The Power of Proofs: New Algorithms for Timed Automata Model Checking[⋆]

Peter Fontana and Rance Cleaveland

Department of Computer Science,
University of Maryland, College Park, MD 20742, USA

Abstract. This paper presents the first model-checking algorithm for an expressive modal mu-calculus over timed automata, $L_{\nu,\mu}^{rel,af}$, and reports performance results for an implementation. This mu-calculus contains extended time-modality operators and can express all of TCTL. Our algorithmic approach uses an "on-the-fly" strategy based on proof search as a means of ensuring high performance for both positive and negative answers to model-checking questions. In particular, a set of proof rules for solving model-checking problems are given and proved sound and complete; our algorithm then model-checks a property by constructing a proof (or showing none exists) using these rules. One noteworthy aspect of our technique is that we show that verification performance can be improved with *derived rules*, whose correctness can be inferred from the more primitive rules on which they are based. In this paper, we give the basic proof rules underlying our method, describe derived proof rules to improve performance, and we compare our implementation to UPPAAL.

1 Introduction

Timed automata are used to model real-time systems in which time is continuous and timing constraints may refer to elapsed time between system events [4]. The timed automata model provides a balance between expressiveness and tractability: a variety of different real-time systems can be captured in the formalism, and various properties, including safety (reachability) and liveness, can also be decided automatically for a given automaton [1, 2, 3].

To specify these properties, different logics have been devised. One popular logic, Timed Computation Tree Logic (TCTL) [3], extends the untimed Computation Tree Logic (CTL) [9] by adding time constraints to the modal operators. Other researchers explored timed extensions to the modal mu-calculus [12]. One such extension, called T_μ [18] extends the untimed modal mu-calculus with a single-step operator. Another extension, which we refer to as $L_{\nu,\mu}$ [21, 26, 27], extends the modal mu-calculus with separate time and action modal operators. This logic is sufficient for expressing some basic safety and liveness properties. However, it cannot express all of TCTL [14]. To address this, $L_{\nu,\mu}$ was extended with *relativization operators* by [7]; we denote this logic as $L_{\nu,\mu}^{rel}$. These additional

[⋆] Research supported by NSF grant CCF-0926194. The Appendix to this paper is available as a supplement on arXiv [16].

operators make the logic expressive enough to express all of TCTL [14]. (Bouyer et al. [7] included only greatest fixpoints, yielding L_ν, which they referred to as L_c; the least fixpoints in $L^{rel}_{\nu,\mu}$ not in L^{rel}_{ν} add expressive power [14].)

Over the model of timed automata, the model checking problem for $L_{\nu,\mu}$ is EXPTIME-complete [1]. Bouyer et al. [7] show that formulas using the relativization operators can be model-checked in EXPTIME. Hence, model checking $L^{rel}_{\nu,\mu}$ over timed automata is EXPTIME-complete. The same model-checking problem for TCTL over timed automata is PSPACE-complete [3].

While timed logics were being studied, tools and implementation algorithms were developed as well. Much of the development focused on handling subsets of properties specified in TCTL. A widely-used tool, UPPAAL [6], supports a fragment of TCTL, which includes many safety and liveness properties; other tools, including KRONOS [25], Synthia [20], and RED/REDLIB [23], have also been developed, some of which are able to model-check all of TCTL. Additionally, some tools were developed for timed modal-mu calculi. Two tools that can model check fragments of a timed mu-calculus include CMC [19], which can handle L_ν, and CWB-RT [13, 26, 27], which can check safety properties written in L_ν.

The contributions of this paper include the first algorithm, and an implementation, to model check $L^{rel,af}_{\nu,\mu}$. By definition, $L^{rel,af}_{\nu,\mu}$ consists of the so-called *alternation-free* formulas of $L^{rel}_{\nu,\mu}$ and is thus a superset of L^{rel}_{ν}. Assuming non-zeno and timelock-free automata, $L^{rel,af}_{\nu,\mu}$ is strong enough to express all of TCTL [14]. Our implementation extends the tool CWB-RT [13, 26, 27]. Implementation details of the model checker are discussed in Section 5; in Section 6, we give a demonstration of some models and properties that can be model checked by our tool as well as a performance comparison to UPPAAL.

