Beyond Lassos: Complete SMT-Based Bounded Model Checking for Timed Automata

Roland Kindermann, Tommi Junttila, Ilkka Niemelä

Beyond Lassos: Complete SMT-Based Bounded Model Checking for Timed Automata

Roland Kindermann, Tommi Junttila, and Ilkka Niemelä

Aalto University
Department of Information and Computer Science
P.O.Box 15400, FI-00076 Aalto, Finland
{Roland.Kindermann, Tommi.Junttila, Ilkka.Niemela}@aalto.fi

Abstract. Timed automata (TAs) are a common formalism for modeling timed systems. Bounded model checking (BMC) is a verification method that searches for runs violating a property using a SAT or SMT solver. Previous SMT-based BMC approaches for TAs search for finite counter-examples and infinite lasso-shaped counter-examples. This paper shows that lasso-based BMC cannot detect counter-examples for some linear time specifications expressed, e.g., with LTL or Büchi automata. This paper introduces a new SMT-based BMC approach that can find a counter-example to any non-holding Büchi automaton or LTL specification and also, in theory, prove that a specification holds. Different BMC encodings tailored for the supported features of different SMT solvers are compared experimentally to lasso-based BMC and discretization-based SAT BMC.

1 Introduction

Timed automata, see, e.g., [1–3], are a convenient formalism for describing and model checking finite state systems augmented with real-valued clocks. There are many tools, Uppaal [4] to name just one, for timed automata and model checking algorithms for timed automata have been studied quite a lot during the last two decades. For verification, Uppaal treats the discrete part of a timed automaton's state in an explicit state fashion while using a symbolic representation for the time-related part. Other approaches use decision diagrams for the verification of timed automata [5, 6].

Bounded model checking (BMC) [7] is a symbolic model checking method that has been shown very efficient in bug hunting (i.e., finding counter-examples to specifications) for finite-state systems during the last ten years. Being fully symbolic, it can handle systems with high degree of non-determinism in data and input signals more naturally than explicit-state model checking methods. The basic idea behind BMC is, given a system, a specification, and an integer bound k, to build a propositional logic formula such that the formula is satisfiable if and only if the system has a counter-example of length at most k violating the specification. The bound is incremented until a satisfiable formula is found (implying that the specification does not hold for the system) or a completeness threshold is reached without finding any satisfiable formulas (implying that the specification holds for the system). Infinite runs are handled in BMC by considering finitely representable lasso-shaped infinite runs consisting of a finite prefix followed by a finite loop. In addition to finite state systems, BMC has also been applied

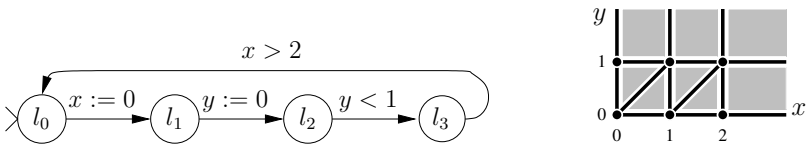
to timed automata [8–11]. When a propositional logic encoding is used (as e.g. in [8, 11]), the infinite state space of a timed automaton has to be reduced into a finite one; this can be achieved by using the region abstraction [1], see e.g. [8, 12] for two different propositional logic encodings of regions. A direct benefit of using the region abstraction is that the resulting BMC method can indeed detect whether a propositional ω-regular specification (expressed, e.g., with propositional linear time temporal logic LTL) holds on the system or not by considering lasso-shaped infinite runs only.

Propositional encodings of regions can be rather complicated and large for systems with many clocks with wide ranges. The introduction of Satisfiability Modulo Theories (SMT, see e.g. [13]) solvers with built-in support for reasoning over real and integer arithmetics has made it possible to devise BMC approaches for timed automata *without* using the region abstraction [9, 10]. In these approaches, the transition relation of the automaton is directly expressed as a propositional logic formula augmented with linear arithmetic constraints. In this paper we show that these previous SMT-based BMC approaches for timed automata are actually incomplete in the sense that for some timed automata they cannot find (a representative of) any infinite run despite such runs exist. This is basically caused by the fact that they search for lasso-shaped infinite runs (of the automaton, *not* of its region abstraction) but, unlike in the context of finite state systems, some timed automata have only non-lasso-shaped infinite runs. We propose an alternative *region-based SMT BMC encoding* for timed automata; in contrast to the propositional encoding, only the loop-detection part of the SMT encoding has to deal with regions but the rest of the encoding remains rather simple. We prove that (i) it can find a representative of an infinite run for any a timed automaton having an infinite run, and (ii) there is a completeness threshold, i.e., an integer such that there is an infinite run representative of at most that length unless there is no infinite run. Therefore, the encoding can be used to build an SMT-based BMC approach for model checking propositional LTL specifications on timed automata which is capable of finding counter-examples to all non-holding specifications and, in theory, also of proving that no counter-examples exist. Due to the use of the region abstraction, the formulas in the region-based BMC encoding are more complicated and may contain mixed integer / real expressions that are not supported by all current SMT solvers. We thus provide some alternative encodings and experimentally evaluate the efficiency of these.

We experimentally compare region-based SMT BMC against a traditional lasso-based SMT BMC encoding. We also compare our prototype implementation of SMT BMC methods against the highly optimized SAT BMC engine NuSMV [14] using a region-based propositional logic encoding [12]. The results show that region-based SMT BMC performs very good and is, in fact, sometimes significantly faster as it can find shorter counter-examples than the other methods tested.

2 Timed automata

We first give basic definitions for timed automata (see e.g. [1–3]). For the sake of simplicity, we use the very basic timed automata defined below in the theoretical parts of the paper. However, in practice (and also in the experimental part of the paper) one usually defines a network of timed automata that can also have (shared and local) finite



(a) automaton (b) regions

Fig. 1. A timed automaton and its regions

domain non-clock variables manipulated on the edges. The symbolic bounded model checking encodings presented later in the paper can be extended to handle both of these features, see, e.g., [9, 10] for how to handle synchronization in a network of timed automata. Similarly, we do not define any property description language in the theoretical part but consider the reachability problem for timed automata and the non-emptiness problem for timed automata extended with Büchi acceptance conditions (like in [1]). We then later study how bounded model checking can be used to solve these problems. Concerning practical model checking, solving the reachability problem corresponds to finding whether a timed automaton (or a network of such) can reach a bad state. Similarly, bearing in mind that linear-time temporal logic (LTL) properties can be translated into Büchi automata (see e.g. [15]), non-emptiness checking of timed Büchi automata corresponds to checking whether a timed automaton can violate an LTL specification. In the experimental part symbolic encodings for LTL model checking [16] are applied.

Let X be a set of real-valued *clock variables*. Then, a *clock valuation* v is a function $v : X \to \mathbb{R}_{\geq 0}$ and $v+\delta$ for a $\delta \in \mathbb{R}_{\geq 0}$ is the valuation for which $\forall x \in X : (v+\delta)(x) = v(x) + \delta$. The set of *clock constraints* over X, $\mathcal{C}(X)$, is defined by the grammar $C ::= \mathbf{true} \mid x \bowtie n \mid C \wedge C$ where $x \in X$, $\bowtie \in \{<, \leq, =, \geq, >\}$ and $n \in \mathbb{N}$. A valuation v satisfies a clock constraint C, denoted by $v \models C$, if it evaluates C to true.

A *timed automaton* (TA) is a tuple $\langle L, l_{\text{init}}, X, E, I \rangle$ where

- L is a finite set of *locations*,
- $l_{\text{init}} \in L$ is the *initial location* of the automaton,
- X is a finite set of real-valued *clock variables*,
- $E \subseteq L \times \mathcal{C}(X) \times 2^X \times L$ is a set of edges, each edge $\langle l, g, R, l' \rangle \in E$ specifying a *guard* g and a set R of *clocks to be reset*, and
- $I : L \to \mathcal{C}(X)$ assigns an *invariant* to each location.

As an example, Fig. 1(a) shows a timed automaton (from [1]). It has four locations $l_0, \ldots, l_3$, l_0 being the initial one, and two clocks, x and y. The edge $\langle l_0, \mathbf{true}, \{x\}, l_1 \rangle$ from l_0 to l_1 resets the clock x and the edge $\langle l_2, x < 1, \emptyset, l_3 \rangle$ has the guard $x < 1$. The invariants of all locations are $\mathbf{true}$.

