Certifying Emptiness of Timed Büchi Automata

Simon Wimmer[1][0000−0001−5998−4655], Frédéric Herbreteau[2][0000−0002−1029−2356], and Jaco van de Pol[3][0000−0003−4305−0625]

[1] Fakultät für Informatik, Technische Universität München, Munich, Germany
wimmers@in.tum.de
[2] Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, 33400, Talence, France
[3] Aarhus University, Department of Computer Science, Denmark

Abstract. Model checkers for timed automata are widely used to verify safety-critical, real-time systems. State-of-the-art tools achieve scalability by intricate abstractions. We aim at further increasing the trust in their verification results, in particular for checking liveness properties. To this end, we develop an approach for extracting certificates for the emptiness of timed Büchi automata from model checking runs. These certificates can be double checked by a certifier that we formally verify in Isabelle/HOL. We study liveness certificates in an abstract setting and show that our approach is sound and complete. To also demonstrate its feasibility, we extract certificates for several models checked by TChecker and Imitator, and validate them with our verified certifier.

Keywords: Timed Automata · Certification · Model Checking

1 Introduction

Real-time systems are notoriously hard to analyze due to intricate timing constraints. A number of model checkers for timed automata (TA) [1] have been implemented and successfully applied to the verification of safety-critical timed systems. Checking liveness properties of timed automata has revealed to be particularly important, as emphasized by a bug in the standard model of the CSMA/CD protocol that has been discovered only recently [16]. Several algorithms have been implemented to scale the verification of liveness specifications to larger systems [33,32,26,22,16]. Users of timed automata model checkers put a high amount of trust in their verification results. However, as verification algorithms get more complex, it becomes highly desirable to justify the users' confidence in their correctness.

There are two main approaches to ensure high degrees of trustworthiness of automated tools: verification and certification. In the first approach, correctness of the verification tool (its implementation and its theory) is proved using another semi-automated method. This technique has been applied to model checkers [34,13] and SAT solvers [7]. In the second approach, the automated tool produces a certificate, i.e. a proof for its verification result. Then an independent tool, the certifier, checks that the proof is indeed valid. In the best case,

the certifier itself is formally verified. Examples include SAT certificate checking [23,20] and unreachability checking of TA [37].

The certification approach promises many advantages over verification, since certificate checking is much simpler than producing the certificate. This drastically reduces the burden of semi-automated verification, which is a laborious task. While proving correctness of a competitive verification tool might be prohibitively complicated, it may be feasible to construct an efficient verified certifier instead (in the case of SAT [23], the verified certifier was even faster than the original SAT solvers). Finally, there is a wide variety of model checking algorithms and high-performance implementations, which are suited for different situations. Instead of verifying them one by one, these tools could produce certificates in a common format, so they can be checked by a single verified certifier.

1.1 Related Work

Model checking LTL properties for timed automata [1,33,32,26,22,16] consists of three conceptual steps: the LTL formula is transformed into a Büchi automaton, the semantics of the TA is computed as a (finite) zone graph, and the cross-product of these objects is checked for accepting cycles. The two main alternative algorithms for detecting accepting cycles are Nested Depth-First Search (NDFS) and the inspection of the Strongly Connected Components (SCC). The NDFS algorithm was generalized to TA in LTSmin [22,21] and extended to parametric TA in Imitator [28,2]. The SCC-based algorithm has also been generalized to TA in TChecker [16,19]. Both algorithms support *abstraction and subsumption between states* to reduce the state space.

Verified model checking. An early approach targeted the verification of a μ-calculus model checker in Coq [27]. The NDFS algorithm was checked in the program verifier Dafny [31,25], while a multi-core version of it was checked in the program verifier Vercors [30,8]. A complete, *executable* LTL model checker was verified in the interactive theorem prover Isabelle/HOL [13] and later extended with partial-order reduction [9]. A verified model checker for TA, Munta [34], has also been constructed in Isabelle/HOL [36,29].

Certification. A certifier for reachability properties in TA has been proposed very recently [37]. A certification approach for LTL model checking was proposed in [15]. It uses k-liveness to reduce the problem to IC3-like invariant checking.

Contributions. In this paper, we extend certificates for unreachability of TA [37] to certificates for liveness properties, i.e. emptiness of timed Büchi automata (TBA). We propose a common certification approach for tools using different algorithms and various abstractions [16,22]. These certificates can be much smaller than the original state space, due to the use of subsumption and abstraction. The difficulty here is that a careless application of subsumption can introduce spurious accepting cycles. Our new contributions are [4]:

- We introduce an abstract theory for certificates of Büchi emptiness, which can be instantiated for zone graphs of TBA with subsumptions.

[4] An artifact containing our code and benchmarks is available on figshare [35].

- We developed a fully, mechanically verified certifier in Isabelle/HOL. In particular, our certifier retains the ability to check certificates in parallel.
- We show that the previous certifier for reachability and our extension to Büchi emptiness are compatible with implicit abstraction techniques for TA.
- We demonstrate feasibility by generating and checking certificates for two external model checkers, representing the NDFS and the SCC approach.

Note that checking counter-examples is easy in practice, but checking "true" model checking results is much harder. This is exactly what we address with certifying emptiness of TBA. The main application would be to increase the confidence in safety-critical real-time applications, which have been verified with an existing model checker. Another possible application of the certifier would be to facilitate a new model checking contest for liveness properties of TA.

2 Timed Automata and Model Checking

In this section, we set the stage for the rest of the paper by recapitulating the basic notions of TA and summarizing the essential concepts of TBA verification.

2.1 Verification Problems for Timed Automata

A TA $A = (Q, q_0, F, I, T, X)$ is a finite automaton extended with a finite set of *clocks* X. Q is a finite set of states with initial state $q_0 \in Q$ and accepting states $F \subseteq Q$. I associates an *invariant* constraint to every state and T associates a *guard* constraint g and *clock reset* $R \subseteq X$ to each transition. Here *(clock) constraints* are conjunctions of formulas $x\#c$, where x is a clock, $c \in \mathbb{N}$ and $\# \in \{<, \leq, =, \geq, >\}$. Observe that we exclude diagonal constraints of the form $x - y\#c$. An example of a timed automaton is depicted in Figure 1.

A clock valuation $v : X \to \mathbb{R}_{\geq 0}$ associates a non-negative real value to each clock $x \in X$. A configuration is a pair (q, v) where q is a state and v is a clock valuation. The initial configuration is $(q_0, \mathbf{0})$. Without loss of generality, we assume that the initial clock valuation $\mathbf{0}$ satisfies the invariant $I(q_0)$. There are two kind of steps from a configuration (q, v):

delay $(q, v) \to_\delta (q, v')$ for a delay $\delta \in \mathbb{R}_{\geq 0}$ if for every clock $x \in X$, $v'(x) = v(x) + \delta$, and v' satisfies the invariant $I(q)$;

transition $(q, v) \to_t (q', v')$ for transition $t = (q, g, R, q') \in T$ if v satisfies the guard g, $v'(x) = 0$ if $x \in R$ and $v'(x) = v(x)$ otherwise, and v' satisfies $I(q')$.

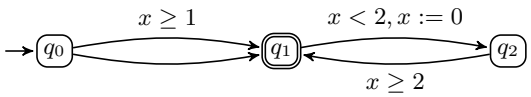


Fig. 1: Timed (Büchi) automaton with initial state q_0 and accepting state q_1.

We write $(q, v) \rightarrow_{\delta,t} (q', v')$ if there exists a configuration (q, v'') such that $(q, v) \rightarrow_{\delta} (q, v'') \rightarrow_{t} (q', v')$. A run of a timed automaton is an (infinite) sequence of transitions of the form: $(q_0, \mathbf{0}) \rightarrow_{\delta_0,t_0} (q_1, v_1) \rightarrow_{\delta_1,t_1} \cdots$. A run is *non-Zeno* if the sum of its delays is unbounded.

The *reachability problem* asks, given a timed automaton A, if there exists a finite run from the initial configuration $(q_0, \mathbf{0})$ to an accepting configuration (q_n, v_n) such that $q_n \in F$.

In Timed Büchi Automata (TBA), F is interpreted as a Büchi acceptance condition. The *liveness problem* then asks, whether a given TBA A is non-empty, i.e. if there is an infinite non-Zeno run from the initial configuration $(q_0, \mathbf{0})$ that visits infinitely many accepting configurations (q_i, v_i) with $q_i \in F$. In this paper, we work under the common assumption that TA only admit non-Zeno runs (see [33] for a construction to enforce this on every TA).