CWB-RT is a proof-search model checker: it verifies properties by constructing a proof using a set of proof rules. These proof rules decompose the given goal (does the automaton satisfy a formula) into (smaller) subgoals. These proof search methods were used for the untimed modal mu-calculus in [10], explored in [21], and extended to the timed setting in [26, 27] in order to produce a fast on-the-fly model checker that can model check timed automata incrementally. The generated proofs not only give additional correctness information but also can be used as a mechanism to improve model-checking performance. We develop the additional proof rules to check the relativized operators, extending the proof rules used in [26, 27]. The additional rules are discussed in Section 3.

Furthermore, through select *derived* proof rules, we can enhance performance. These derived rules, together with a judicious use of *memoization*, yield dramatic performance improvements. We discuss the derived proof rules in Section 4.

2 Background

2.1 Timed Automata

This section defines the syntax of timed automata and sketches their semantics. The interested reader is referred to [2, 15] for a fuller account. To begin with, timed automata rely on *clock constraints*.

Definition 1 (Clock constraint $cc \in \Phi(CX)$). *Given a nonempty finite set of clocks $CX = \{x_1, x_2, \ldots, x_n\}$ and $d \in \mathbb{Z}^{\geq 0}$ (a non-negative integer), a* clock constraint *cc may be constructed using the following grammar:*

$$cc ::= x_i < d \mid x_i \leq d \mid x_i > d \mid x_i \geq d \mid cc \wedge cc$$

$\Phi(CX)$ is the set of all possible clock constraints over CX. We also use the following abbreviations: true (`tt`*) for $x_1 \geq 0$, false (*`ff`*) for $x_1 < 0$, and $x_i = d$ for $x_i \leq d \wedge x_i \geq d$.*

Timed automata may now be defined as follows.

Definition 2 (Timed automaton). *A* timed automaton *is a tuple $(L, l_0, \Sigma, CX, I, E)$, where:*

- *L is the finite set of* locations.
- *$l_0 \in L$ is the* initial location.
- *Σ is the finite set of* action symbols.
- *$CX = \{x_1, x_2, \ldots, x_n\}$ is the nonempty finite set of* clocks.
- *$I : L \longrightarrow \Phi(CX)$ maps each location l to a clock constraint, $I(l)$, referred to as the* invariant *of l.*
- *$E \subseteq L \times \Sigma \times \Phi(CX) \times 2^{CX} \times L$ is the set of* edges. *In an edge $e = (l, a, cc, \lambda, l')$ from l to l' with action a, $cc \in \Phi(CX)$ is the* guard *of e, and λ represents the set of clocks to* reset *to* 0.

The semantics of timed automata rely on *clock valuations*, which are functions $\nu : CX \longrightarrow \mathbb{R}^{\geq 0}$ ($\mathbb{R}^{\geq 0}$ is the set of non-negative real numbers); intuitively, $\nu(x_i)$ is the current time value of clock x_i. A timed automaton begins execution in its initial location with the initial clock valuation ν_0 assigning 0 to every clock. When the automaton is in a given clock location l with current clock valuation ν, two types of transitions can occur: time advances and action executions. During a time advance, the location stays the same and the clock valuation ν advances $\delta \in \mathbb{R}^{\geq 0}$ units to the valuation $\nu + \delta$, where $\nu + \delta$ is defined as $(\nu + \delta)(x_i) = \nu(x_i) + \delta$. For a time advance to be allowed, for all $0 \leq \delta' \leq \delta$, $\nu + \delta'$ must satisfy the invariant of location l. Due to convexity of clock constraints, it suffices to ensure that both ν and $\nu + \delta$ satisfy $I(l)$. An *action execution* of action a can occur when ν satisfies the guard for an edge leading from l to l', the edge is labeled by action a, and , the invariant of l' is satisfied after the clocks are reset as specified in the edge. In this case the location changes to l' and the clocks in λ are reset to 0. These intuitions can be formalized as a labeled transition system whose states consist of locations paired with clock valuations, each state notated as (l, ν). A *timed run* of the automaton is a sequence of transitions starting from the initial location and ν_0. On occasion, we also augment each timed automaton with a set of atomic propositions AP and a labeling function $M : L \longrightarrow 2^{AP}$ where $M(l)$ is the subset of propositions in AP that location l satisfies.