A *state* of a timed automaton $\mathcal{A} = \langle L, l_{\text{init}}, X, E, I \rangle$ is a pair $\langle l, v \rangle$, where $l \in L$ is a location in $\mathcal{A}$ and v is a clock valuation over X. A state $\langle l, v \rangle$ is (i) *initial* if $l = l_{\text{init}}$ and $v(x) = 0$ for each $x \in X$, and (ii) *valid* if $v \models I(l)$. Let $\langle l, v \rangle$ and $\langle l', w \rangle$ be states of $\mathcal{A}$. There is a time *elapse step* from $\langle l, v \rangle$ to $\langle l', w \rangle$, denoted by $\langle l, v \rangle \xrightarrow{\text{e}} \langle l', w \rangle$, if (i) $l = l'$, (ii) $w = v + \delta$ for some $\delta \in \mathbb{R}_{>0}$, and (iii) $\langle l', w \rangle$ is a valid state. Intuitively, there is a time elapse step from a state to another if the second state can be reached

from the first one by letting a certain amount of time pass. There is a *discrete step* from $\langle l, v\rangle$ to $\langle l', w\rangle$, denoted by $\langle l, v\rangle \xrightarrow{\mathrm{d}} \langle l', w\rangle$, if there is an edge $\langle l, g, R, l'\rangle \in E$ such that (i) $v \models g$, (ii) $\langle l', w\rangle$ is a valid, and (iii) $w(x) = 0$ for all $x \in R$ and $w(x) = v(x)$ for all $x \in X \setminus R$. That is, discrete steps can be used to change the current location as long as the guard and the target location invariant are satisfied. A discrete step resets some clocks and leaves the other's values unchanged, i.e., a discrete step does not take any time. For situations in which the type of step between two states is insignificant, we define that $\langle l, v\rangle \to \langle l', w\rangle$ iff $\langle l, v\rangle \xrightarrow{\mathrm{e}} \langle l', w\rangle$ or $\langle l, v\rangle \xrightarrow{\mathrm{d}} \langle l', w\rangle$.

A *run* of $\mathcal{A}$ is a finite or infinite sequence $\pi = \langle l_0, v_0\rangle\langle l_1, v_1\rangle \ldots$, such that (i) $\langle l_0, v_0\rangle$ is a valid initial state, and (ii) $\langle l_i, v_i\rangle \to \langle l_{i+1}, v_{i+1}\rangle$ for each consecutive pair of states in the sequence. As an example, the automaton in Fig. 1(a) has the run $\langle l_0, (0,0)\rangle\langle l_0, (0.7, 0.7)\rangle\langle l_1, (0, 0.7)\rangle\langle l_2, (0,0)\rangle\langle l_3, (0,0)\rangle$ where each clock valuation $\{x \mapsto v, y \mapsto w\}$ is abbreviated to (v, w). An infinite run is (i) *non-zeno* if the sum of time passed in time elapse steps in it is infinite, and (ii) *lasso-shaped* if it can be written as $\langle l_0, v_0\rangle \ldots \langle l_{i-1}, v_{i-1}\rangle\big(\langle l_i, v_i\rangle \ldots \langle l_k, v_k\rangle\big)^\omega$ for some $0 \le i \le k$. In the context of BMC we sometimes consider k the length of the lasso-shaped run, as it is the length needed to represent the run. The automaton in Fig.1(a) has a lasso-shaped non-zeno run $\langle l_0, (0,0)\rangle\big(\langle l_0, (2.1, 2.1)\rangle\langle l_1, (0, 2.1)\rangle\langle l_2, (0,0)\rangle\langle l_3, (0,0)\rangle\langle l_3, (2.1, 2.1)\rangle\big)^\omega$. While it does not have any zeno runs, the automaton obtained by removing the guard $x > 2$ has the lasso-shaped zeno run $\big(\langle l_0, (0,0)\rangle\langle l_1, (0,0)\rangle\langle l_2, (0,0)\rangle\langle l_3, (0,0)\rangle\big)^\omega$.

2.1 Model Checking Problems

As said earlier, we study two model checking problems for timed automata. Firstly,

Definition 1 (Reachability problem). *Given a timed automaton $\mathcal{A}$ and a location l, does $\mathcal{A}$ have a finite run $\langle l_0, v_0\rangle\langle l_1, v_1\rangle \ldots \langle l_k, v_k\rangle$ with $l_k = l$?*

Secondly, we define a *timed Büchi automaton* to be a tuple $\mathcal{B} = \langle L, l_{\text{init}}, X, E, I, F\rangle$ such that (i) $\langle L, l_{\text{init}}, X, E, I\rangle$ is a timed automaton, and (ii) $F \subseteq L$ is the set of *accepting locations*. States, steps, and runs are defined as for timed automata. A run of $\mathcal{B}$ is *accepting* if it is infinite and a location $l \in F$ occurs infinitely many times in it.

Definition 2 (Non-emptiness problem). *Given a timed Büchi automaton $\mathcal{B}$, does it have an accepting run?*

For example, consider the timed Büchi automaton obtained from the timed automaton in Fig. 1(a) by letting $F = \{l_3\}$. It has a lasso-shaped, non-zeno accepting run $\langle l_0, (0,0)\rangle\langle l_1, (0,0)\rangle\big(\langle l_2, (0,0)\rangle\langle l_3, (0,0)\rangle\langle l_3, (3,3)\rangle\langle l_0, (3,3)\rangle\langle l_1, (0,3)\rangle\big)^\omega$. Both the reachability and non-emptiness problems are PSPACE-complete [1].

2.2 The Region Abstraction

We will also need the classic concepts of regions and region automata [1] later in the paper. Assume a timed automaton $\mathcal{A} = \langle L, l_{\text{init}}, X, E, I\rangle$. For each clock $x \in X$, let m_x be the largest constant n occurring in any atom of form $x \bowtie n$ on the guards and invariants of the automaton. For each $v \in \mathbb{R}_{\ge 0}$, let $\lfloor v \rfloor \in \mathbb{N}$ be the integral and

$\mathrm{fract}(v) \in [0, 1[$ the fractional part of v, i.e., $v = \lfloor v \rfloor + \mathrm{fract}(v)$. Two valuations, v and w, over X are *equivalent*, denoted by $v \sim w$, if all the following conditions hold:

1. For each clock $x \in X$, either $\lfloor v(x) \rfloor = \lfloor w(x) \rfloor$ or $(v(x) > \mathrm{m}_x) \wedge (w(x) > \mathrm{m}_x)$.
2. For all pairs of clocks $x, y \in X$ with $v(x) \leq \mathrm{m}_x$ and $v(y) \leq \mathrm{m}_y$, it holds that $\mathrm{fract}(v(x)) \leq \mathrm{fract}(v(y))$ iff $\mathrm{fract}(w(x)) \leq \mathrm{fract}(w(y))$.
3. For all clocks $x \in X$ with $v(x) \leq \mathrm{m}_x$ it holds $\mathrm{fract}(v(x)) = 0$ iff $\mathrm{fract}(w(x)) = 0$.

A *region* is an equivalence class of valuations induced by the relation $\sim$, and the region of a valuation v is denoted by $[v]$. The set of all regions is denoted by $\mathrm{regions}(\mathcal{A})$ and it contains *at most* $|X|! \cdot 2^{|X|} \cdot \prod_{x \in X}(2\mathrm{m}_x + 2)$ regions [1].

As an example, Fig. 1(b) graphically illustrates the regions of the timed automaton in Fig. 1(a); the 28 regions are the thick black dots and lines as well as the gray areas.

The *region automaton* of a timed automaton $\mathcal{A}$ is the finite state automaton

$$\mathcal{A}^{\mathrm{R}} = \langle Q, q_{\text{init}}, \Delta \rangle,$$

where (i) $Q = L \times \mathrm{regions}(\mathcal{A})$ is the set of states, (ii) $q_{\text{init}} = \langle l_{\text{init}}, [v_0] \rangle$ with $v_0(x) = 0$ for all $x \in X$ is the initial state, and (iii) $\Delta \subseteq Q \times Q$ is the transition relation with $\langle \langle l, r \rangle, \langle l', r' \rangle \rangle \in \Delta$ iff $\exists v, v' : \langle l, v \rangle \rightarrow \langle l', v' \rangle \wedge [v] = r \wedge [v'] = r'$. A run of $\mathcal{A}^{\mathrm{R}}$ is a finite or infinite sequence $q_0 q_1 \ldots$ of states in Q such that (i) $q_0 = q_{\text{init}}$, and (ii) $\langle q_i, q_{i+1} \rangle \in \Delta$ for all consecutive pairs of states in the sequence.

A timed automaton $\mathcal{A}$ and its region automaton $\mathcal{A}^{\mathrm{R}}$ are bisimilar in the sense that

1. $\langle l, v \rangle \rightarrow \langle l', v' \rangle$ implies $\langle \langle l, [v] \rangle, \langle l', [v'] \rangle \rangle \in \Delta$, and
2. $\langle (l, r), (l', r') \rangle \in \Delta$ implies $\forall v : ([v] = r) \Rightarrow \exists v' : ([v'] = r') \wedge \langle l, v \rangle \rightarrow \langle l', v' \rangle$.