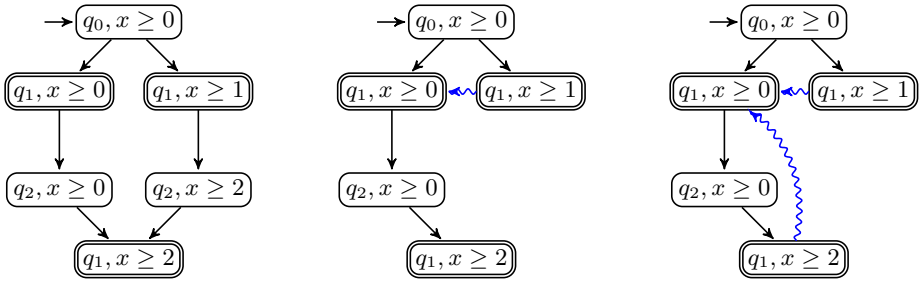
Both problems are known to be PSPACE-complete [1]. Due to density of time, these two verification problems cannot be solved directly from the transition system induced by configurations and steps. A well-known solution to this problem is the region graph construction of Alur and Dill [1]. Yet, it is not used in practice, as the region graph is enormous even for rather simple automata.

2.2 Zone Graph and Abstractions

The practical solution that is implemented in state-of-the-art tools like UPPAAL [24], TChecker [19] and the Imitator tool [2] is based on zones. Let us fix a set of clocks X. A zone Z is a set of valuations represented as a conjunction of constraints of the form $x \# c$ or $x - y \# c$ for $x, y \in X$, $\# \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{Z}$. Zones can be efficiently represented using Difference Bound Matrices (DBMs) [12]. Moreover, zones admit a canonical representation, hence equality and inclusion of two zones can be checked efficiently [6].

We now define the symbolic semantics [11] of a TA A. Let q, q' be two states of A, and let $W, W' \subseteq \mathbb{R}^X_{\geq 0}$ be two non-empty sets of clock valuations. We have $(q, W) \Rightarrow^t (q', W')$ for some transition $t \in T$, if W' is the set of all clock valuations v' for which there exists a valuation $v \in W$ and a delay $\delta \in \mathbb{R}_{\geq 0}$ such that $(q, v) \rightarrow_{\delta,t} (q', v')$. In other words, W' is the strongest postcondition of W along transition t. The symbolic semantics of A , denoted by $\Rightarrow$, is the union of all $\Rightarrow^t$ over $t \in T$. The symbolic semantics is a sound and complete representation of the finite and infinite runs of A. Indeed, A admits a finite (resp. infinite) run $(q_0, v_0) \rightarrow_{\delta_0,t_0} (q_1, v_1) \rightarrow_{\delta_1,t_1} \ldots (q_n, v_n) \rightarrow_{\delta_n,t_n} \ldots$ if and only if there exists a finite (resp. infinite) path $(q_0, W_0) \Rightarrow^{t_0} (q_1, W_1) \Rightarrow^{t_1} \ldots (q_n, W_n) \Rightarrow^{t_n} \ldots$ such that $v_i \in W_i$ for all $i \geq 0$ and $W_0 = \{\mathbf{0}\}$ [11]. It is well-known that if Z is a zone, and $(q, Z) \Rightarrow (q', W')$ then W' is a zone as well [6]. Since $\{\mathbf{0}\}$ is a zone, all the reachable nodes in $\Rightarrow$ are zones as well. The reachable part of $\Rightarrow$ is called the *zone graph* of A. The nodes of the zone graph are denoted as (q, Z) in the sequel and the zone graph is simply denoted by its transition relation $\Rightarrow$. Fig. 2a depicts the zone graph of the automaton in Fig. 1.

Still, the zone graph of a timed automaton may be infinite. As a remedy, finite abstractions have been introduced in the literature [11,4,5].



(a) Zone Graph (b) Liveness compatible (c) Not compatible

Fig. 2: Three subsumption graphs for the automaton in Fig. 1.

An *abstraction* α transforms a zone Z into a zone $\alpha(Z)$ such that $Z \subseteq \alpha(Z)$, $\alpha(\alpha(Z)) = \alpha(Z)$, and every run that is feasible from a valuation $v' \in \alpha(Z)$ is simulated by a run from a valuation $v \in Z$. Such abstractions are called extrapolations in the literature [5]. An abstraction is finite when the set of abstracted zones $\{\alpha(Z) \mid Z \text{ is a zone}\}$ is finite. Given an abstraction α, the *abstracted zone graph* has initial node $(q, \alpha(\{\mathbf{0}\}))$ and transitions of the form $(q, Z) \Rightarrow_\alpha^t (q', \alpha(Z'))$ for each transition $(q, Z) \Rightarrow^t (q', Z')$. Let $\Rightarrow_\alpha$ denote the union of all $\Rightarrow_\alpha^t$ over $t \in T$. The abstracted zone graph is sound and complete: there is a run $(q_0, v_0) \rightarrow_{\delta_0,t_0} (q_1, v_1) \rightarrow_{\delta_1,t_1} \ldots (q_n, v_n)$ $(\rightarrow_{\delta_n,t_n} \ldots)$ in A if and only if there is an infinite path $(q_0, Z_0) \Rightarrow_\alpha^{t_0} (q_1, Z_1) \Rightarrow_\alpha^{t_1} \ldots (q_n, Z_n)$ $(\Rightarrow_\alpha^{t_n} \ldots)$ with $v_i \in Z_i$ for all $i \geq 0$. Hence, when α is a finite abstraction, the verification problems for a TA A can be algorithmically solved from its abstracted zone graph. The abstraction Extra^+_{LU} [5] is implemented by state-of-the-art verification tools UPPAAL [24] and TChecker [19]. Our results hold for any finite, sound and complete abstraction. The abstracted zone graph is denoted $\Rightarrow_\alpha$ in the sequel.

2.3 Subsumption

Consider the TA in Figure 1 and its zone graph in Figure 2a. Observe that every run that is feasible from node $(q_1, x \geq 1)$ is also feasible from $(q_1, x \geq 0)$ since the zone $x \geq 1$ is included in the zone $x \geq 0$ (recall that zones are sets of clock valuations). We say that $(q_1, x \geq 1)$ is *subsumed* by the node $(q_1, x \geq 0)$. As a result, if an accepting node is (repeatedly) reachable from $(q_1, x \geq 1)$, then an accepting node is also (repeatedly) reachable from $(q_1, x \geq 0)$.

This leads to a crucial optimization for the verification of TA: reachability and liveness verification problems can be solved without exploring subsumed nodes. This optimization is called inclusion abstraction in [11]. Figure 2b shows the graph obtained when the exploration is stopped at node $(q_1, x \geq 1)$. All the runs that are feasible from $(q_1, x \geq 1)$ are still represented in this graph, as they can be obtained by first taking the subsumption edge from $(q_1, x \geq 1)$ to $(q_1, x \geq 0)$ (depicted as a blue squiggly arrow), and then any sequence of

(actual or subsumption) edges from $(q_1, x \geq 0)$. Such graphs with both actual and subsumption edges are called *subsumption graphs* in the sequel.

It is tempting to use subsumption as much as possible, and only explore maximal nodes (w.r.t. zone inclusion). While this is correct for the verification of reachability properties, subsumption must be used with care for liveness verification. The bottom node $(q_1, x \geq 2)$ in Figure 2b is also subsumed by the node $(q_1, x \geq 0)$. A subsumption edge can thus be added between these two nodes as depicted in Figure 2c. However, due to this new subsumption edge, the graph has a Büchi accepting path (of actual and subsumption edges) that does not correspond to any run of the timed automaton in Figure 1. Indeed, subsumption leads to an overapproximation of the runs of the automaton. While all the runs from node $(q_1, x \geq 2)$ are feasible from node $(q_1, x \geq 0)$, the converse is not true: the transition $q_1 \xrightarrow{x<2, x:=0} q_2$ is not feasible from $(q_1, x \geq 2)$.

The subsumption graphs in Figure 2b and 2c can be seen as certificates issued by verification algorithms. The graph in Figure 2b is a valid certificate for liveness verification as 1) it contains no accepting paths, and 2) every run of the automaton is represented in the graph. In constrast, the graph in Figure 2c is not a valid certificate for liveness verification as it has an accepting path that does not correspond to any run of the automaton. In the next sections, we introduce an algorithm to check the validity of certificates produced by liveness verification algorithms, as well as a proven implementation of the algorithm.

3 Certificates for Büchi Emptiness

In this section, we study certificates for Büchi emptiness in the setting of a slight variation of well-structured transition systems [14]. First, we present reachability invariants, which certify that every run in the original system can be simulated on the states given in the invariant. Next, we show that the absence of certain cycles in the invariant is sufficient to prove that the original transition system does not contain accepting runs. Then, we add a proof of absence of these cycles to the certificate. Finally, we instantiate this framework for the case of TA.