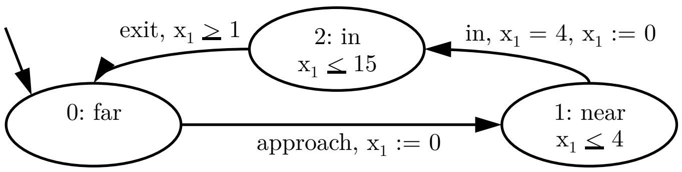

Fig. 1. Timed automaton of a train

Example 1 (Train timed automaton). The timed automaton in Figure 1 models a train component of the GRC (Generalized Railroad Crossing) protocol [17]. There are three locations: 0: far (initial), 1: near, and 2: in; and one clock x_1. Σ has the actions *approach*, *in*, and *exit*. Here, location 1: near has the invariant $x_1 \leq 4$ while 0: far has the vacuous invariant $\texttt{tt}$. The edge $(1\text{: near}, in, x_1 = 4, \{x_1\}, 2\text{: in})$ has action *in*, guard $x_1 = 4$, and resets x_1 to 0.

A sample timed run of this timed automaton is: $(0\text{: far}, x_1 = 0) \xrightarrow{5} (0\text{: far}, x_1 = 5) \xrightarrow{approach} (1\text{: near}, x_1 = 0) \xrightarrow{4} (1\text{: near}, x_1 = 4) \xrightarrow{in} (2\text{: in}, x_1 = 0) \xrightarrow{3} (2\text{: in}, x_1 = 3) \xrightarrow{2} (2\text{: in}, x_1 = 5) \xrightarrow{exit} (0\text{: far}, x_1 = 5) \ldots$

2.2 Timed Logic $L^{rel}_{\nu,\mu}$ and Modal Equation Systems (MES)

The following definition of $L^{rel}_{\nu,\mu}$ uses the modal-equation system (MES) format used in [11] for untimed systems and in [26, 27] for $L_{\nu,\mu}$.

Definition 3 ($L_{\nu,\mu}$, $L^{rel}_{\nu,\mu}$ basic formula syntax). *Let $CX = \{x_1, x_2, \ldots\}$ and $CX_f = \{z, z_1, \ldots\}$ be disjoint sets of clocks. Then the syntax of a $L_{\nu,\mu}$* basic formulas *is given by the following grammar:*

$$\begin{aligned}\psi ::= & p \mid \neg p \mid \texttt{tt} \mid \texttt{ff} \mid cc \mid Y \mid \psi \wedge \psi \mid \psi \vee \psi \mid \langle a \rangle(\psi) \\ & \mid [a](\psi) \mid \exists(\psi) \mid \forall(\psi) \mid z.(\psi)\end{aligned}$$

Here, $p \in AP$ is an atomic proposition, $cc \in \Phi(CX)$ is a clock constraint over clock set CX, $Y \in Var$ is a propositional variable (Var is the set of propositional variables), and $a \in \Sigma$ is an action. In formula $z.\psi$ z is a clock in CX_f; the $z.$ operator is often referred to as freeze quantification.

The relativized timed modal-mu calculus $L^{rel}_{\nu,\mu}$ syntax *replaces $\exists(\psi)$ and $\forall(\psi)$ with $\exists_{\psi_1}(\psi_2)$ and $\forall_{\psi_1}(\psi_2)$, where each ψ_1 and ψ_2 are basic formulas in $L^{rel}_{\nu,\mu}$.*