Thus $\mathcal{A}$ and $\mathcal{A}^{\mathrm{R}}$ also have corresponding runs: (i) if $\mathcal{A}$ has a run $\langle l_0, v_0 \rangle \langle l_1, v_1 \rangle \ldots$, then $\mathcal{A}^{\mathrm{R}}$ has a run $\langle l_0, [v_0] \rangle \langle l_1, [v_1] \rangle \ldots$, and (ii) if $\mathcal{A}^{\mathrm{R}}$ has a run $\langle l_0, r_0 \rangle \langle l_1, r_1 \rangle \ldots$, then $\mathcal{A}$ has a run $\langle l_0, v_0 \rangle \langle l_1, v_1 \rangle \ldots$ such that $[v_i] = r_i$ for each i. Note that some runs of $\mathcal{A}^{\mathrm{R}}$ may have both corresponding zeno runs and corresponding non-zeno runs in $\mathcal{A}$. We define that a *run in* $\mathcal{A}^{\mathrm{R}}$ *is non-zeno* if it has at least one corresponding non-zeno run in $\mathcal{A}$.

3 Bounded Model Checking for Reachability and Lassos

As explained in the introduction, the idea behind bounded model checking is to construct formulas whose satisfying interpretations correspond to runs having some desired property (e.g., reachability, Büchi acceptance) and *bounded length*. The bound is incremented until a satisfiable formula (and thus a run with the desired property) is found or a completeness threshold is reached (meaning that no such run exists). This section introduces BMC for finite runs and lasso-shaped infinite runs of timed (Büchi) automata. The lasso-based BMC for TAs is very similar to BMC for untimed systems and has been previously described in [9, 10]. Lasso-based BMC is complete for untimed finite state systems but, as will be shown, despite a previous claim not complete for TAs.

Let $\mathcal{A} = \langle L, l_{\text{init}}, X, E, I \rangle$ be a timed automaton (or a timed Büchi automaton $\langle L, l_{\text{init}}, X, E, I, F \rangle$) and let k be the "bound" i.e. the length of the runs currently considered. We first construct a quantifier-free first order formula $|[\mathcal{A}, k]|^{\text{runs}}$ using linear

arithmetics over reals whose satisfying interpretations represent $\mathcal{A}$'s runs of length k. For each clock $x \in X$, we introduce $k+1$ "timed copies" $x^{[0]}, x^{[1]}, \ldots, x^{[k]}$ where the variable $x^{[i]}$ gives the value of the clock x at the ith state in the run. If C is a clock constraint, then $C^{[i]}$ is the "timed version" of C obtained by substituting each clock $x \in X$ in it by $x^{[i]}$; e.g. $((x < 3) \wedge (y \geq 2))^{[4]} = (x^{[4]} < 3) \wedge (y^{[4]} \geq 2)$. To represent automaton locations in the run, we use the set $\{at^{[0]}, at^{[1]}, \ldots, at^{[k]}\}$ of variables over the domain L. Similarly, to select whether a time elapse or discrete step is taken at the ith step, we use the Boolean variables $elapse^{[0]}, \ldots, elapse^{[k-1]}$, and for the time taken in time elapse steps the real-valued variables $\delta^{[0]}, \ldots, \delta^{[k-1]}$. We now define the formula for runs of length k by

$$|[\mathcal{A}, k]|^{\text{runs}} := |[\mathcal{A}]|^{\text{init}} \wedge |[\mathcal{A}, k]|^{\text{inv}} \wedge |[\mathcal{A}, k]|^{\text{trans}}$$

where $|[\mathcal{A}]|^{\text{init}} := (at^{[0]} = l_{\text{init}}) \wedge \bigwedge_{x \in X}(x^{[0]} = 0)$ ensures that the values of $\{at^{[0]}\} \cup \{x^{[0]} \mid x \in X\}$ represent the initial state of $\mathcal{A}$, $|[\mathcal{A}, k]|^{\text{inv}} := \bigwedge_{0 \leq i \leq k} \bigwedge_{l \in L}(at^{[i]} = l) \Rightarrow I(l)^{[i]}$ forces all the $k+1$ states to be valid ones, and the formula $|[\mathcal{A}, k]|^{\text{trans}} := \bigwedge_{0 \leq i < k}(elapse^{[i]} \Rightarrow \phi^{[i]}) \wedge (\neg elapse^{[i]} \Rightarrow \psi^{[i]})$ captures the transition relation. The formula $\phi^{[i]} := (\delta^{[i]} > 0) \wedge (at^{[i+1]} = at^{[i]}) \wedge \bigwedge_{x \in X}(x^{[i+1]} = x^{[i]} + \delta^{[i]})$ encodes time elapse steps, while $\psi^{[i]}$ does the same for discrete steps:

$$\psi^{[i]} := (\delta^{[i]} = 0) \wedge \bigvee\nolimits_{\langle l, g, R, l' \rangle \in E} \Big((at^{[i]} = l) \wedge (at^{[i+1]} = l') \wedge g^{[i]} \wedge \bigwedge\nolimits_{x \in R}(x^{[i+1]} = 0) \wedge \bigwedge\nolimits_{x \in X \setminus R}\big(x^{[i+1]} = x^{[i]}\big) \Big)$$

The automaton $\mathcal{A}$ has a run $\langle l_0, v_0 \rangle \ldots \langle l_k, v_k \rangle$ iff the formula $|[\mathcal{A}, k]|^{\text{runs}}$ is satisfiable under any interpretation extending $\{at^{[i]} \mapsto l_i, x^{[i]} \mapsto v_i(x) \mid 0 \leq i \leq k, x \in X\}$.

BMC for reachability. Based on the tight correspondence between the runs of $\mathcal{A}$ and satisfying interpretations of $|[\mathcal{A}, k]|^{\text{runs}}$, one can use $|[\mathcal{A}, k]|^{\text{runs}}$ to solve the reachability problem. Given a timed automaton $\mathcal{A}$ and a location l in it, check whether the formula

$$|[\mathcal{A}, l, k]|^{\text{reach}} := |[\mathcal{A}, k]|^{\text{runs}} \wedge \bigvee_{0 \leq i \leq k} (at^{[i]} = l)$$

is satisfiable for some bound $k \in \{0, 1, \ldots\}$. If $|[\mathcal{A}, l, k]|^{\text{reach}}$ indeed is satisfiable, then one can construct a run of $\mathcal{A}$ ending in l from the satisfying interpretation for $|[\mathcal{A}, l, k]|^{\text{reach}}$. We return to the issue of completeness, i.e. detecting the case that l is not reachable, later in this section.

BMC for lasso-shaped infinite runs. Assume now that $\mathcal{B}$ is a timed Büchi automaton $\langle L, l_{\text{init}}, X, E, I, F \rangle$. We can use the formula $|[\mathcal{B}, k]|^{\text{runs}}$ to define a formula $|[\mathcal{B}, k]|^{\text{lasso}}$ such that the timed Büchi automaton $\mathcal{B}$ has an accepting infinite lasso-shaped run $\langle l_0, v_0 \rangle \ldots \langle l_{i-1}, v_{i-1} \rangle \big(\langle l_i, v_i \rangle \ldots \langle l_k, v_k \rangle\big)^{\omega}$ for some $1 \leq i < k$ iff $|[\mathcal{B}, k]|^{\text{lasso}}$ is satisfiable under an interpretation extending $\{at^{[i]} \mapsto l_i, x^{[i]} \mapsto v_i(x) \mid 0 \leq i \leq k, x \in X\}$.

To do this, we use an auxiliary set $loop^{[1]}, \ldots, loop^{[k]}$ of Boolean *loop variables* to detect loops in the finite runs represented with $|[\mathcal{B}, k]|^{\text{runs}}$. A variable $loop^{[i]}$ being true means that the $i-1$th and the kth state are the same, meaning that a lasso-shaped run can be obtained by looping back from the kth to the ith state. Furthermore, an auxiliary set $acc^{[1]}, \ldots, acc^{[k]}$ of Boolean variables is used to compute whether an accepting location is visited at the ith or later state in the run. We define

$$|[\mathcal{B}, k]|^{\text{lasso}} := |[\mathcal{B}, k]|^{\text{runs}} \wedge |[\mathcal{B}, k]|^{\text{loop}} \wedge |[\mathcal{B}, k]|^{\text{accept}}$$

where $|[\mathcal{B}, k]|^{\text{loop}} := \bigwedge_{1 \leq i \leq k} \Big(loop^{[i]} \Rightarrow (at^{[i-1]} = at^{[k]}) \wedge \bigwedge_{x \in X} (x^{[i-1]} = x^{[k]}) \Big)$ detects the loops in the finite runs, and $|[\mathcal{B}, k]|^{\text{accept}} := (acc^{[k]} \Leftrightarrow \bigvee_{l \in F} (at^{[k]} = l)) \wedge \bigwedge_{1 \leq i \leq k-1} \big(acc^{[i]} \Leftrightarrow acc^{[i+1]} \vee \bigvee_{l \in F} (at^{[i]} = l) \big) \wedge \big(\bigvee_{1 \leq i \leq k} (loop^{[i]} \wedge acc^{[i]}) \big)$ forces satisfying interpretations to correspond to accepting runs only: there shall be a loop in the run and an accepting location in the loop.