3.1 Self-Simulating Transition Systems

A *transition system* $(S, \to)$ consists of a set of states S and a transition relation $\to \subseteq S \times S$. If S is clear from the context, we simply write $\to$. We say that $s_1 \to s_2 \to \ldots \to s_n$ is a path or that $s_1 \to s_2 \to \ldots$ is an (infinite) run in $\to$ if $s_i \to s_{i+1}$ for all i. Given an initial state s_0 and a predicate for accepting states ϕ, the path $s_0 \to s_1 \to \ldots \to s_n$ is accepting if $\phi(s_n)$. A run $s_0 \to s_1 \to \ldots$ is an (accepting) Büchi run if $\phi(s_i)$ for infinitely many i.

A transition system $\to$ is simulated by the transition system $\to'$ if there exists a simulation relation $\sqsubseteq$ such that:

$$\forall s, s', t.\ s \sqsubseteq s' \wedge s \to t \longrightarrow \exists t'.\ s' \to' t' \wedge t \sqsubseteq t'$$

This *simulation property* can be lifted to paths and runs:

Proposition 1. *If $s_1 \to s_2 \to \ldots \to s_n (\to \ldots)$ is a path (run) and $s_1 \sqsubseteq t_1$, then there is a path (run) $t_1 \to' t_2 \to' \ldots \to' t_n (\to \ldots)$ with $s_i \sqsubseteq t_i$ for all i.*

Definition 1. *A* self-simulating transition system (SSTS) *$(S, \to, \preceq)$ consists of a transition system $(S, \to)$ and a quasi-order (a reflexive and transitive relation) $\preceq \subseteq S \times S$ on states such that $\to$ is simulated by $\to$ itself for $\preceq$.*

In comparison to well-structured transition systems [14], our definition is slightly more relaxed, as we only demand that $\preceq$ is a quasi order, not a well-quasi order. Intuitively, transitivity of $\preceq$ is needed to allow for correct simulation by arbitrary "bigger" nodes. In TA, $\preceq$ corresponds to subsumption $\subseteq$, and $\to$ corresponds to $\Rightarrow$.

3.2 Reachability Invariants on Abstract Transition Systems

In this section, we introduce the concept of *reachability invariants* for SSTS.

Definition 2. *A set $I \subseteq S$ is a* reachability invariant *of an SSTS $(S, \to, \preceq)$ iff for all $s \in I$ and t with $s \to t$, there exists a $t' \in I$ such that $t \preceq t'$.*

A useful invariant is also fulfilled by some inital state. Such states will show up in theorems below. In the remainder, unless noted otherwise, $(S, \to, \preceq)$ is an SSTS and I is a reachability invariant of it. Figures 2a to 2c all form a reachability invariant for the zone graph from Figure 2a.

As was observed by Wimmer and von Mutius [37], reachability invariants can directly be applied as certificates for *unreachability*.

Definition 3. *A predicate ϕ (for accepting states) is compatible with an SSTS $(S, \to, \preceq)$ iff for all $s, s' \in S$, if $\phi(s)$ and $s \preceq s'$, then also $\phi(s')$.*

An invariant I can now certify that no accepting state s with $\phi(s)$ is reachable:

Theorem 1. *If $\forall s \in I.\ \neg\phi(s)$, for some compatible ϕ, $s_0 \in S$ and $s_0' \in I$ with $s_0 \preceq s_0'$, then there is no accepting path $s_0 \to s_1 \to \ldots \to s_n$ with $\phi(s_n)$.*

Note that this approach to certifying unreachability is also complete: if no accepting state is reachable from s_0 in $(S, \to, \preceq)$, we can simply set $I := S$. However, this is not practical for infinite transition systems, of course. Thus we will revisit the question of completeness for TA below.

Finally, we observe that the invariant can be limited to a restriction of $\preceq$.

Definition 4. *A pair $(I, \trianglelefteq)$ of a set $I \subseteq S$ and a binary relation $\trianglelefteq$ is a* restricted reachability invariant *of an SSTS $(S, \to, \preceq)$ iff:*

1. *For all $s \in I$ and t with $s \to t$, there exists a $t' \in I$ such that $t \trianglelefteq t'$.*
2. *For all s, t, if $s \trianglelefteq t$, then also $s \preceq t$.*

In Figure 2, the $\rightsquigarrow$-arrows would play the role of $\trianglelefteq$. In Figure 2b, $(q_1, x \geq 0)$ is subsumed by both $(q_1, x \geq 1)$ and $(q_1, x \geq 2)$, but as we have seen in Figure 2c, it is crucial to disregard these subsumptions. Therefore we need to consider restricted reachability invariants.

For any restricted reachability invariant, we can define a simulating transition system $\to_{\trianglelefteq}$:

Definition 5. *The transition system $(S, \to_{\trianglelefteq})$ is defined such that $s \to_{\trianglelefteq} t'$ iff there exists a t such that $s \to t$ and $t \trianglelefteq t'$.*

This simulation theorem is the key property of restricted reachability invariants [5]:

Theorem 2. *Given $s_1 \trianglelefteq t_1$ with $t_1 \in I$, if $s_1 \to s_2 \to \ldots \to s_n (\to \ldots)$ is a path (run), then there is a path (run) $t_1 \to_{\trianglelefteq} t_2 \to_{\trianglelefteq} \ldots \to_{\trianglelefteq} t_n (\to \ldots)$ such that $s_i \trianglelefteq t_i$ and $t_i \in I$ for all i.*

Analogously to $\to_{\trianglelefteq}$, the transition system $\to_{\trianglerighteq}$ can be defined, and Theorem 2 can be proved for $\to_{\trianglerighteq}$. This is used for the proof of Theorem 1 (see [37]).

3.3 Büchi Emptiness on Abstract Transition Systems

In this section, we first give a general means of certifying that a transition system does not contain a cycle, and then combine the idea with reachability invariants to certify the absence of Büchi runs on SSTS.

Definition 6. *Given a transition system $\to$ and an accepting state predicate ϕ, a* topological numbering *of $\to$ is a function f with an integer range such that:*

1. *For all s, t, if $s \to t$, then $f(s) \geq f(t)$.*
2. *For all s, t, if $s \to t$ and $\phi(s)$, then $f(s) > f(t)$.*

Proposition 2. *Let f be a topological numbering of $\to$ and ϕ. If there exists a path of the form $s \to s_1 \to s_2 \to \ldots \to s$, then $\neg\phi(s)$.*

These certificates are also complete:

Proposition 3. *If there is no path $s \to s_1 \to s_2 \to \ldots \to s$ with $\phi(s)$ in $\to$, then the following are topological numberings for $\to$.*

1. *The number of accepting states that are reachable from a node: $f(s) := |\{x \mid s \to^* x \wedge \phi(x)\}|$ (assuming $\{x \mid s \to^* x \wedge \phi(x)\}$ is finite for any s).*
2. *If h is a topological numbering (in the classical sense) of the strongly connected components (SCCs) of $\to$, then set $g(s) := h(C)$ if $s \in C$.*

We now lift this idea to the case of (restricted) reachability invariants.

Definition 7. *Given an SSTS $(S, \to, \trianglelefteq)$, an accepting state predicate ϕ, and a corresponding restricted reachability invariant $(I, \trianglelefteq)$, a* restricted topological numbering *of $(S, \to, \trianglelefteq)$ is a function f with an integer range such that:*

1. *For all $s, t' \in I$ and $t \in S$, if $s \to t$, and $t \trianglelefteq t'$, then $f(s) \geq f(t')$.*
2. *For all $s, t' \in I$ and $t \in S$, if $s \to t$, $t \trianglelefteq t'$, and $\phi(s)$, then $f(s) > f(t')$.*

Moreover, let $\rightsquigarrow_{\trianglelefteq}$ be the restriction of $\to_{\trianglelefteq}$ to I, i.e. the transition system such that $s \rightsquigarrow_{\trianglelefteq} t'$ iff $s, t' \in I$ and there exists a t such that $s \to t$ and $t \trianglelefteq t'$.

[5] All proofs are omitted for brevity and can be found in the appendix.

Now, f is clearly a topological numbering for $\rightsquigarrow_{\trianglelefteq}$. Thus $\rightsquigarrow_{\trianglelefteq}$ is free of accepting cycles. Additionally, the transition system $\rightsquigarrow_{\trianglelefteq}$ trivially simulates $\rightarrow_{\trianglelefteq}$ with $s \sqsubseteq s'$ iff $s' = s$ and $s \in I$. Therefore, any accepting cycle $s \rightarrow_{\trianglelefteq}^{+} s$ in $\rightarrow_{\trianglelefteq}$ with $s \in I$ and $\phi(s)$ yields an accepting cycle $s \rightsquigarrow_{\trianglelefteq}^{+} s$. Hence $\rightarrow_{\trianglelefteq}$ is free of accepting cycles.

From this, we conclude our main theorem that allows one to certify absence of Büchi runs in a transition system $\rightarrow$.