What follows is a sketch of the semantics; [7, 14] contains a formal definition. Formulas are interpreted with respect to states (i.e. (location, clock valuation) pairs) of a timed automaton whose clock set is CX and labeling function is M, and an environment θ associating each propositional variable Y with a set of states. A state (l, ν) satisfies an atomic proposition p if and only if p is in the set $M(l)$. A state satisfies Y if and only if $(l, \nu) \in \theta(Y)$. $\langle a \rangle(\psi)$ holds in a state if, after executing action a, ψ is true of the state after the action transition; $[a](\psi)$

means after all action transitions involving a, ψ holds in the target state; $\exists(\psi)$ holds of a state if after some time advance of $\delta \geq 0$, ψ holds in the new state, while $\forall(\psi)$ is satisfied in a state if for all possible time advances of $\delta \geq 0$, ψ is true in the resulting states. Formula $z.(\psi)$ holds in a state if, after introducing a new clock z (z is not a clock of the timed automaton) and setting it to 0 without altering other clocks, ψ is true. The formula $\exists_{\psi_1}(\psi_2)$ means, "there exists a time advance where ψ_2 is true and ψ_1 is true for all times up to, but not including, that advance", and $\forall_{\psi_1}(\psi_2)$ means, "either ψ_2 is true for all time advances or ψ_1 releases ψ_2 from being true after some time advance."

We also introduce two derived operators: $[-](\psi)$ for $\bigwedge_{a \in \Sigma_{TA}} [a](\psi)$ (for all next actions) and $\langle - \rangle(\psi)$ for $\bigvee_{a \in \Sigma_{TA}} \langle a \rangle(\psi)$ (there exists a next action). It may be seen that $\exists(\psi)$ is equivalent to $\exists_{\mathtt{tt}}(\psi)$, and $\forall(\psi)$ to $\forall_{\mathtt{ff}}(\psi)$.

$L^{rel}_{\nu,\mu}$ MESs are mutually recursive systems of equations whose right-hand sides are basic formulas as specified above. The formal definition follows.

Definition 4 ($L^{rel}_{\nu,\mu}$ MES syntax). *Let $X_1, X_2, \ldots, X_v$ be propositional variables, and let $\psi_1, \ldots \psi_v$ all be $L^{rel}_{\nu,\mu}$ basic formulae. Then* a $L^{rel}_{\nu,\mu}$ modal equation system (MES) *is an ordered system of equations as follows, where each equation is labeled with a* parity *(μ for least fixpoint, ν for greatest fixpoint): $X_1 \overset{\mu/\nu}{=} \psi_1, X_2 \overset{\mu/\nu}{=} \psi_2, \ldots, X_v \overset{\mu/\nu}{=} \psi_v$.*

In our MES, we will assume that all variables are bound *(every variable in the right of the equation appears as some left-hand variable).*

The formal definition of the semantics of MESs may be found in [26, 27]; we recount the highlights here. Given a timed automaton and atomic-proposition interpretation M, a basic $L^{rel}_{\nu,\mu}$ formula may be seen as a function mapping sets of timed-automaton states (corresponding to the meaning of the propositional variables to the formula) to a single set of states (the states that make the formula true, given the input sets just referred to). The set of subsets of timed-automaton states ordered by set inclusion form a complete lattice; it turns out that the functions over this lattice definable by basic formulae are monotonic over this lattice, meaning they have unique greatest and least fixpoints. This fact is the lynch-pin of the formal semantics of MESs. Specifically, given MES $X_1 \overset{\mu/\nu}{=} \psi_1, \ldots, X_v \overset{\mu/\nu}{=} \psi_v$, we may construct a function that, given a set of states for X_1, returns the set of states satisfying ψ_1, where the values for $X_2, \ldots, X_v$ have been computed recursively. This function is monotonic, and therefore has a unique least and greatest fixpoint. If the parity for X_1 is μ, then the set of states satisfying X_1 is the least fixpoint of this function, while if the parity is ν then the set of states satisfying X_1 is the greatest fixpoint. By convention, the meaning of a MES is the set of states associated with X_1, the first left-hand-side in the sequence of equations. However, in the MES, each variable X_i can be interpreted as its own subformula; this interpretation will prove useful constructing proofs that a state satisfies a MES.

Given timed automaton TA, atomic-proposition interpretation function M, and propositional variable environment θ, we use $[\![\psi]\!]_{TA,M,\theta}$ to denote the set of states satisfying ψ. For an MES $\mathcal{M}$ of form $X_1 \overset{\mu/\nu}{=} \psi_1 \ldots X_v \overset{\mu/\nu}{=} \psi_v$, we write $[\![\mathcal{M}]\!]_{TA,M,\theta}$, or equivalently $[\![X_1]\!]_{TA,M,\theta}$ when there is no confusion, for the set of states satisfying the MES.

