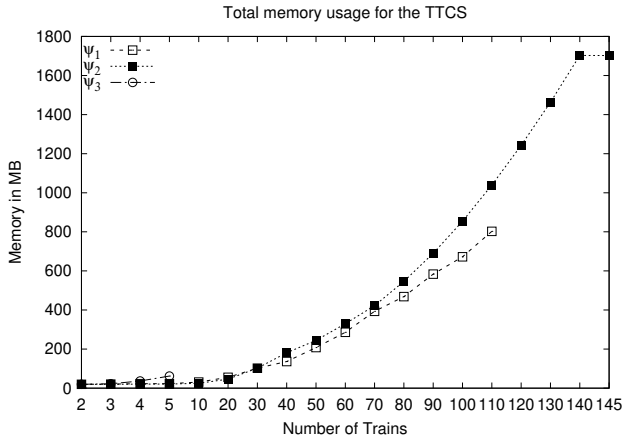

Fig. 8: Total memory usage for TTCS.

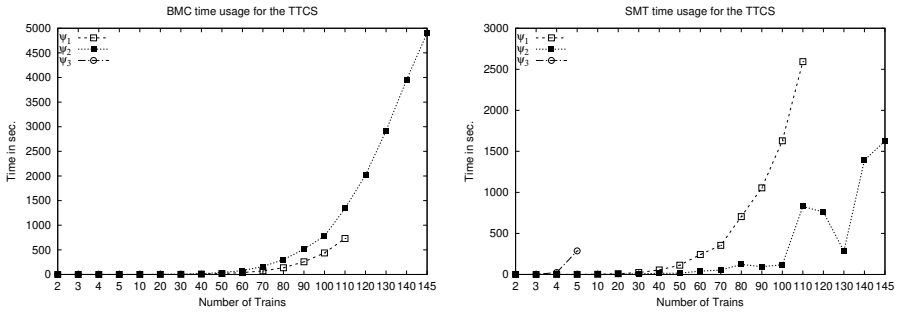

Fig. 9: SMT and BMC time usage for TTCS.

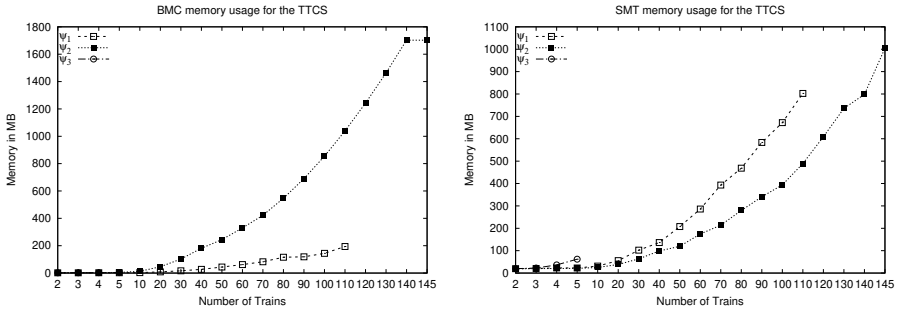

Fig. 10: SMT and BMC memory usage for TTCS.

6 Conclusions

We have implemented and experimentally evaluated an SMT-based BMC method for real-time systems, which are modelled by timed automata with dense time, and for properties expressible in MTL with the semantics over timed automata. We have experimentally evaluated the method. The method is based on a translation of the existential model checking for MTL to the existential model checking for $\mathrm{LTL_q}$, and then on the translation of the existential model checking for $\mathrm{LTL_q}$ to the satisfiability modulo theories problem.

In the future we would like to develop a corresponding SAT-based method. Developing the SMT-based method as a first method was a natural way to solve the problem. SMT encoding is easier to implement. However, the SAT encoding in many cases is more precise, what may give better experimental results.

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 and D. Dill. A theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.
4. Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999.
5. Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 390–401, 1990.
6. Mislav Balunovic, Pavol Bielik, and Martin T. Vechev. Learning to solve SMT formulas. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 10338–10349, 2018.
7. Stefano Baratella and Andrea Masini. A two-dimensional metric temporal logic. *CoRR*, abs/1903.05894, 2019.

8. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. of the 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.
9. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.
10. P. Bouyer. Model-checking timed temporal logics. *Electr. Notes Theor. Comput. Sci.*, 231:323–341, 2009.
11. Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. *Inf. Comput.*, 208(2):97–116, 2010.
12. Marius Bozga, Jianmin Hou, Oded Maler, and Sergio Yovine. Verification of asynchronous circuits using timed automata. *Electr. Notes Theor. Comput. Sci.*, 65(6):47–59, 2002.
13. Henning Dierks. Plc-automata: a new class of implementable real-time automata. *Theor. Comput. Sci.*, 253(1):61–93, 2001.
14. Deepak D'Souza and Pavithra Prabhakar. On the expressiveness of MTL in the pointwise and continuous semantics. *STTT*, 9(1):1–4, 2007.
15. 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.
16. Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
17. W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.
18. A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57, 1977.
19. Vladislav Ryzhikov, Przemyslaw Andrzej Walega, and Michael Zakharyaschev. Data complexity and rewritability of ontology-mediated queries in metric temporal logic under the event-based semantics. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1851–1857, 2019.
20. Mathijs Schuts, Feng Zhu, Faranak Heidarian, and Frits W. Vaandrager. Modelling clock synchronization in the chess gmac WSN protocol. In *Proceedings First Workshop on Quantitative Formal Methods: Theory and Applications, QFM 2009, Eindhoven, The Netherlands, 3rd November 2009.*, pages 41–54, 2009.
21. Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems, Third International Symposium Organized Jointly with the Working Group Provably Correct Systems - ProCoS, Lübeck, Germany, September 19-23, Proceedings*, pages 694–715, 1994.
22. B. Woźna-Szcześniak and A. Zbrzezny. Checking MTL properties of discrete timed automata via bounded model checking. *Fundam. Inform.*, 135(4):553–568, 2014.
23. Bożena Woźna-Szcześniak and Andrzej Zbrzezny. Checking MTL properties of discrete timed automata via bounded model checking. *Fundam. Inform.*, 135(4):553–568, 2014.
24. Agnieszka Zbrzezny. *Selected SAT- and SMT-based model checking methods*. PhD thesis, Institue of Computer Science, Polish Academy of Sciences, 2017.
25. Agnieszka Zbrzezny and Andrzej Zbrzezny. Simple bounded MTL model checking for discrete timed automata (extended abstract). In *Proceedings of the 25th International Workshop on Concurrency, Specification and Programming, Rostock, Germany, September 28-30, 2016.*, pages 37–48, 2016.