Theorem 3. *Let f be a restricted topological numbering of $(S, \rightarrow, \preceq)$ for a compatible predicate ϕ and a finite restricted reachability invariant $(I, \trianglelefteq)$. Then, for any initial state $s_0 \in S$ with $s_0 \trianglelefteq t_0$ for $t_0 \in I$, there is no Büchi run from s_0.*

In practice, a certificate for $\rightarrow$ can now be given as a finite reachability invariant I as described above, and a corresponding restricted topological numbering f. Both properties can be checked locally for each individual state in I.

3.4 Instantiation for Timed Automata

We now want to instantiate this abstract certification framework for the concrete case of TBA. Our goal is to certify that the zone graph $\Rightarrow$ does not contain any Büchi runs. As the zone graph is complete, this implies that the underlying TBA is empty. Thus we set $\rightarrow := \Rightarrow$. Subsumptions in the zone graph shall correspond to the self-simulation relation of the SSTS. Hence we define $\preceq$ such that $(q, Z) \preceq (q', Z')$ iff $q' = q$ and $Z \subseteq Z'$.

To certify unreachability, it is sufficient to consider arbitrary subsumptions in the zone graph, i.e. $\trianglelefteq := \preceq$ [37]. In other words it is sufficient to check that the given certificate I is a reachability invariant for $(S, \rightarrow, \preceq)$. We have not yet given the set of states S. Abstractly, S is simply the set of non-empty states, i.e. $S := \{(q, Z) \mid Z \neq \emptyset\}$. If it was allowed to reach empty zones, then soundness of the zone graph would not be given. In practice, the certifier needs to be able to compute $\Rightarrow$ effectively, typically using the DBM representation of zones. To this end one wants to add the assumption on states that all DBMs are in canonical form. One needs to ensure that states are split according to ϕ, i.e. $\forall (q, Z) \in S.\, Z \subseteq \Phi(q) \vee Z \cap \Phi(q) = \emptyset$ where $\Phi(q) = \{v \mid \phi(q, v)\}$. This is trivial for commonly used properties that concern only the finite state part.

Following these considerations, we propose the following certifier for the emptiness of TBA. A certificate C is a set of triplets (q, Z, i) where q is a discrete state, Z is a corresponding zone, and i is the topological number for (q, Z). The certifier runs Algorithm 1 on this certificate. The algorithm extends the one by Wimmer and Mutius [37] with the topological numbers for liveness checking.

Theorem 4. *If* Büchi-Emptiness(ϕ, C, q_0) *accepts the certificate, then $\Rightarrow_{DBM}$ has no Büchi run for ϕ. Consequently, the underlying TBA is empty.*

The proof constructs a suitable $\trianglelefteq$ such that $(q, Z) \trianglelefteq (q, Z')$ if $Z \subseteq Z'$ and $(q, Z', k) \in C$, where k is selected to be minimal. Setting $I := \{(q, Z) \mid \exists i.\, (q, Z, i) \in C)\}$ and $f(q, Z) := \mathsf{min}\{i \mid (q, Z, i) \in C\}$, Theorem 3 can be applied.

Algorithm 1 Certifier for the emptiness of TBA

```
1: procedure Büchi-Emptiness(φ, C, q_0)
2:     for all (q, Z, i) ∈ C do                          ▷ All DBMs are well-formed
3:         if Z = ∅ ∨ Z is not canonical
4:             then reject certificate
5:     if ∄(q_0, Z_0, i) ∈ C. {0} ⊆ Z_0                  ▷ The initial state is covered
6:         then reject certificate
7:     for all (q, Z, i) ∈ C do                          ▷ The certificate is:
8:         for all (q_1, Z_1) s.t. (q, Z) ⇒_DBM (q_1, Z_1) do
9:             if (∄(q_1, Z'_1, j) ∈ C. Z_1 ⊆ Z'_1       ▷ an invariant,
                   ∧ (φ(q) ⟶ i > j) ∧ i ≥ j)             ▷ and a topological numbering
10:                then reject certificate
11:    accept certificate
```

The algorithm inherits several beneficial properties from [37]. First, it can easily be parallelized. Most importantly however, the certifier does not need to compute an abstraction operation α. Suppose the model checker starts with a state $(q_0, \{\mathbf{0}\})$ and explores the transition $(q_0, \{\mathbf{0}\}) \Rightarrow (q_1, Z_1)$. The model checker could then abstract zone Z_1 to $\alpha(Z_1)$, and explore more edges from $(q_1, \alpha(Z_1))$, e.g. $(q_1, \alpha(Z_1)) \Rightarrow (q_2, Z_2)$. The certificate just needs to include $(q_0, \{\mathbf{0}\})$, $(q_1, \alpha(Z_1))$, and $(q_2, \alpha(Z_2))$, and the certificate checker just needs to check the following inclusions: $\{\mathbf{0}\} \subseteq \{\mathbf{0}\}$, $Z_1 \subseteq \alpha(Z_1)$, and $Z_2 \subseteq \alpha(Z_2)$. The checker does not need to compute α as $\alpha(Z_1)$ and $\alpha(Z_2)$ are part of the certificate.

It is rather easy to see that these certificates are also complete for timed automata. For any finite abstraction α, the abstracted zone graph $\Rightarrow_\alpha$ is finite and complete. Thus, for a starting state $(q_0, \{\mathbf{0}\})$ the set

$$I := \{(q, Z) \mid (q_0, \{\mathbf{0}\}) \Rightarrow_\alpha^* (q, Z)\}$$

is a trivial finite reachability invariant that can be computed effectively for common abstractions α. Moreover, if the underlying TBA is empty, then $\Rightarrow_\alpha$ cannot contain a Büchi run either, since the abstract zone graph is complete. Because $\Rightarrow_\alpha$ is finite, this means it cannot contain a cycle through ϕ. Hence a forward numbering of I can be given by computing the strongly connected components of I. However, this type of certificate is not of practical interest as subsumptions are not considered. How certificates can be obtained for model checking algorithms that make use of subsumption is the topic of Section 5.1.

4 Incorporating Advanced Abstraction Techniques

We have already discussed that the techniques that were presented above are in principle agnostic to the concrete abstraction α used. This, however, is only true for standard verification algorithms for T(B)A that use zone inclusion $Z \subseteq Z'$ as a simulation relation on the abstract zone graph. There is also the noteworthy abstraction $\alpha_{\preceq LU}$ [5], which is the coarsest zone abstraction that can be defined

from clock bounds L, U [18]. Herbreteau et al. have shown that even though $\alpha_{\preceq LU}(Z)$ is usually not a zone, it can be checked whether $Z \subseteq \alpha_{\preceq LU}(Z')$ directly from the DBM representation of Z and Z', without computing $\alpha_{\preceq LU}(Z')$ [18]. Hence, one can use $\alpha_{\preceq LU}$-subsumption over zones, $Z \subseteq \alpha_{\preceq LU}(Z')$, instead of standard inclusion $Z \subseteq Z'$ to explore fewer symbolic states. This technique can also be integrated with our certification approach. This time, we will need more knowledge about the concrete abstraction α, however.

We first describe the concept of time-abstract simulations, on which the definition of $\alpha_{\preceq LU}$ is based.

Definition 8. *A time-abstract simulation between clock valuations is a quasi-order $\preceq$ such that if $v \preceq v'$ and $(q, v) \to_{\delta,t} (q_1, v_1)$ then there exist δ' and v_1' such that $(q, v') \to_{\delta',t} (q_1, v_1') \wedge v_1 \preceq v_1'$.*

Behrmann et al. defined the simulation $\preceq_{LU}$ based on the clock bounds L and U, and showed that it is a time-abstract simulation [5] (in fact one can show that $\preceq_{LU}$ is even a simulation, i.e. $\delta' = \delta$). For any $\preceq$, one can define the corresponding abstraction $\alpha_{\preceq}(Z) = \{v \mid \exists v' \in Z. v \preceq v'\}$. This yields a sound and complete abstraction for any time-abstract simulation $\preceq$ [5]. Observe that $\alpha_{\preceq}(Z)$ is the set of all valuations that are simulated by a valuation in Z w.r.t. $\preceq$. As a result, every sequence of transitions feasible from $\alpha_{\preceq}(Z)$ is also feasible from Z (although with different delays).

The implicit abstraction technique based on the subsumption check $Z \subseteq \alpha_{\preceq}(Z')$ is compatible with our certification approach for any $\alpha_{\preceq}$ for which $\preceq$ is a time-abstract simulation, and in particular $\alpha_{\preceq LU}$. Actually, we are still able to use algorithm Büchi-Emptiness with the only modification that the condition $Z_1 \subseteq Z_1'$ is replaced with $Z_1 \subseteq \alpha_{\preceq}(Z_1')$. We will justify this by showing that if the algorithm accepts the certificate, then it represents a restricted reachability invariant with a suitable topological numbering for $\Rightarrow_{\alpha_{\preceq}}$. This means that $\Rightarrow_{\alpha_{\preceq}}$ does not have a Büchi run (Theorem 3), which, as $\alpha_{\preceq}$ is a complete abstraction, implies that the underlying TBA does not have a Büchi run either.