To handle the clocks used in freeze quantification ($z.(\psi)$), we extend the timed automaton's states (l, ν) to *extended states* (l, ν, ν_f) using the additional valuation component $\nu_f : CX_f \longrightarrow \mathbb{R}^{\geq 0}$. This formalism comes from [7]. When clear from context, we will refer to an extended state as (l, ν) and omit the explicit notation of ν_f.

In this paper we only consider MESs that are *alternation-free*. Intuitively, an MES is alternation free if there is no mutual recursion involving variables of different parities. For more information on the notion, see [12]. We denote the alternation-free fragment of $L^{rel}_{\nu,\mu}$ as $L^{rel,af}_{\nu,\mu}$. By definition, $L^{rel,af}_{\nu,\mu}$ is a superset of L^{rel}_{ν} because any formula with an alternation must have at least one greatest fixpoint and at least one least fixpoint. The alternation-free restriction is not prohibitive because for any timelock-free nonzeno timed automaton (see [8]), we can express any TCTL formula into a $L^{rel,af}_{\nu,\mu}$ MES [14].

Example 2 (Specifying properties with MES). Again consider the timed automaton in Figure 1 of Example 1. Two $L^{rel,af}_{\nu,\mu}$ specifications we can ask are:

$$X_1 \overset{\nu}{=} \neg broke \wedge \forall([-](X_1)) \tag{1}$$

$$X_1 \overset{\nu}{=} \neg far \vee \Big(\forall([-](X_1)) \wedge \exists(z.(\forall(z < 1)))\Big) \tag{2}$$

Equation 1 says "it is always the case the the train is not broken," and equation 2 says "it is inevitable that a train is not far."

3 Checking $L^{rel,af}_{\nu,\mu}$ Properties: A Proof-Based Approach

The $L^{rel,af}_{\nu,\mu}$ model-checking problem for timed automata may be specified as follows: given timed automaton $TA = (L, l_0, \Sigma_{TA}, CX, I, E)$, atomic-proposition interpretation function M, and $L^{rel,af}_{\nu,\mu}$ formula ψ with initial environment θ, determine if the initial state of TA satisfies ψ, i.e.: is $(l, \nu) \in [\![\psi]\!]_{TA,M,\theta}$. This section describes the proof-based approach that we use to solve such problems.

Our model-checking technique relies on the construction of proofs that are intended to establish the truth of judgments, or *sequents*, of the form $(l, cc) \vdash \psi$, where $l \in L$ is a location, $cc \in \Phi(CX \cup CX_f)$ is a clock constraint, and ψ is a $L^{rel,af}_{\nu,\mu}$ formula. Note that cc includes clocks from the timed automaton as well as any clocks used in freeze quantifications. Note that semantically, a clock constraint cc can be viewed as the set of valuations $cc = \{\nu \mid \nu \models cc\}$; likewise, we can encode a valuation ν as the clock constraint $cc_\nu = x_1 = \nu(x_1) \wedge \ldots \wedge x_n = \nu(x_n)$. A *proof rule* contains a finite number of hypothesis sequents and a conclusion sequent and may be written as follows.

$$\frac{\text{Premise 1} \qquad \ldots \qquad \text{Premise } n}{\text{Conclusion}} \; (\textit{Rule Name})$$

The intended reading of such a rule is that if each premise is valid, then so is the conclusion. Some proof rules, *axioms*, have no premises and thus assert the truth the validity of their conclusion. Given a collection of rules, our verifier builds a *proof* by chaining these proof rules together. A proof is *valid* if the proof rules are applied properly, meaning that the premise of the previous rule is the conclusion of the next rule. The proof rules are designed to be sound and complete, meaning: $(l, \nu) \in [\![\psi]\!]_{TA,M,\theta}$ if and only if there is a valid proof for $(l, cc_\nu) \vdash \psi$. The proof-construction process proceeds in an "on-the-fly" manner: rules whose conclusion matches the sequent to be proved are applied to this goal sequent, yielding new sequents that must be proved. This procedure is applied recursively, and systematically, until either a proof is found, or none can be.