The encoding can easily be modified to accept only non-zeno runs. A lasso-shaped run is zeno iff it does not contain any time elapse step in its looping part. Thus, non-zenoness can be enforced by requiring the looping part to contain at least one time elapse step. For this purpose k Boolean variables $el^{[0]}, \ldots, el^{[k-1]}$ where $el^{[i]}$ being true for a given i means that there is a time elapse step after the ith state. Looping back to the ith state is allowed only if $el^{[i]}$ is true, leading to the following conjunct:

$$|[\mathcal{B}, k]|^{\text{lnz}} := (el^{[k-1]} \Leftrightarrow elapse^{[k-1]}) \wedge \bigwedge_{1 \leq i \leq k-2} \Big(el^{[i]} \Leftrightarrow (elapse^{[i]} \vee el^{[i+1]}) \Big) \wedge \bigwedge_{1 \leq i \leq k} loop^{[i]} \Rightarrow el^{[i-1]}$$

Note that for a run that loops back from the last to the ith state, the step that loops back from the last to the ith state is of the same type as the step from the $i-1$th to the ith state. Thus, it is sufficient if the step from the $i-1$th to the ith state is a time elapse step in order to have a time elapse step in the looping part of the run, which is the reason why we only require $el^{[i-1]}$ to hold if $loop^{[i]}$ holds and not $el^{[i]}$.

(In)completeness. We now study the completeness of the two BMC encodings given above. As in [7, 9], by completeness we mean the ability to find a run if one exists or to demonstrate that no runs exists if this is the case. To show completeness, a completeness threshold is needed, i.e. an integer bound K such that a run of interest (witnessing reachability or Büchi acceptance) exists if and only if one exists with bound K or less.

Previous completeness results. For finite state systems, the simple run-unfolding BMC is complete for reachability problems and lasso-BMC is complete for non-emptiness under Büchi acceptance conditions [7]. When considering timed automata, the reachability BMC encoding given above is complete [9]. This is because (i) a location l is reachable in an automaton $\mathcal{A}$ iff it is reachable in its region automaton $\mathcal{A}^{\text{R}}$ due to bisimilarity (recall Sect. 2.2), (ii) $\mathcal{A}^{\text{R}}$ has at most $|L| \cdot |X|! \cdot 2^{|X|} \cdot \prod_{x \in X} (2\text{m}_x + 2)$ states, which implies that l is reachable with at most $K_{\text{reach}} = |L| \cdot |X|! \cdot 2^{|X|} \cdot \prod_{x \in X} (2\text{m}_x + 2) - 1$ steps in $\mathcal{A}^{\text{R}}$, and (iii) thus l is reachable with at most K_{reach} steps in $\mathcal{A}$. Therefore, l is

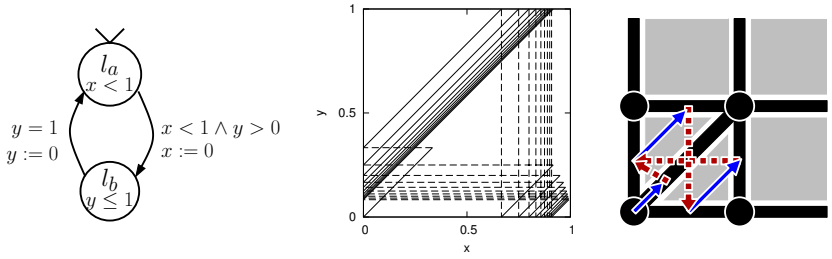
reachable in $\mathcal{A}$ iff $|[\mathcal{A}, l, K_{\text{reach}}]|^{\text{reach}}$ is satisfiable. Of course, using K_{reach} as a bound is usually infeasible in practice but its existence guarantees the theoretical completeness of the BMC approach.

Incompleteness for Büchi acceptance conditions on TAs. We now show that, despite a previous claim in [9], lasso-based BMC is not complete for checking non-emptiness of timed Büchi automata or, in fact, for even detecting whether a timed automaton has at least one infinite run. Incompleteness of an encoding can best be shown by giving an automaton for which the encoding can not find a run. For this purpose, we will use automaton in Fig. 2(a) which, as will be demonstrated, does not have any lasso-shaped infinite runs despite having infinite non-zeno runs. Therefore, lasso-based BMC will fail to find any run for the automaton and *is thus incomplete* for (i) detecting whether a timed automaton has at least one infinite run, (ii) deciding non-emptiness of timed Büchi automata, and (iii) model checking propositional LTL on timed automata.

Let us now study the automaton in Fig. 2(a) a bit more closely. It has two locations, l_a and l_b, and two clocks, x and y. For a given infinite run in the automaton, let t_i^{a} be the time spent in l_a the ith time the run visits l_a and t_i^{b} be defined analogously for l_b. The edge from l_a can only be traversed when x is less than one. Furthermore, x is reset when the edge is traversed. Therefore, the time between two subsequent traversals of this edge is strictly less than one time unit: $\forall i \geq 1 : t_i^{\text{b}} + t_{i+1}^{\text{a}} < 1$. Analogously, the time between two subsequent traversals of the edge form l_b to l_a is exactly one time unit: $\forall i \geq 1 : t_i^{\text{a}} + t_i^{\text{b}} = 1$. Combining the two formulas results in $\forall i \geq 1 : t_{i+1}^{\text{a}} < t_i^{\text{a}}$ and $\forall i \geq 1 : t_{i+1}^{\text{b}} > t_i^{\text{b}}$, i.e. in any run the time spent in l_a strictly decreases from any visit to the next and the time spent in l_b strictly increases. Furthermore, the difference between the two clocks in l_b equals the time spent in l_a on the previous visit and vice versa. Consequently, the difference between the clocks strictly increases in l_a and strictly decreases in l_b. Thus, the same location is never reached twice with the same clock difference and therefore, no run can ever visit the same state twice. Hence, the automaton does not have any lasso-shaped run. This, however, does not mean that the automaton does not have any infinite runs at all. A valid infinite run, e.g., stays $\frac{1}{i+2}$ time units in location l_a the ith time it is visited and $1 - \frac{1}{i+2}$ units in location l_b. Figure 2(b) shows the clock valuations on a ten time unit long prefix of this run, while Fig. 2(c) illustrates the clock regions visited by the run. Note that while the run is not lasso-shaped in the space of clock valuations, it indeed is lasso-shaped in the clock regions. Also, the run is non-zeno as the time passing is $\sum_{i=1}^{\infty} \frac{1}{i+2} + 1 - \frac{1}{i+2} = \infty$.

4 Region-based BMC

As shown above, lasso-based BMC is not complete for checking non-emptiness of timed Büchi automata. This section introduces a region-based BMC approach that fixes this problem. The approach is inspired by the observation that even though the automaton in Fig. 2(a) does not have any lasso-shaped infinite runs, its region automaton does. Based on this observation, our new encoding modifies the lasso-based BMC encoding by not requiring the last state of the run to be exactly the same as a previous state but



(a) The automaton

(b) The clock valuations of a prefix of a run

(c) The same run in the region abstraction

Fig. 2. A non-empty timed automaton that does not have any lasso-shaped run

only in the same region as a previous state. Such a run corresponds to a lasso-shaped run in the region automaton and thus to a set of infinite runs of the TA.

Note that, in order to get runs in which time does not suddenly just stop, it is not sufficient to require that the last state of the run is in the same region as an earlier state. For many clock valuations, it is possible to reach a valuation in the same region by a sufficiently small time elapse step. Thus, it is often possible to extend a finite run with a short time elapse step to get a run in which the last and second to last state have the same location and clock valuations in the same region. If the only requirement to a run was that the last state is in the same clock region as a previous state, such a run would be accepted. While it is possible to extend such a run to a valid infinite (though zeno) run by adding smaller and smaller time elapse steps, in practice runs of the described type are typically not of interest as they correspond to time not progressing past a certain point. Therefore, we exclude runs of the described type by restricting to non-zeno runs.