We first prove the following monotonicity property (which can be seen as a generalization of Lemma 4 in the work of Herbreteau et al. [18]).

Proposition 4. *Let $\preceq$ be a time-abstract simulation. If $\alpha_{\preceq}(W) \subseteq \alpha_{\preceq}(W')$, $(q, W) \Rightarrow^t (q_1, W_1)$, and $(q, W') \Rightarrow^t (q_1, W_1')$, then $\alpha_{\preceq}(W_1) \subseteq \alpha_{\preceq}(W_1')$.*

Reminding ourselves that $\alpha_{\preceq}$ is idempotent, if follows that if $(q, W) \Rightarrow^t (q_1, W_1)$ and $(q, \alpha_{\preceq}(W)) \Rightarrow^t (q_1, W_1')$ for some states q, q_1, and sets of valuations W, W_1, and W_1', then $\alpha_{\preceq}(W_1) = \alpha_{\preceq}(W_1')$. In other words, $\Rightarrow$ simulates $\Rightarrow_{\alpha_{\preceq}}$ for $\sqsupseteq$ defined as $(q, W) \sqsupseteq (q, Z) \longleftrightarrow W = \alpha_{\preceq}(Z)$.

Now, we show that the conditions of Definitions 4 and 7 can be transferred along this simulation.

Theorem 5. *Assume that the following conditions hold:*

1. *For all states q, and zones Z, Z', Z'', if $(q, Z) \trianglelefteq (q, Z')$, then $Z \subseteq \alpha_{\preceq}(Z')$. Moreover, if $\alpha_{\preceq}(Z) = \alpha_{\preceq}(Z')$ and $(q, Z) \trianglelefteq (q, Z'')$, then $(q, Z') \trianglelefteq (q, Z'')$.*

2. *For all q, Z, if $\phi((q, \alpha_{\preceq}(Z)))$, then $\phi((q, Z))$.*
3. *$(I, \trianglelefteq)$ satisfies condition (1) of Definition 4 for $\Rightarrow$.*
4. *f is a restricted topological numbering for $\Rightarrow$, $(I, \trianglelefteq)$, and ϕ.*

Let $(q, W) \trianglelefteq' (q, W') \longleftrightarrow \exists Z, Z'. W = \alpha_{\preceq}(Z) \wedge W' = \alpha_{\preceq}(Z') \wedge (q, Z) \trianglelefteq (q, Z')$, $I' := \{s' \mid \exists s \in I. s \sqsubseteq s'\}$ and $f'(s') := Min\{f(s) \mid s \in I \wedge s \sqsubseteq s'\}$. Then

1. *$(I', \trianglelefteq')$ is a restricted reachability invariant for $(\Rightarrow_{\alpha_{\preceq}}, \subseteq)$.*
2. *f' is a restricted topological numbering for $\Rightarrow_{\alpha_{\preceq}}$, $(I', \trianglelefteq')$, and ϕ.*

Algorithm Büchi-Emptiness ensures that there exist an invariant $(I, \trianglelefteq)$ and a numbering f that fulfill the conditions of Theorem 5 for $\Rightarrow$ (as indicated after Theorem 4). Thus, if the algorithm accepts the certificate, there is a restricted reachability invariant $(I', \trianglelefteq')$ with a corresponding topological numbering f' for $\Rightarrow_{\alpha_{\preceq}}$. Hence $\Rightarrow_{\alpha_{\preceq}}$ does not have a Büchi run.

5 Evaluation

In this section, we first give a brief description of the model checking algorithms we consider and describe how certificates can be extracted from them. Then, we outline the general architecture of our certification tool chain, and finally we present some experiments on standard TA models.

5.1 Extracting Certificates From Model Checkers

We consider the two state-of-the-art algorithms for checking Büchi emptiness for TA: the NDFS-based algorithm by Laarman et al. [22] and the iterative SCC-based algorithm by Herbreteau et al. [16]. Both algorithms can be applied to any abstracted zone graph $\Rightarrow_\alpha$ for a finite, sound and complete abstraction α. As was noted by Herbreteau et al. [16], they also have in common that their correctness can be justified on the basis that they both compute subsumption graphs that are *liveness compatible*, in the sense that they do not contain any cycle with an accepting node and a subsumption edge.

Considering NDFS for TA from [22] more closely, it prunes the search space by using subsumption in certain safe places. In particular, the outer (blue) search is pruned when it reaches a state s that is subsumed by a state on which the inner (red) search has been called, i.e. $s \sqsubseteq t$ and t is red. In order to generate a liveness-compatible subsumption graph, the blue search exports all the states which are not subsumed along with their $\rightarrow$-successors. Moreover, the algorithm exports $\rightsquigarrow$-edges as soon as the pruning by subsumption is applied.

The iterative algorithm from [16] interleaves reachability analysis and SCC decompositions. The reachability analysis computes a subsumption graph with maximal subsumption: a subsumption edge $s \rightsquigarrow t'$ is added whenever a new state t is visited from s, and t is subsumed by some visited state t'. The resulting graph $\rightarrow \cup \rightsquigarrow$ is a subsumption graph that preserves state reachability, but that may not be liveness compatible. Therefore, an SCC decomposition is run,

and all subsumption edges from SCCs that contain both an accepting node and a subsumption edge are removed. States which are not subsumed anymore are re-explored in the next iteration of the main loop. Upon termination, the subsumption graph $\rightarrow \cup \rightsquigarrow$ is liveness compatible.

Both algorithms compute liveness compatible subsumption graphs. In order to obtain a certificate we run one extra SCC decomposition of the graph with $\rightarrow \cup \rightsquigarrow$-edges from which we compute a topological ordering.

5.2 General Architecture

Our certifier is implemented as an extension of the tool Munta [34], which has been fully verified in Isabelle/HOL [36,37]. Figure 3 depicts the architecture of our tool chain to certify the emptiness of a given TBA. The model (a TBA) and the acceptance property are given in the input format of Munta. For the model checker in the middle, we used Imitator and TChecker. In a first step, the Munta model is translated to an input model for the model checker. The model checker decides whether the given TBA is empty. If not, then either the model checker's answer is correct or it has found a spurious counterexample; in both cases no certificate can be extracted. Otherwise, the model checker emits a certificate consisting of a number of symbolic states and the set of edges in the subsumption graph. The latter can either include proper transitions ($\rightarrow$) and subsumptions ($\rightsquigarrow$) (this is done for Imitator with NDFS and subsumption), or the edges that merge these two types ($\rightsquigarrow$') (which is done for TChecker and for Imitator with state merging enabled, see Section 5.3). In either case, in the next step where the certificate is translated to Munta's binary input format for certificates, the SCC numbers (c.f. Proposition 3) are re-computed blindly from these edges. This step additionally makes use of a renaming dictionary to map from human readable labels for states, actions, etc., to natural numbers.

Finally, the TBA model, the translated certificate, and the renaming are given to Munta. If it accepts the certificate, then there is an Isabelle/HOL theorem that guarantees that the given TBA is indeed empty. If the certificate is rejected,

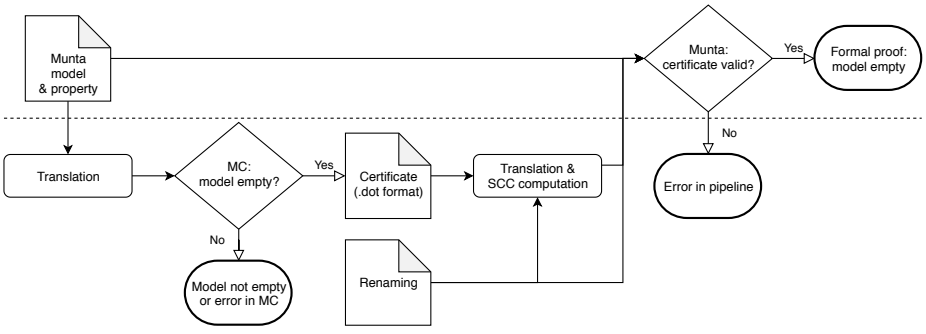


Fig. 3: Workflow of the certifier pipeline. The dashed line is the trust boundary. If the correct model is given, then the answer on the right can be trusted.

any of the steps in the tool chain could have failed. Note that the basis of trust is minimal. One just needs to ensure that the model represents what one has in mind, and to trust the correctness of Munta. To trust Munta, one essentially needs to trust its TBA semantics, which is less than 200 lines long, some core parts of Isabelle/HOL, and an SML compiler (MLton in our case). For details, we refer the interested reader to previous publications on Munta [34,37].