3.1 Proof Rules for $L^{af}_{\nu,\mu}$ Over Timed Automata

The proof-based approach in this paper is inspired by a generic proof framework in [26, 27] based on a general theory called Predicate Equation Systems (PES). PES involved fixpoint equations over first-order predicates and used the proof-search to establish the validity of a PES. For practical reasons, one generally wishes to avoid the construction of the PES explicitly; this paper adopts this point of view, and the proof rules that it presents thus involve explicit mention of timed-automata notions, including location and edge. A selection of proof rules derived from [26, 27] is given in Figure 2. The remaining rules are in Appendix A of the supplement [16]. Several comments are in order.

1. Each rule is intended to relate a conclusion sequent involving a formula with a specific outermost operator to premise sequents involving the maximal subformula(e) of this formula. The name of the rule is based on this operator.
2. The premises also involve the use of functions *succ* and *pred*. Intuitively, $succ((l, cc))$ represents all states that are time successors of any state whose location component is l and whose clock valuation satisfies cc, while $pred((l, cc))$ are the time predecessors of these same states. These operators may be computed symbolically; that is, for any (l, cc) there is a cc' such that (l, cc') is equivalent to $succ((l, cc))$.
3. Some of the rules involve *placeholders*, which are (potentially) unions of clock constraints, given as (subscripted versions of) ϕ. Given a specific placeholder, the premise sequent $(l, cc), \phi$ is semantically equivalent to $(l, cc \wedge \phi)$; however, for notational and implementation ease, the placeholder ϕ is tracked separately from the clock constraint cc.

More discussion of placeholders is in order. Intuitively, placeholders encode a set of clock valuations that will make a sequent valid, and which will be computed once the proof is complete. In practice, we are interested in computing the largest such set. To understand their use in practice, consider the operator $\exists$. To check $\exists$, we need to find some time advance δ such that ψ is satisfied after δ time units. Rather than non-deterministically guessing δ, we use a placeholder ϕ_s in the left premise in rule $\exists_{t1}$ to encode all the time valuations that

$$\frac{(l_1, cc \wedge g_1) \vdash \psi[\lambda_1 := 0] \qquad \ldots \qquad (l_n, cc \wedge g_n) \vdash \psi[\lambda_n := 0]}{(l, cc) \vdash [a](\psi)} \; ([a]_{Act}), \text{ cond}[a]$$

cond$[a]$: $\bigcup_i \{(g_i, \lambda_i, l_i)\} = \{(l', g', \lambda') \mid (l, a, g', \lambda', l') \in E\}$

$$\frac{(l, cc), \phi_s \vdash \psi_1 \qquad (l, cc), \neg\phi_s \vdash \psi_2}{(l, cc) \vdash \psi_1 \vee \psi_2} \; (\vee_c) \qquad \frac{succ((l, cc)) \vdash \psi}{(l, cc) \vdash \forall(\psi)} \; (\forall_{t1})$$

$$\frac{succ((l, cc)), \phi_s \vdash \psi \qquad succ((l, cc), \phi_\forall) \vdash succ((l, cc)) \wedge \phi_s}{(l, cc), \phi_\forall \vdash \forall(\psi)} \; (\forall_{t2})$$

$$\frac{succ((l, cc)), \phi_s \vdash \psi \qquad (l, cc) \vdash pred(\phi_s)}{(l, cc) \vdash \exists(\psi)} \; (\exists_{t1}) \qquad \frac{succ((l, cc)), \phi_s \vdash \psi \qquad \phi_\exists \vdash pred(\phi_s)}{(l, cc), \phi_\exists \vdash \exists(\psi)} \; (\exists_{t2})$$

Fig. 2. Select proof rules from [26, 27] adapted for timed automata and MES

ensure satisfaction of ψ. The right premise then checks that the placeholder ϕ_s is some δ-unit time elapse from (l, cc). The placeholder allows us to delay the non-deterministic guess of the value of ϕ_s until it is no longer required to guess. Additionally, for performance reasons, we use *new placeholders* to handle time advance operators for sequents with placeholders. An example may be found in Rule $\exists_{t2}$, where a new placeholder $\phi_\exists$ is introduced in the right premise. While useful for performance, this choice results in subtle implementation complexities, which we discuss in Section 5.3.