In order to check whether two clock valuations are in the same clock region, one needs to split up each clock's value into its integral an fractional parts. Therefore, region-based BMC uses two additional variables, $x_{\text{int}}^{[i]}$ and $x_{\text{fract}}^{[i]}$, for each clock $x \in X$ and each state index i. The integer variable $x_{\text{int}}^{[i]}$ represents the integral part of the value of x while the real-valued variable $x_{\text{fract}}^{[i]}$ represents its fractional part. Given a timed Büchi automaton $\mathcal{B} = \langle L, l_{\text{init}}, X, E, I, F \rangle$, the region-based BMC encoding is

$$|[\mathcal{B}, k]|^{\text{region}} := |[\mathcal{B}, k]|^{\text{runs}} \wedge |[\mathcal{B}, k]|^{\text{accept}} \wedge |[\mathcal{B}, k]|^{\text{close}} \wedge |[\mathcal{B}, k]|^{\text{nz}}$$

where $|[\mathcal{B}, k]|^{\text{runs}}$ and $|[\mathcal{B}, k]|^{\text{accept}}$ are defined as in Sect. 3, $|[\mathcal{B}, k]|^{\text{nz}}$ is used to ensure non-zenoness and is defined later in this section and $|[\mathcal{B}, k]|^{\text{close}}$, detecting whether the clock valuations in the $i-1$th and kth states are in the same region, is defined as

$$|[\mathcal{B}, k]|^{\text{close}} := \bigwedge_{0 \leq i \leq k} \bigwedge_{x \in X} \left(0 \leq x_{\text{fract}}^{[i]} \wedge x_{\text{fract}}^{[i]} < 1 \wedge x^{[i]} = x_{\text{int}}^{[i]} + x_{\text{fract}}^{[i]}\right) \wedge \bigwedge_{1 \leq i \leq k} \left(loop^{[i]} \Rightarrow (at^{[i-1]} = at^{[k]} \wedge S_{i,k})\right)$$

with $S_{i,k} := \bigwedge_{x \in X} \Big((x_{\text{int}}^{[i-1]} = x_{\text{int}}^{[k]}) \vee (x_{\text{int}}^{[i-1]} > \mathrm{m}_x \wedge x_{\text{int}}^{[k]} > \mathrm{m}_x) \Big) \wedge \Big(x_{\text{int}}^{[k]} \leq \mathrm{m}_x \Rightarrow \big((x_{\text{fract}}^{[i-1]} = 0 \Leftrightarrow x_{\text{fract}}^{[k]} = 0) \wedge \bigwedge_{y \in X \setminus \{x\}} (y_{\text{int}}^{[k]} \leq \mathrm{m}_y \Rightarrow (x_{\text{fract}}^{[i-1]} \leq y_{\text{fract}}^{[i-1]} \Leftrightarrow x_{\text{fract}}^{[k]} \leq y_{\text{fract}}^{[k]})) \big) \Big)$. The first line in $|[\mathcal{B}, k]|^{\text{close}}$ ensures the integral+fractional decomposition of clock values. Its sub-expression $x^{[i]} = x_{\text{int}}^{[i]} + x_{\text{fract}}^{[i]}$ mixes integer and real variables; such mixed-type expressions are supported, e.g., by the SMT solver Yices [17]. As they are, however, not supported by all SMT solvers, an alternative encoding not requiring them will be introduced in Sect. 5. Analogously to the lasso-based encoding, $loop^{[i]}$ is a Boolean variable indicating that it is possible to loop from the last state in the run to the ith state, or, more precisely, to a state in the same region as the ith state.

4.1 Ensuring Non-Zenoness

In order to complete the encoding, a way to ensure non-zenoness of the run is needed. In lasso-based BMC, non-zenoness can be ensured by requiring that the looping part of the run contains at least one time elapse step. For region-based BMC, this approach does not work. Any concrete run corresponding to a lasso-shaped region automaton run having a time elapse step in the looping part is guaranteed to have an infinite number of time elapse steps. The sum of the delays of these steps is, however, not guaranteed to be diverging. Thus, we will instead ensure non-zenoness using the following theorem:

Theorem 1 (Alur and Dill [1]). *An infinite run of a region automaton that has an infinite number of time elapse steps is non-zeno iff each clock $x \in X$ either infinitely often is zero or infinitely often has a value greater than m_x.*[1]

It is straightforward to turn this theorem into a non-zenoness condition for the BMC encoding. Due to the fact that x's value cannot decrease unless x is reset, x is guaranteed to exceed m_x in all states of the looping part if it exceeds m_x in at least one state of the looping part and is never reset inside the looping part. Therefore it is sufficient to require that x is either zero at least once inside the looping part or exceeds its maximum value in the run's last state which is guaranteed to be in its looping part.

Requiring at least one time elapse step in the looping part of the run can be done using the formula $|[\mathcal{B}, k]|^{\text{lnz}}$ as for lasso-shaped paths. Furthermore, for any clock $x \in X$, k Boolean variables $ok_x^{[i]}$ are used to ensure that either x is zero at least once in the looping part of the run or exceeds its maximum value in the last state of the run. This results in the following definition of $|[\mathcal{B}, k]|^{\text{nz}}$:

$$|[\mathcal{B}, k]|^{\text{nz}} := \bigwedge_{x \in X} \Big(ok_x^{[k]} \Leftrightarrow (x^{[k]} = 0 \vee x^{[k]} > \mathrm{m}_x) \Big) \wedge \bigwedge_{1 \leq i \leq k-1} \Big(ok_x^{[i]} \Leftrightarrow (x^{[i]} = 0 \vee ok_x^{[i+1]}) \Big) \wedge \bigwedge_{1 \leq i \leq k} \Big(loop^{[i]} \Rightarrow \bigwedge_{x \in X} ok_x^{[i]} \Big) \wedge |[\mathcal{B}, k]|^{\text{lnz}}$$

By Theorem 1, any run with infinitely many time elapse steps in the region automaton that does not satisfy the "infinitely often $x = 0 \vee x > \mathrm{m}_x$" condition for a clock $x \in X$ is zeno. Therefore, the bound required to find a run using our encoding is the length of the shortest lasso-shaped non-zeno run in the region automaton.

[1] Note that using the slightly different definition of the region automaton in [1] any infinite run of the region automaton is guaranteed to have an infinite number of elapse steps.

4.2 Completeness

In the following the completeness of the proposed approach will be shown. More precisely, it will be shown that, given a timed Büchi automaton $\mathcal{B} = \langle L, l_{\text{init}}, X, E, I, F \rangle$, $K_{\text{region}} := (|X| + 3) \cdot |L| \cdot |X|! \cdot 2^{|X|} \cdot \prod_{x \in X} (2\mathrm{m}_x + 2)$ is a completeness threshold, meaning that our approach when used with bound K_{region} can find a non-zeno run for $\mathcal{B}$ if $\mathcal{B}$ has any non-zeno run. To do this, we prove the following theorem.

Theorem 2. *Unless a given timed Büchi automaton $\mathcal{B}$ does not have a single accepting non-zeno run, its region automaton $\mathcal{B}^{\text{R}}$ has an accepting lasso-shaped non-zeno run of length at most K_{region}.*

The theorem will be proven in two parts, each of which is stated as a lemma. The proofs of the lemmas are given in the appendix.

Lemma 1. *If $\mathcal{B}$ has an infinite accepting non-zeno run $\pi = \langle l_0, v_0 \rangle \langle l_1, v_1 \rangle \ldots$, then $\mathcal{B}$'s region automaton $\mathcal{B}^{\text{R}}$ has a* lasso-shaped *accepting non-zeno run $\pi^{\text{R}}_{\text{lasso}}$.*

Lemma 2. *If $\mathcal{B}^{\text{R}}$ has an accepting lasso-shaped non-zeno run, then $\mathcal{B}^{\text{R}}$ has an accepting lasso-shaped non-zeno run of length at most K_{region}.*

Note that the completeness threshold K_{region} should first and foremost be considered a theoretical result, as the given completeness threshold in practice even for small systems is infeasibly high. In order to find more practical completeness thresholds, an approach similar to the ones used for untimed systems in [18] could be used.