5.3 Experiments

We have evaluated our approach on the TBA models that were also used by Herbreteau et al. [16]. These are inspired by standard TA benchmarks, and all consist of the product of a TA model and an additional Büchi automaton that encodes the complement of the language of a given LTL formula that one wants to check. Details are given by Herbreteau et al. [16].

For Imitator we tried two methods: NDFS with subsumption and reachability analyis with merging [3]. Imitator does not apply abstractions (since it was designed for parametric TA), so the full zone graph is often infinite and most NDFS runs fail. The one that succeeds generates a valid certificate. *Merging* tries to reduce the number of zones, by computing the exact convex hull of zones. This creates new zones that could subsume several existing ones, and often yields a finite zone graph. The certificate produced by merging is always a reachability invariant but not necessarily a subsumption graph. Merging may introduce spurious cycles, in which case the certificate is not liveness compatible; these cases are caught by the Munta certifier. If there are no (spurious) accepting cycles, we obtain a valid and quite small certificate. Note that the generalization from subsumption graphs to our certificates is crucial to allow for merging.

Table 1 summarizes our experimental results. TChecker was run with the algorithm from [16] and [22] and Imitator with the algorithm from [22], and with a reachability procedure with full merging. The *** entries indicate cases where Imitator did not terminate within 30s. The results show that the certifier accepts those certificates that we expect it to accept, but also rejects those that stem from subsumption graphs that are not liveness compatible. Moreover, the certifier was fully verified in Isabelle/HOL and still yields reasonable performance, certifying models with more than a 100k symbolic states in under 230s.

6 Conclusion

Starting from an abstract theory on self-simulating transition systems, we have presented an approach to extract certificates from state-of-the-art model checking algorithms (including state-of-the-art abstraction techniques) that decide emptiness of timed Büchi automata. The certificates prove that a given model is indeed Büchi empty. We have verified the theory and a checker for these certificates in Isabelle/HOL, using the tool Munta as a basis. We demonstrated that our approach is feasible by extracting certificates for some standard benchmark models from the tools TChecker and Imitator. We hope that our work can

Model	TChecker						Imitator					
	Iterative SCC			NDFS			Merge			NDFS		
CC1	✓	57	0.01	✓	3281	0.06	✓	58	0.01	***		
CC4	✓	195858	221.56	✓	32575	7.75	***			***		
CC5	✓	65639	30.63	✓	143057	218.98	***			***		
FD1	✓	214	0.02	✓	677	0.03	✗	294	0.02	✓	1518	0.11
FI1	✓	65	0.01	✓	71	0.00	✓	136	0.00	***		
FI2	✓	314	0.01	✓	344	0.01	✓	589	0.01	***		
FI4	✓	204	0.00	✓	224	0.01	✓	793	0.01	***		
FI5	✓	3091	0.13	✓	2392	0.09	✗	863	0.03	***		

Table 1: Benchmark results on a 2017 MacBook Pro with 16 GB RAM and a Quad-Core Intel Core i7 CPU at 3.1 GHz. For each algorithm, we show whether the certificate was accepted, the number of DBMs in the certificate, and the time for certificate checking on a single core in seconds.

help to increase confidence in safety-critical systems that have been verified with timed automata model checkers. Furthermore, we envision that our tool could help in the organization of future competitions for such model checkers.

To close, we want to illuminate some potential future directions of research. First, one is usually not only interested in the emptiness of TBA per se, but more generally in the question if a TA model satisfies some LTL requirements. Thus, our tool would ideally be combined with a verified translation from LTL formulas to Büchi automata or with a certifier for such a construction. The former has been realized by the CAVA project [13], while an avenue towards the latter is opened by the recent work of Seidl et al. [10].

Second, Herbreteau et al. have developed a technique of computing abstractions for TA on the fly, starting from very coarse abstractions and refining them as needed [17]. It seems that our approach is in principle compatible with this technique when augmenting certificates with additional information on the computed abstractions, whose validity would have to be checked by the certifier.

Third, one could attempt to reduce the size of the certificates. In one approach, reachability certificates have been compressed after model checking (c.f. [37]). On the other hand, model checking algorithms could speculate that the given TBA is empty, and use this fact to use additional subsumptions to reduce the search space, while risking to miss accepting runs. However, given the certification step afterwards, this is of no concern. For instance, one could remove the red search from the NDFS algorithm, and use subsumption on blue nodes instead of red nodes, as a quick pre-check. If the result passes the certifier, we are done.

Finally, as our theory is not specific to timed automata per se, it could be interesting to find other application domains for this approach to certification. In light of the large body of existing work on well-structured transition systems, this looks particularly promising as any such system is also self-simulating.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
2. André, É., Fribourg, L., Kühne, U., Soulat, R.: Imitator 2.5: A tool for analyzing robustness in scheduling problems. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012: Formal Methods. pp. 33–36. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
3. André, É., Soulat, R.: Synthesis of timing parameters satisfying safety properties. In: RP. Lecture Notes in Computer Science, vol. 6945, pp. 31–44. Springer (2011)
4. Behrmann, G., Bouyer, P., Fleury, E., Larsen, K.G.: Static guard analysis in timed automata verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. pp. 254–270. Springer Berlin Heidelberg (2003)
5. Behrmann, G., Bouyer, P., Larsen, K.G., Pelanek, R.: Lower and upper bounds in zone-based abstractions of timed automata. International Journal on Software Tools for Technology Transfer (STTT) **8**(3), 204–215 (2006)
6. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets: Advances in Petri Nets. LNCS, vol. 3908, pp. 87–124. Springer (2004). https://doi.org/10.1007/978-3-540-27755-2_3
7. Blanchette, J.C., Fleury, M., Lammich, P., Weidenbach, C.: A verified SAT solver framework with learn, forget, restart, and incrementality. J. Autom. Reasoning **61**(1-4), 333–365 (2018). https://doi.org/10.1007/s10817-018-9455-7
8. Blom, S., Darabi, S., Huisman, M., Oortwijn, W.: The vercors tool set: Verification of parallel and concurrent software. In: IFM. Lecture Notes in Computer Science, vol. 10510, pp. 102–110. Springer (2017)
9. Brunner, J., Lammich, P.: Formal verification of an executable LTL model checker with partial order reduction. Journal of Automated Reasoning **60**(1), 3–21 (2018)
10. Brunner, J., Seidl, B., Sickert, S.: A verified and compositional translation of LTL to deterministic Rabin automata. In: Harrison, J., O'Leary, J., Tolmach, A. (eds.) ITP 2019, September 9-12, 2019, Portland, OR, USA. LIPIcs, vol. 141, pp. 11:1–11:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.ITP.2019.11
11. Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: TACAS 1998. Lecture Notes in Computer Science, vol. 1384, pp. 313–329. Springer (1998)
12. Dill, D.L.: Timing Assumptions and Verification of Finite-State Concurrent Systems. In: Sifakis, J. (ed.) Automatic Verification Methods for Finite State Systems, International Workshop. LNCS, vol. 407, pp. 197–212. Springer (1989)
13. Esparza, J., Lammich, P., Neumann, R., Nipkow, T., Schimpf, A., Smaus, J.G.: A fully verified executable LTL model checker. In: Proc. of CAV'13. LNCS, vol. 8044, pp. 463–478. Springer (2013)
14. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science **256**(1), 63 – 92 (2001). https://doi.org/10.1016/S0304-3975(00)00102-X, iSS
15. Griggio, A., Roveri, M., Tonetta, S.: Certifying proofs for LTL model checking. 2018 Formal Methods in Computer Aided Design (FMCAD) pp. 1–9 (2018)
16. Herbreteau, F., Srivathsan, B., Tran, T.T., Walukiewicz, I.: Why liveness for timed automata is hard, and what we can do about it. In: Lal, A., Akshay, S., Saurabh, S., Sen, S. (eds.) FSTTCS. LIPIcs, vol. 65, pp. 48:1–48:14. Schloss Dagstuhl (2016)