Constructing Proofs. Given sequents and proof rules, proofs now may be constructed in a goal-directed fashion. A sequent is proven by applying a proof rule whose conclusion matches the form of that sequent, yielding as subgoals the corresponding premises of that rule. These subgoals may then recursively be proved. If a sequent may be proved using a rule with no premises, then the proof is complete; similarly, if a sequent is encountered a second time (because of loops in the timed automaton and recursion in an MES), then the second occurrence is also a leaf. Details may be found in [26, 27]. If the recurrent leaf involves an MES variable with parity μ, then the leaf is unsuccessful; if it involves a variable with parity ν, it is successful. A proof is valid if all its leaves are successful.

Example 3. To illustrate the proof rules, consider the timed automaton in Figure 1. Suppose we wish to prove the sequent $(2 : in, x_1 \leq 3) \vdash [exit](0 : far)$. Utilizing the first proof rule in Figure 2, we get the proof:

$$\frac{\overline{(0 : far, 1 \leq x_1 \leq 3) \vdash 0 : far}}{(2 : in, x_1 \leq 3) \vdash [exit](0 : far)}$$

In this rule, we intersect the clock constraint with the guard $x_1 \geq 1$ (if $x_1 < 1$, then there are no possible actions so the formula is true), make the destination location the new sequent, and ask if the destination satisfies the formula. Since the location is $0 : far$, the proof is complete.

$$\frac{(l,cc),\phi_{s_1} \vdash \psi_1 \qquad (l,cc),\phi_{s_2} \vdash \psi_2 \qquad (l,cc) \vdash \phi_{s_1} \vee \phi_{s_2}}{(l,cc) \vdash \psi_1 \vee \psi_2} \; (\vee_s)$$

$$\frac{(l,cc),\phi_{s_1} \vdash \psi_1 \qquad (l,cc),\phi_{s_2} \vdash \psi_2 \qquad (l,cc),\phi_\vee \vdash \phi_{s_1} \vee \phi_{s_2}}{(l,cc),\phi_\vee \vdash \psi_1 \vee \psi_2} \; (\vee_{s2})$$

$$\frac{\mathit{succ}((l,cc)),\phi_s \vdash \psi_2 \qquad \mathit{succ}((l,cc)),\mathit{pred}_<(\phi_s) \vdash \psi_1 \qquad (l,cc) \vdash \mathit{pred}(\phi_s)}{(l,cc) \vdash \exists_{\psi_1}(\psi_2)} \; (\exists_{r1})$$

$$\frac{\mathit{succ}((l,cc)),\phi_{s'} \vdash \psi_2 \qquad \mathit{succ}((l,cc)),\mathit{pred}_<(\phi_{s'}) \vdash \psi_1 \qquad (l,cc),\phi_s \vdash \mathit{pred}(\phi_{s'})}{(l,cc),\phi_s \vdash \exists_{\psi_1}(\psi_2)} \; (\exists_{r2})$$

Fig. 3. Proof Rules for $\vee$ and $\exists_{\phi_1}(\phi_2)$

3.2 New Proof Rules for the Relativized Operators of $L_{\nu,\mu}^{rel,af}$

We now introduce rules for handling the relativized time-passage modalities in $L_{\nu,\mu}^{rel,af}$. Figure 3 gives the rules for the operator $\exists_{\psi_1}(\psi_2)$. For the $\forall_{\psi_1}(\psi_2)$ operator, we use the derivation given in Lemma 1.

Here is an explanation of the proof rule $\exists_{r1}$; the proof rule $\exists_{r2}$ is similar. The idea is for the placeholder ϕ_s to encode the δ time advance needed for ψ_1 to be true. The proof-rule premises enforce that this placeholder has three properties:

1. *Left premise:* This premise checks that after the time advance taken by ϕ_s, ψ_2 is satisfied.
2. *Middle premise:* This premise checks that until all δ time-units have elapsed, that ψ_1 is indeed true. The $\mathit{pred}_<(\phi_s)$ encodes the times before ϕ_s.
3. *Right premise:* This premise checks that ϕ_s encodes some range of time elapses δ, ensuring that the state can elapse to valuations in ϕ_s.