5 Alternative encodings

Avoiding mixed-type expressions and unbounded integer variables. One challenge in the encoding introduced in Sect. 4 is that it uses mixed-type expressions that use both integer and real variables; such are not supported by some SMT solvers. One can, however, modify the encoding to get rid of the mixed-type expressions. To do this, instead of the $x^{[i]}$ clock variables, we use $x^{[i]}_{\text{int}}$ variables for their integral parts and $x^{[i]}_{\text{fract}}$ variables for their fractional parts. Likewise, the difference variables for time elapse steps $\delta^{[i]}$ are each replaced by two variables $\delta^{[i]}_{\text{int}}$ and $\delta^{[i]}_{\text{fract}}$ for their integral and fractional parts. After this, (in)equalities of form $x^{[i]} \bowtie n$ for an $n \in \mathbb{N}$, used e.g. to encode the TA's guards and invariants and to enforce that the initial values of the clocks are zero, are modified to use $x^{[i]}_{\text{int}}$ and $x^{[i]}_{\text{fract}}$ instead of $x^{[i]}$. For instance, $x^{[i]} \leq n$ is replaced with $x^{[i]}_{\text{int}} < n \vee (x^{[i]}_{\text{int}} = n \wedge x^{[i]}_{\text{fract}} = 0)$. Expressions of the form $x^{[i+1]} = x^{[i]} + \delta^{[i]}$ have to be replaced by a case distinction to ensure that the fractional part of $x^{[i+1]}$ is always less than one: each expression of form $x^{[i+1]} = x^{[i]} + \delta^{[i]}$ is replaced with $\big((x^{[i]}_{\text{fract}} + \delta^{[i]}_{\text{fract}} < 1) \Rightarrow (x^{[i+1]}_{\text{int}} = x^{[i]}_{\text{int}} + \delta^{[i]}_{\text{int}} \wedge x^{[i+1]}_{\text{fract}} = x^{[i]}_{\text{fract}} + \delta^{[i]}_{\text{fract}})\big) \wedge \big((x^{[i]}_{\text{fract}} + \delta^{[i]}_{\text{fract}} \geq 1) \Rightarrow (x^{[i+1]}_{\text{int}} = x^{[i]}_{\text{int}} + \delta^{[i]}_{\text{int}} + 1 \wedge x^{[i+1]}_{\text{fract}} = x^{[i]}_{\text{fract}} + \delta^{[i]}_{\text{fract}} - 1)\big)$. When all expressions using the $x^{[i]}$ clock variables are modified as described, the encoding does not contain any mixed-type expressions anymore and can thus be used with a SMT solver not supporting them.

As the exact value of a clock $x \in X$ is irrelevant once it exceeds m_x, the encoding can be further modified to turn the $x^{[i]}_{\text{int}}$ and $\delta^{[i]}_{\text{int}}$ variables into bounded integer

variables without affecting the correctness of the approach: the domain of each $x_{\text{int}}^{[i]}$ is $\{0, \ldots, \mathrm{m}_x + 1\}$ while $\delta_{\text{int}}^{[i]}$ has the domain $\{0, \ldots, \max_{x \in X} \mathrm{m}_x\}$. A simple case distinction similar to the technique shown for $x^{[i]} + \delta^{[i]}$ expressions is used to set each variable's value to its maximum value whenever it in the unbounded case would exceed its maximum. Restricting to bounded integers is necessary for SMT solvers that do not support unbounded integer variables.

Alternative non-zenoness condition. In [19] an alternative way to ensure that all runs of a timed Büchi automaton are non-zeno is proposed. The approach modifies the TA using one additional clock such that accepting states only count as accepting if at least one time unit passed since the last accepting state. While using this approach for BMC would be feasible for Büchi acceptance conditions, it is not clear how it could be extended to other approaches like directly encoding an LTL formula (cf. e.g. [16]) as in such an encoding there is no notion of accepting states. Thus we propose a similar but slightly different approach as an alternative encoding ensuring non-zenoness by requiring that the looping part of the run is at least one time unit long. The resulting non-zenoness condition is simpler and needs less variables than the original one proposed in Sect. 4:

$$|[\mathcal{B}, k]|^{\text{nz2}} := (\Sigma^{[k-1]} = \delta^{[k-1]}) \wedge \bigwedge_{0 \leq i \leq k-2} (\Sigma^{[i]} = \Sigma^{[i+1]} + \delta^{[i]}) \wedge \bigwedge_{1 \leq i \leq k} (\mathit{loop}^{[i]} \Rightarrow \Sigma^{[i]} \geq 1)$$

Correctness. A lasso-shaped run of the region automaton returns to a previously visited region at the end of the looping part. If the looping part is at least one time unit long, as required by our alternative non-zenoness-encoding, the integral part of the value of any clock $x \in X$ changes somewhere in the looping part. Unless x has exceeded m_x, this implies that a new region is reached. In this case, x has to be reset before the looping part can return to the original region. Hence, each clock x is either reset or exceeds m_x in the looping part, i.e. infinitely often, which according to Theorem 1 implies non-zenoness. Therefore any lasso-shaped run of the region automaton along whose looping part at least one time unit passes is non-zeno and the alternative encoding is correct.

Note that the opposite is not true, i.e. not every lasso-shaped non-zeno run of the region automaton has a looping part along which at least one time unit passes. This implies that using the alternative non-zenoness-encoding sometimes results in needing a higher bound to find a run than would be required with the original encoding.

6 Experiments

The Fischer mutual exclusion protocol and an industrial model with both handmade and random properties were used to compare the different SMT BMC encodings and discretization-based SAT BMC experimentally.

Setup. In the experiments, lasso-based BMC (not requiring non-zenoness, cf. Sect. 3) was compared to region-based BMC. In addition to the basic encoding given in Sect. 4, the two non-mixed type encodings and the alternative non-zenoness condition given in

Table 1. Execution times and maximum bound reached for the Fischer protocol (median over 11 executions, "*to*" means timeout)

	$\neg\mathbf{GF}proc1.crit$										$\neg(\mathbf{GF}proc1.crit \wedge \mathbf{GF}\neg proc1.crit)$									
# processes	lasso BMC		mixed type		non-mixed		limited integer		discretization		lasso BMC		mixed type		non-mixed		limited integer		discretization	
	Time	Bound	Time	Bound	Time	Bound	Time	Bound	Time	Bound	Time	Bound	Time	Bound	Time	Bound	Time	Bound	Time	Bound
2	0.35	9	0.28	**5**	0.2	**5**	0.28	**5**	**0.11**	7	0.33	9	0.31	**6**	0.25	**6**	0.3	**6**	**0.16**	10
3	1.73	12	0.26	**5**	0.28	**5**	0.28	**5**	**0.15**	7	1.41	12	**0.33**	**6**	**0.33**	**6**	0.34	**6**	4.07	13
4	16.94	15	0.35	**5**	0.33	**5**	0.34	**5**	**0.22**	7	14.54	15	**0.41**	**6**	0.43	**6**	0.42	**6**	*to*	14
5	83.46	18	0.5	**5**	0.42	**5**	0.43	**5**	**0.31**	7	66.53	18	0.54	**6**	**0.51**	**6**	0.53	**6**	*to*	13
7	*to*	20	0.99	**5**	0.57	**5**	0.6	**5**	**0.51**	7	*to*	20	1.39	**6**	**0.7**	**6**	0.74	**6**	*to*	13
10	*to*	19	1.53	**5**	**0.9**	**5**	0.97	**5**	0.95	7	*to*	18	*to*	5	**1.13**	**6**	1.22	**6**	*to*	13
20	*to*	15	*to*	4	**2.45**	**5**	2.59	**5**	3.57	7	*to*	18	*to*	5	**3.46**	**6**	3.86	**6**	*to*	12
30	*to*	14	*to*	4	**5.23**	**5**	5.45	**5**	10.04	7	*to*	18	*to*	5	**8.12**	**6**	8.98	**6**	*to*	11

Sect. 5 were used. The basic encoding is referred to as "mixed type, basic nz." while the modifications are referred to as "non-mixed" for the unlimited range integer non-mixed type encoding, "limited integer" for the limited range integer encoding and "alt. nz." for the alternative non-zenoness condition (cf. Sect. 5).

For comparison to the SMT-based BMC variants, discretization-based SAT BMC was applied. That is, the real-time models were transformed into discrete time models [12] and then checked using a discrete time BMC SAT encoding [16] implemented in NuSMV [14] 2.5.4. The used translation algorithm encodes the region abstraction and thus is complete in the same sense as region-based SMT BMC. All BMC approaches were used in an incremental fashion, i.e. successively increasing the bound using an incremental SMT / SAT solver. The real-time models were encoded as symbolic timed transition systems [12], a symbolic representation variant of timed automata. Properties were specified as LTL formulas and encoded using [16].

In our experiments, we focused on comparing different BMC approaches. It should, however, be noted that both benchmarks have been previously studied using different verification methods including the model checker Uppaal [4].

All experiments were conducted on GNU/Linux computers with AMD Opteron 2435 CPUs limited to ten minutes of CPU time and 4 GB of RAM. For the SMT-based BMC variants, Yices [17] 1.0.33 was used.