17. Herbreteau, F., Srivathsan, B., Walukiewicz, I.: Lazy abstractions for timed automata. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 990–1005. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
18. Herbreteau, F., Srivathsan, B., Walukiewicz, I.: Better abstractions for timed automata. Information and Computation **251**, 67–90 (2016)
19. Herbreteau, F., Point, G.: TChecker (2019), `https://github.com/fredher/tchecker`
20. Heule, M., Hunt, W., Kaufmann, M., Wetzler, N.: Efficient, verified checking of propositional proofs. In: Ayala-Rincón, M., Muñoz, C.A. (eds.) Interactive Theorem Proving. pp. 269–284. Springer International Publishing, Cham (2017)
21. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: High-performance language-independent model checking. In: TACAS. Lecture Notes in Computer Science, vol. 9035, pp. 692–707. Springer (2015)
22. Laarman, A., Olesen, M.C., Dalsgaard, A.E., Larsen, K.G., van de Pol, J.: Multi-core emptiness checking of timed Büchi automata using inclusion abstraction. In: Sharygina, N., Veith, H. (eds.) CAV. pp. 968–983. Springer Berlin Heidelberg (2013)
23. Lammich, P.: Efficient verified (UN)SAT certificate checking. In: de Moura, L. (ed.) Automated Deduction – CADE 26. pp. 237–254. Springer International Publishing, Cham (2017)
24. Larsen, G.K., Pettersson, P., Yi, W.: Uppaal in a nutshell. Software Tools for Technology Transfer **1**(1), 134–152 (1997)
25. Leino, K.R.M.: Developing verified programs with dafny. In: ICSE. pp. 1488–1490. IEEE Computer Society (2013)
26. Li, G.: Checking timed Büchi automata emptiness using LU-abstractions. In: FORMATS. LNCS, vol. 5813, pp. 228–242 (2009)
27. Namjoshi, K.S.: Certifying model checkers. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. pp. 2–13. Springer Berlin Heidelberg (2001)
28. Nguyen, H.G., Petrucci, L., van de Pol, J.: Layered and collecting NDFS with subsumption for parametric timed automata. In: ICECCS. pp. 1–9. IEEE Computer Society (2018)
29. Nipkow, T., Lawrence C. Paulson, Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002). https://doi.org/10.1007/3-540-45949-9
30. Oortwijn, W., Huisman, M., Joosten, S.J.C., van de Pol, J.: Automated verification of parallel nested DFS. In: Biere, A., Parker, D. (eds.) TACAS 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12078, pp. 247–265. Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_14
31. van de Pol, J.: Automated verification of Nested DFS. In: FMICS. Lecture Notes in Computer Science, vol. 9128, pp. 181–197. Springer (2015)
32. Tripakis, S.: Checking timed Büchi emptiness on simulation graphs. ACM Trans. on Computational Logic **10**(3) (2009)
33. Tripakis, S., Yovine, S., Bouajjani, A.: Checking timed Büchi automata emptiness efficiently. Formal Methods in System Design **26**(3), 267–292 (2005)
34. Wimmer, S.: Munta: A verified model checker for timed automata. In: André, É., Stoelinga, M. (eds.) FORMATS 2019, Proceedings. Lecture Notes in Computer Science, vol. 11750, pp. 236–243. Springer (2019). https://doi.org/10.1007/978-3-030-29662-9_14
35. Wimmer, S., Herbreteau, F., van de Pol, J.: Certifying emptiness of timed büchi automata: Artifact (Jul 2020). https://doi.org/10.6084/m9.figshare.12620582.v1

36. Wimmer, S., Lammich, P.: Verified model checking of timed automata. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. pp. 61–78. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89960-2_4
37. Wimmer, S., von Mutius, J.: Verified certification of reachability checking for timed automata. In: Biere, A., Parker, D. (eds.) TACAS 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12078, pp. 425–443. Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_24

Appendix

The appendix restates all propositions of the paper together with their proofs. For coherence, some propositions are stated slightly more precisely than in the paper.

Proposition 1. *If $s_1 \to s_2 \to \ldots \to s_n\,(\to \ldots)$ is a path (run) and $s_1 \sqsubseteq t_1$, then there is a path (run) $t_1 \to' t_2 \to' \ldots \to' t_n\,(\to \ldots)$ with $s_i \sqsubseteq t_i$ for all i.*

Theorem 1. *Let I be an invariant and ϕ be a compatible predicate for an SSTS $(S, \to, \preceq)$. If $\forall s \in I.\ \neg\phi(s)$, then for all $s_0 \in S$ for which there exists an $s_0' \in I$ with $s_0 \preceq s_0'$, there is no accepting path $s_0 \to s_1 \to \ldots \to s_n$ with $\phi(s_n)$.*

Proof. Assume that there is a path $s_0 \to s_1 \to \ldots \to s_n$ with $\phi(s_n)$ in $(S, \to, \preceq)$. By the invariant simulation property (Theorem 2) we get a path $t_0 \to_\preceq t_1 \ldots \to_\preceq t_n$ with $s_n \preceq t_n$ and $t_n \in I$. Because ϕ is compatible, we have $\phi(t_n)$. However, this contradicts $t_n \in I$. □

We state this theorem again for $\preceq$ instead of $\trianglelefteq$ as we did above. The proofs are the same.

Theorem 2. *Let I be a reachability invariant for $(S, \to, \preceq)$. Then, for all $s_1 \preceq t_1$ with $t_1 \in I$, if $s_1 \to s_2 \to \ldots \to s_n$ is a path, then there is a path $t_1 \to_\preceq t_2 \ldots \to_\preceq t_n$ such that $s_i \preceq t_i$ and $t_i \in I$ for all i. Similarly, if $s_1 \to s_2 \to \ldots$ is a run, then there is a run $t_1 \to_\preceq t_2 \to_\preceq \ldots$ such that $s_i \preceq t_i$ and $t_i \in I$ for all i.*

Proof. By induction (coinduction) on the inductive (coinductive) definition of $s_1 \to s_2 \to \ldots \to s_n$ $(s_1 \to_\preceq s_2 \to_\preceq \ldots)$. □

Proposition 2. *Let f be a topological numbering of $\to$. If there exists a path of the form $s \to s_1 \to s_2 \to \ldots \to s$, then $\neg\phi(s)$.*

Proof. For the sake of contradiction, assume $\phi(s)$. From $s \to s_1$, we have $f(s) > f(s_1)$. Moreover, from $s_1 \to s_2 \to \ldots \to s$, we have $f(s_1) \geq f(s)$ by induction on the path definition. Together, we arrive at the contradiction $f(s) > f(s)$. □

Proposition 3. *If there is no path $s \to s_1 \to s_2 \to \ldots \to s$ with $\phi(s)$ in $\to$, then the following are topological numberings for $\to$.*

1. *The number of accepting states that are reachable from a node: $f(s) := |\{x \mid s \to^* x \wedge \phi(x)\}|$ (assuming $\{x \mid s \to^* x \wedge \phi(x)\}$ is finite for any s).*
2. *If h is a topological numbering (in the classical sense) of the strongly connected components (SCCs) of $\to$, then set $g(s) := h(C)$ if $s \in C$.*

Proof. 1. Clearly, if $s \to t$, then $f(s) \geq f(t)$. Suppose $s \to t$, $\phi(s)$, and $\{x \mid s \to^* x \wedge \phi(x)\} \subseteq \{x \mid t \to^* x \wedge \phi(x)\}$. Then, we have $t \to^* s$ and thus $s \to^+ s$. Because of $\phi(s)$, this contradicts the assumption and hence $f(s) > f(t)$.

2. As h is a topological numbering (in the classical sense, i.e. without condition (2) of definition 6) of the SCCs of $\rightarrow$, we have $g(s) \geq g(t)$ if $s \rightarrow t$. Moreover, any state s with $\phi(s)$ has to form its own trivial SCC as otherwise the SCC of s would form an accepting cycle. Thus $g(s) > g(t)$ if $\phi(s)$ and $s \rightarrow t$.

□

Theorem 3. *Let f be a restricted topological numbering of $(S, \rightarrow, \preceq)$ for a compatible ϕ and a restricted reachability invariant $(I, \trianglelefteq)$ such that I is finite. Then, for any initial state $s_0 \in S$ such that there exists a $t_0 \in I$ with $s_0 \trianglelefteq t_0$, there is no Büchi run from s_0 in $\rightarrow$.*

Proof. Working towards a contradiction, suppose that there is an accepting Büchi run $s_0 \rightarrow s_1 \rightarrow \ldots$. By proposition 2, there exists a run $t_0 \rightarrow t_1 \rightarrow \ldots$ such that $s_i \trianglelefteq t_i$ and $t_i \in I$ for all i. As there are infinitely many s_i with $\phi(s_i)$, and ϕ is compatible with $(S, \rightarrow, \preceq)$, there are also infinitely many t_i with $\phi(t_i)$. Because I is finite, accepting states have to repeat eventually and there is an accepting lasso, i.e. there exists an accepting t_k (with $\phi(t_k)$) such that $t_0 \rightarrow^* t_k$ and $t_k \rightarrow^+ t_k$. This contradicts proposition 2. □

Theorem 4. *If* Büchi-Emptiness(ϕ, C, q_0) *accepts the certificate, then $\Rightarrow_{DBM}$ has no Büchi run for ϕ. Consequently, the underlying TBA is empty.*

The proof will make clear why we needed to define the concept of a restricted reachability invariant in the first place: to only select certain subsumptions that go downward as far as possible.