Fischer protocol. As a first benchmark, the Fischer mutual exclusion protocol, a standard benchmark for real-time verification, and two non-holding properties ("process one can only finitely often be in the critical section" and "process one can only finitely often enter and exit the critical section") were used. The protocol has been previously studied using Uppaal [20]. Table 1 shows the time needed for finding counter-examples and the maximum bound reached on instances of different sizes. For space restrictions, only results for the basic non-zenoness condition are listed. However, results for the alternative non-zenoness encoding are very similar.

For the first property the discretization-based approach scaled only slightly worse than the non mixed-type region-based encodings. Lasso-based BMC, in contrast, scaled significantly worse, already timing out at size 7. The reason is that the region based

Table 2. Execution times in seconds for finding counter-examples to the non-holding properties on the industrial benchmark (median over 11 executions, "*to*" means timeout, "nz." abbreviates "non-zenoness condition")

	Entire model						Medium size submodel						Small submodel					
Property	lasso BMC	mixed basic nz.	mixed alt. nz.	non-mixed basic nz.	non-mixed alt. nz.	discreti-zation	lasso BMC	mixed basic nz.	mixed alt. nz.	non-mixed basic nz.	non-mixed alt. nz.	discreti-zation	lasso BMC	mixed basic nz.	mixed alt. nz.	non-mixed basic nz.	non-mixed alt. nz.	discreti-zation
1	**1.0**	43.55	*to*	3.97	*to*	*to*	**0.55**	0.85	*to*	0.73	*to*	*to*	**0.29**	0.69	*to*	0.51	*to*	*to*
2	**2.19**	*to*	*to*	18.11	20.4	*to*	**0.4**	0.43	0.56	0.41	0.47	*to*	**0.21**	0.3	0.37	0.27	0.27	*to*
3	**0.66**	2.12	16.81	2.04	1.78	*to*	**0.32**	0.49	0.81	0.42	0.5	*to*	**0.21**	1.0	0.36	0.28	0.29	*to*
4	**2.16**	*to*	*to*	21.1	19.27	*to*	**0.32**	0.47	0.59	0.45	0.52	*to*	**0.21**	0.45	0.38	0.3	0.37	*to*
5	**1.87**	*to*	*to*	19.74	22.14	*to*	**0.32**	0.63	0.65	0.47	0.54	*to*						
6	**1.8**	*to*	*to*	20.36	30.36	*to*	**0.33**	0.58	0.59	0.44	0.49	*to*						
7	**0.47**	0.66	1.11	0.73	0.8	*to*	**0.27**	0.3	0.32	0.31	0.32	*to*						
8	**0.67**	1.94	4.55	1.26	1.75	*to*	**0.33**	0.47	0.47	0.43	0.44	*to*						
9	**0.57**	0.63	1.07	0.71	0.77	*to*												
10	**1.32**	*to*	*to*	3.41	4.64	*to*												
11	*to*	*to*	*to*	**25.4**	41.12	*to*												
12	*to*	*to*	*to*	**26.15**	26.71	*to*												
13	*to*	*to*	*to*	**27.54**	28.68	*to*												

BMC variants can find significantly shorter counter-examples. Similarly, the discretization-based method may return significantly longer counter-examples, as it only allows time elapse steps that either leave the ordering of clocks' fractional parts unchanged or advance to the very next region. This often makes it necessary to break up a single time elapse step in a counter-example into multiple time elapse steps in the discretized model. While not affecting execution times for the first property by much, the longer counter-examples for the discretization-based approach had a huge impact on the second, more complicated property. Another interesting result is that the mixed-type encoding performed significantly worse than the non-mixed type encodings, indicating that avoiding mixed-type expressions can be beneficial even for SMT-solvers that support them.

Industrial benchmark. As second benchmark, a model of an emergency diesel generator intended for the use in a nuclear power plant was used. The model is fairly large, having 24 clocks and its location being defined by the valuation of 130 finite domain state variables. Additionally, two submodels (7 clocks / 64 state variables and 6 clocks / 36 state variables) which are sufficient for verifying some of the properties were used. The model has previously been studied using different verification methods including Uppaal [21, 12]. Table 2 shows the execution times for non-holding properties. For space restrictions, the results for the limited range integer encoding, which are just slightly worse than the unlimited range integer encoding, are omitted here. The discretization-based approach timed out for all properties on all submodels, clearly indicating that its applicability is restricted to small models. The choice of non-zenoness condition was irrelevant for all except one property. Unlike for the Fischer protocol, lasso-based BMC performed significantly better for some properties while performing significantly worse for others. A likely explanation again is the length of the counter-examples that can be found using the respective variants.

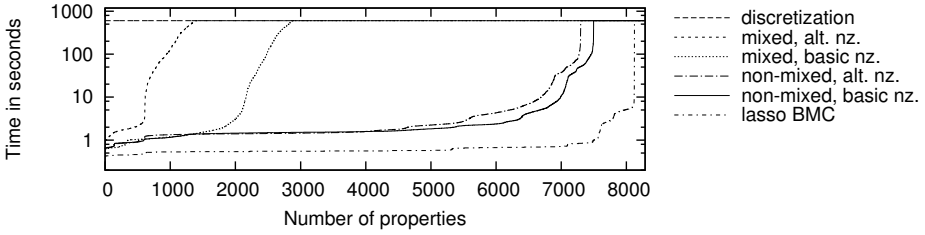


Fig. 3. Execution time by number of properties for random properties on the industrial benchmark. A point (x, y) indicates that for x properties y or less time was needed (each), corresponding to a plot of quantiles

Furthermore, random properties for the industrial model were generated using the following LTL patterns: unconditional fairness ($\mathbf{GF}x$), strong fairness ($\mathbf{GF}x \Rightarrow \mathbf{GF}y$), weak fairness ($\mathbf{FG}x \Rightarrow \mathbf{GF}y$), "leads to" ($\mathbf{G}(x \Rightarrow \mathbf{F}y)$) and $\mathbf{G}x \Rightarrow \mathbf{G}y$, a pattern that had been used by the authors of the industrial benchmark. 2000 properties of each type were randomly selected, except for the unconditional fairness pattern for which all 371 generated properties were selected. Figure 3 shows the times required to find counter-examples for all properties for which at least one method found a counter-example. Again, the discretization-based approach timed out for all properties. For random properties, lasso-based BMC was clearly faster than the region-based approaches. A likely explanation is that most random properties have very short counter-examples, meaning that the potentially smaller bound needed by region-based BMC is outweighted by the more complicated transition relation. Furthermore, random properties that involve only non-timing related parts of the system tend to have exceptionally short zeno counter-examples, further favoring lasso-based BMC which does not require non-zenoness.

7 Conclusions

In this paper, we have shown that traditional lasso-based SMT BMC is not complete for model checking linear-time properties on timed automata, introduced region-based SMT BMC to fix this problem, and shown its completeness. Different variations of the approach tailored for supported features of different SMT solvers are given. The variations of region-based SMT BMC have been experimentally compared to each other, to lasso-based SMT BMC and to discretization-based SAT BMC. The experiments indicate that region-based SMT BMC outperforms discretization-based SAT BMC. For hand-made properties, region-based SMT BMC also was more robust than lasso-based SMT BMC. For random properties, however, lasso-based SMT BMC performed better.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2) (1994) 183–235
2. Alur, R.: Timed automata. In: CAV 1999. Volume 1633 of LNCS., Springer (1999) 8–22

3. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets. Volume 3098 of LNCS., Springer (2003) 87–124
4. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: FM-RT 2004. Volume 3185 of LNCS., Springer (September 2004) 200–236
5. Behrmann, G., Larsen, K.G., Pearson, J., Weise, C., Yi, W.: Efficient timed reachability analysis using clock difference diagrams. In: CAV 1999. Volume 1633 of LNCS., Springer (1999) 341–353
6. Beyer, D., Noack, A.: Can decision diagrams overcome state space explosion in real-time verification? In: FORTE. Volume 2767 of LNCS., Springer (2003) 193–208
7. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: TACAS 1999. Volume 1579 of LNCS., Springer (1999) 193–207
8. Woźna, B., Zbrzezny, A., Penczek, W.: Checking reachability properties for timed automata via SAT. Fundamenta Informatica **55**(2) (2003) 223–241
9. Sorea, M.: Bounded model checking for timed automata. Electronic Notes in Theoretical Computer Science **68**(5) (2002)
10. Audemard, G., Cimatti, A., Kornilowicz, A., Sebastiani, R.: Bounded model checking for timed systems. In: FORTE 2002. Volume 2529 of LNCS., Springer (2002) 243–259
11. Malinowski, J., Niebert, P.: SAT based bounded model checking with partial order semantics for timed automata. In: TACAS 2010. Volume 6015 of LNCS., Springer (2010) 405–419
12. Kindermann, R., Junttila, T., Niemelä, I.: Modeling for symbolic analysis of safety instrumented systems with clocks. In: ACSD 2011, IEEE (2011) 185–194
13. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability. IOS Press (2009) 825–885
14. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An opensource tool for symbolic model checking. In: CAV 2002. Volume 2404 of LNCS., Springer (2002) 359–364
15. Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press (1999)
16. Biere, A., Heljanko, K., Junttila, T., Latvala, T., Schuppan, V.: Linear encodings of bounded LTL model checking. Logical Methods in Computer Science **2**(5:5) (2006) 1–64
17. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: CAV 2006. Volume 4144 of LNCS., Springer (2006) 81–94
18. Clarke, E.M., Kroening, D., Ouaknine, J., Strichman, O.: Completeness and complexity of bounded model checking. In: VMCAI 2004. Volume 2937 of LNCS., Springer (2004) 85–96
19. Tripakis, S., Yovine, S., Bouajjani, A.: Checking timed büchi automata emptiness efficiently. Formal Methods in System Design **26**(3) (2005) 267–292
20. Larsen, K.G., Pettersson, P., Yi, W.: Model-checking for real-time systems. In: FCT 1995. Volume 965 of LNCS., Springer (1995) 62–88
21. Lahtinen, J., Björkman, K., Valkonen, J., Frits, J., Niemelä, I.: Analysis of an emergency diesel generator control system by compositional model checking. VTT Working Papers 156, VTT Technical Research Centre of Finland (2010)

A Appendix: Proofs of Lemmas 1 and 2

Lemma 1. *If $\mathcal{B}$ has an infinite accepting non-zeno run $\pi = \langle l_0, v_0\rangle\langle l_1, v_1\rangle \ldots$, then $\mathcal{B}$'s region automaton $\mathcal{B}^{\mathrm{R}}$ has a* lasso-shaped *accepting non-zeno run $\pi^{\mathrm{R}}_{\mathrm{lasso}}$.*

Proof. Let $\pi^{\mathrm{R}} = \langle l_0, [v_0]\rangle\langle l_1, [v_1]\rangle \ldots$ be the region automaton run corresponding to π. In the following we will construct a lasso-shaped run $\pi^{\mathrm{R}}_{\mathrm{lasso}}$ of $\mathcal{B}^{\mathrm{R}}$ from a prefix of π^{R}.

Let $X^{\mathrm{U}} \subseteq X$ be the set of clocks which are reset only finitely many times along π. Then according to Theorem 1, any clock $x \in X^{\mathrm{U}}$ exceeds m_x infinitely often on π^{R}, which combined with the fact that x is only reset finitely often, yields that x exceeds m_x in all states after a given point in π, i.e. $x \leq \mathrm{m}_x$ holds only finitely often on π. Let $s^{\mathrm{R}}_{\mathrm{inf}}$ be an arbitrary state occurring infinitely often in π^{R}. Note that such a state is guaranteed to exist due to the fact that $\mathcal{B}^{\mathrm{R}}$ has only finitely many states. Each $x \in X^{\mathrm{U}}$ is guaranteed to exceed m_x in $s^{\mathrm{R}}_{\mathrm{inf}}$, as $s^{\mathrm{R}}_{\mathrm{inf}}$ occurs infinitely often on π^{R} while $x \leq \mathrm{m}_x$ holds only finitely often. Let i_{inf} the lowest number such that $\langle l_{i_{\mathrm{inf}}}, [v_{i_{\mathrm{inf}}}] \rangle = s^{\mathrm{R}}_{\mathrm{inf}}$. We will turn π^{R} into a lasso-shaped run by looping back from a later occurrence of $s^{\mathrm{R}}_{\mathrm{inf}}$ to its occurrence at index i_{inf}.

Let i_{end} be the lowest index such that $\langle l_{i_{\mathrm{end}}}, [v_{i_{\mathrm{end}}}] \rangle = s^{\mathrm{R}}_{\mathrm{inf}}$ and (i) every clock in $X \setminus X^{\mathrm{U}}$ is reset, (ii) there is a time elapse step and (iii) an accepting state of $\mathcal{B}$ visited between the i_{inf}th and the i_{end}th state of π. As each of the described events occurs infinitely often on π, a corresponding choice of i_{end} is possible. Now let

$$\pi^{\mathrm{R}}_{\mathrm{lasso}} := \langle l_1, [v_1] \rangle \dots \langle l_{i_{\mathrm{inf}}-1}, [v_{i_{\mathrm{inf}}-1}] \rangle (\langle l_{i_{\mathrm{inf}}}, [v_{i_{\mathrm{inf}}}] \rangle \dots \langle l_{i_{\mathrm{end}}-1}, [v_{i_{\mathrm{end}}-1}] \rangle)^{\omega}$$

Then $\pi^{\mathrm{R}}_{\mathrm{lasso}}$ is a lasso shaped run of $\mathcal{B}^{\mathrm{R}}$. Furthermore, any clock $x \in X^{\mathrm{U}}$ exceeds its m_x value infinitely often on $\pi^{\mathrm{R}}_{\mathrm{lasso}}$, namely in $\langle l_{i_{\mathrm{inf}}}, [v_{i_{\mathrm{inf}}}] \rangle = s^{\mathrm{R}}_{\mathrm{inf}}$. In addition, due to the way i_{end} was chosen, any clock $X \setminus X^{\mathrm{U}}$ is reset infinitely often, an accepting state visited infinitely often and a time elapse step taken infinitely often on $\pi^{\mathrm{R}}_{\mathrm{lasso}}$. According to Theorem 1, this implies the non-zenoness of $\pi^{\mathrm{R}}_{\mathrm{lasso}}$, which concludes the proof of Lemma 1. □

Lemma 2. *If $\mathcal{B}^{\mathrm{R}}$ has an accepting lasso-shaped non-zeno run, then $\mathcal{B}^{\mathrm{R}}$ has an accepting lasso-shaped non-zeno run of length at most K_{region}.*

Proof. We will prove the lemma by construction of a lasso-shaped non-zeno run of $\mathcal{B}^{\mathrm{R}}$ of length at most K_{region} from the lasso-shaped run $\pi^{\mathrm{R}}_{\mathrm{lasso}}$ constructed in the proof of Lemma 1. Note that we only needed certain parts of the looping part of $\pi^{\mathrm{R}}_{\mathrm{lasso}}$ for showing non-zenoness of the run, namely one time elapse step, one reset step per clock in $X \setminus X^{\mathrm{U}}$, an accepting state and $s^{\mathrm{R}}_{\mathrm{inf}}$. Thus, we can replace every segment between two such relevant steps / states by the shortest path between them without affecting the non-zenoness of the run or the fact that it is accepting. The resulting loop consists of $\left|X \setminus X^{\mathrm{U}}\right|$ reset steps, one time elapse step, one accepting state, $s^{\mathrm{R}}_{\mathrm{inf}}$ and $\left|X \setminus X^{\mathrm{U}}\right| + 3$ shortest paths in between. Any shortest path has length of at most $\left|\mathcal{B}^{\mathrm{R}}\right| - 2$, not counting initial and final state, where $\left|\mathcal{B}^{\mathrm{R}}\right| \leq |L| \cdot |X|! \cdot 2^{|X|} \cdot \prod_{x \in X}(2\mathrm{m}_x + 2)$ is the number of states in $\mathcal{B}^{\mathrm{R}}$. Taking into account that every step consists of two states, this yields a length of at most $(\left|X \setminus X^{\mathrm{U}}\right|+1) \cdot 2 + 2 + (\left|X \setminus X^{\mathrm{U}}\right|+3) \cdot (\left|\mathcal{B}^{\mathrm{R}}\right|-2) < (|X|+3) \cdot \left|\mathcal{B}^{\mathrm{R}}\right| \leq (|X|+3) \cdot |L| \cdot |X|! \cdot 2^{|X|} \cdot \prod_{x \in X}(2\mathrm{m}_x + 2) \leq K_{\mathrm{region}}$.

After modifying the looping part of our lasso-shaped run, we can replace the non-looping part with the shortest path from the initial state of $\mathcal{B}^{\mathrm{R}}$ to any state in the looping part. Note that in the resulting run the set of states that occur in the non-looping part is disjunct from the set of states in the looping part. This implies that we can reduce our upper bound to the length of the looping part by $|X| + 3$ for each state in the non-looping part and, ultimately, that K_{region} is an upper bound for the length of the entire resulting lasso-shaped run. □