Proof. We define $I := \{(q, Z) \mid \exists i.\, (q, Z, i) \in C)\}$. The whole proof hinges on the fact that we are able to define a suitable $\trianglelefteq$ such that the certificate can form a restricted reachability invariant $(I, \trianglelefteq)$ for $(S, \rightarrow, \preceq)$, and in addition we can define a suitable numbering f which is a restricted topological numbering for $(S, \rightarrow, \preceq)$ with respect to $\rightarrow_{\trianglelefteq}$. The main idea for this is to only consider subsumptions that go downward as far as possible. That is, we have $(q, Z) \trianglelefteq (q', Z')$ iff $q' = q$, $Z \subseteq Z'$, and if there exists a k^* such that $(q, Z', k^*) \in C$ and k^* is minimal among all k for which there exists a Z'' with $Z \subseteq Z''$ and $(q, Z'', k) \in C$. Lines 7-10 of the algorithm ensure that there exists one such triplet (q, Z'', k) for each $(q, Z) \in I$. Thus the first condition of definition 4 is verified. The condition $I \subseteq S$ is ensured by lines 2-4. The second condition, finally, is trivially met by the definition of $\trianglelefteq$.

Now let $f(q, Z)$ be the minimal i such that $(q, Z, i) \in C$. We need to prove that the conditions of definition 7 are met. Suppose $(q, Z) \in I$, $(q, Z) \Rightarrow (q_1, Z_1)$ and $(q_1, Z_1) \trianglelefteq (q_1, Z_1')$. We know that $(q, Z, f(q, Z)) \in C$ by the definition of f. Lines 7-10 of the algorithm ensure that there exists a triplet $(q_1, Z', k) \in C$ such that $Z \subseteq Z'$, $f(q, Z) \geq k$, and $\phi(q) \longrightarrow f(q, Z) > k$. Then, by the definition of $\trianglelefteq$, there exists a k^* such that $(q_1, Z_1', k^*) \in C$ and $k \geq k^*$. Thus $k \geq f(q_1, Z_1')$ by the definition of f and both conditions of 7 are met.

Finally, together with the fact that $(q_0, \{\mathbf{0}\})$ is covered by C, which is ensured by lines 5-6, we can invoke theorem 3 to show that $\Rightarrow_{DBM}$ does not have a

Büchi run. Then, by bisimulation $\Rightarrow$ does not have a Büchi run eiher, and by completeness of $\Rightarrow$ the underlying TBA is empty as well. □

Proposition 4. *Let $\preceq$ be a time-abstract simulation. If $\alpha_\preceq(W) \subseteq \alpha_\preceq(W')$, $(q, W) \Rightarrow^t (q_1, W_1)$, and $(q, W') \Rightarrow^t (q_1, W_1')$, then $\alpha_\preceq(W_1) \subseteq \alpha_\preceq(W_1')$.*

Proof. Suppose $v \in W_1$ and $u \preceq v$. We need to show that there exists a $v' \in W_1'$ such that $u \preceq v'$. By the definition of $\Rightarrow^t$ there exists a δ and $u_0 \in W$ such that $(q, u_0) \rightarrow_{\delta,t} (q_1, v)$. Because $\preceq$ is reflexive and $\alpha_\preceq(W) \subseteq \alpha_\preceq(W')$ there is a $v_0 \in W'$ with $u_0 \preceq v_0$. As $\preceq$ is a time-abstract simulation, we can find a v' such that $(q, v_0) \rightarrow_{\delta,t} (q_1, v')$ and $v \preceq v'$. Thus we have $v' \in W_1'$ by definition of $\Rightarrow^t$ and $u \preceq v'$ by transitivity. □

In comparison to the main part of the paper (Sec. 4), we will first prove a more abstract version of Thm. 5 and then instantiate it for the concrete case of timed automata.

Theorem 5. *Assume that $\rightarrow$ simulates $\rightarrow'$ with $\sqsupseteq$ and that the following conditions hold:*

1. *$\trianglelefteq'$ simulates $\trianglelefteq$ with $\sqsubseteq$.*
2. *For all s, s', t, and t', if $s \sqsubseteq s'$, $t \sqsubseteq t'$ and $s' \trianglelefteq' t'$, then $s \trianglelefteq t$.*
3. *For all s and s', if $s \sqsubseteq s'$ then $\phi'(s') \longrightarrow \phi(s)$.*

Let $I' := \{s' \mid \exists s \in I.\, s \sqsubseteq s'\}$ and $f'(s') := Min\,\{f(s) \mid s \in I \wedge s \sqsubseteq s'\}$.

1. *Suppose that $(I, \trianglelefteq)$ verifies condition (1) of definition 4 for $\rightarrow$. Then $(I', \trianglelefteq')$ verifies conditions (1) of definition 4 for $\rightarrow'$.*
2. *Suppose that f verifies conditions (1) and (2) of definition 7 for $\rightarrow$, $\trianglelefteq$, I, and ϕ. Then f' verifies conditions (1) and (2) of definition 7 for $\rightarrow'$, $\trianglelefteq'$, I', and ϕ'.*

Proof. 1. Assume $s \in I$, $s \sqsubseteq s'$ and $s' \rightarrow' t'$. We need to show that there is an $r' \in I'$ with $t' \trianglelefteq' r'$. By simulation we can find a t such that $s \rightarrow t$ and $t \sqsubseteq t'$. Thus there is an $r \in I$ with $t \trianglelefteq r$. With (1), we know there is an r' with $t' \trianglelefteq' r'$ and $r \sqsubseteq r'$.

2. Assume $s \in I$, $s \sqsubseteq s'$, $s' \rightarrow' t'$, $t' \trianglelefteq' r'$, $r \in I$, and $r \sqsubseteq r'$. There is an $s_0 \in I$ such that $f'(s') = f(s_0)$ and $s_0 \sqsubseteq s'$. By simulation we can find a t such that $s_0 \rightarrow t$ and $t \sqsubseteq t'$. With (2), we have $t \trianglelefteq r$. Thus we have $f(s_0) \geq f(r)$ and $\phi(s_0) \longrightarrow f(s_0) > f(r)$. Moreover, $f'(r') \leq f(r)$ by definition of f'. Finally, with (3) and $f'(s') = f(s_0)$, we get $f'(s') \geq f'(r')$ and $\phi'(s') \longrightarrow f'(s') > f(r')$.

□

Instantiation of theorem 5 for timed automata. As mentioned above, we instantiate the theorem for $\rightarrow := \Rightarrow$, $\rightarrow' := \Rightarrow_{\alpha_\preceq}$, and $(q, Z) \sqsubseteq (q, W) \longleftrightarrow W = \alpha_\preceq(Z)$. To satisfy condition (3), any ϕ which is compatible with $\alpha_\preceq$ and the

instantiation $\phi' := \phi$ suffice. Let $\trianglelefteq^*$ be the subsumption relation defined in the proof of theorem 3. That is:

$$\begin{aligned} & (q, Z) \trianglelefteq^* (q, Z') \\ \longleftrightarrow\ & \exists k.\, Z \subseteq \alpha_{\preceq}(Z') \wedge (q, Z', k) \in C \\ & \quad \wedge (\forall (q, Z'', k') \in C.\, Z \subseteq \alpha_{\preceq}(Z'') \longrightarrow k' \geq k) \end{aligned}$$

We set[6]:

- $(q, Z) \trianglelefteq (q', W') \longleftrightarrow \exists Z'.\, W' = \alpha_{\preceq}(Z') \wedge (q, Z) \trianglelefteq^* (q', Z')$ and
- $(q, W) \trianglelefteq' (q', W')$
  $\longleftrightarrow \exists Z, Z'.\, W = \alpha_{\preceq}(Z) \wedge W' = \alpha_{\preceq}(Z') \wedge (q, Z) \trianglelefteq^* (q', Z')$.

Then condition (1) is trivially satisfied. Condition (2) can be verified by observing that $\trianglelefteq^*$ is constructed in a way such that it is deterministic, i.e. if $(q, Z) \trianglelefteq^* (q', Z_1)$ and $\alpha_{\preceq}(Z) = \alpha_{\preceq}(Z')$, then $(q, Z') \trianglelefteq^* (q', Z_1)$ (because $Z \subseteq \alpha_{\preceq}(Z_1) \longleftrightarrow Z' \subseteq \alpha_{\preceq}(Z_1)$ if $\alpha_{\preceq}(Z) = \alpha_{\preceq}(Z')$). Finally, it follows from proposition 4 that $\Rightarrow$ simulates $\Rightarrow_\alpha$ for $\sqsupseteq$ as outlined in Sec. 4.

[6] In the statement of Thm. 5 given in Sec. 4, we set $\trianglelefteq := \trianglelefteq^*$ for brevity.