To implement this rule, we check the premises in left-to-right order. Some subtleties involving the middle premise are discussed in Section 5.3.

Now we give the claims ensuring the correctness of these new proof rules. Their proofs are in Appendix B of the supplement [16]. This first lemma is a corrected version of a similar lemma in [7].

Lemma 1. *$\forall_{\phi_1}(\phi_2)$ is logically equivalent to $\forall(\phi_2) \vee \exists_{\phi_2}(\phi_1 \wedge \phi_2)$.*

Theorem 1 (Soundness and Completeness). *The additional $L_{\nu,\mu}^{rel,af}$ proof rules are sound and complete: for any $L_{\nu,\mu}^{rel,af}$ formula ψ and any state (l,ν), $(l,\nu) \in [\![\psi]\!]_{M,TA,\theta}$ if and only if $(l, cc_\nu) \vdash \psi$.*

4 Optimizing Performance via Derived Proof Rules

To simplify reasoning about soundness and completeness, sets of proof rules are often kept small and simple. However, we can improve the performance or proof search by having the computer work with *derived* proof rules. We describe two such situations where we use derived proof rules. We discuss a third situation, invariants, in Appendix C.2 of the supplement [16].

Optimizing $\vee$. For performance reasons we replace a rule for $\vee$ in [26, 27]. Those papers use the proof rule $\vee_c$ given in Figure 2. We instead use the proof rule $\vee_s$, which we give in Figure 3. By pushing fresh placeholders for both branches, we avoid computing the complementation operator, which often results in forming a placeholder involving a union of clock constraints.

Optimizing $\forall_{\psi_1}(\psi_2)$. Recall the derived formula for $\forall_{\psi_1}(\psi_2)$ from Lemma 1: $\forall_{\psi_1}(\psi_2)$ is equivalent to $\forall(\psi_2) \vee \exists_{\psi_2}(\psi_1 \wedge \psi_2)$. This formula requires ψ_2 to be checked three times. However, by modifying the proof rule, we notice that we can perform the checking of ψ_2 *only once*. First, we rewrite $\exists_{\psi_2}(\psi_1 \wedge \psi_2)$ as $\exists_{\leq,\psi_2}(\psi_1)$, pushing the boundary case into the left subformula. Second, the key is to compute the largest placeholder that satisfies ψ_2, to remember those states (memoize), and then to reason with this placeholder (and its time predecessor) to find the placeholders needed to satisfy the two branches of the derived formula. This reasoning allows the tool to reason with the subformula ψ_2 only once, reusing the obtained information. The derived proof rules are in Figure 4. The first two handle the simpler cases when either ψ_2 is always true (or when ψ_1 is always false) or ψ_1 is immediately true (such as when ψ_1 is an atomic proposition); the third rule ($\forall_{ro3}$) is the more complex case. The proof rules involving placeholders are similar. Their derivations as well as their proofs of soundness and completeness are in Appendix C.1 of the supplement [16].

$$\frac{(l,cc) \vdash \forall(\psi_2)}{(l,cc) \vdash \forall_{\psi_1}(\psi_2)}\,(\forall_{ro1}) \qquad \frac{(l,cc) \vdash \psi_1 \wedge \psi_2}{(l,cc) \vdash \forall_{\psi_1}(\psi_2)}\,(\forall_{ro2})$$

$$\frac{\begin{array}{cc} succ((l,cc)), \phi_{s_1} \vdash \psi_1 & \phi_\exists \vdash pred(\phi_{s_1}) \\ succ((l,cc)), \phi_{s_2} \vdash \psi_2 & succ((l,cc), \phi_\forall) \vdash succ((l,cc)) \wedge \phi_{s_2} \\ succ((l,cc)), pred(\phi_{s_1}) \vdash succ((l,cc)), \phi_{s_2} & (l,cc) \vdash \phi_\exists \vee \phi_\forall \end{array}}{(l,cc) \vdash \forall_{\psi_1}(\psi_2)}\,(\forall_{ro3})$$

Fig. 4. Derived proof rules for $\forall_{\psi_1}(\psi_2)$

5 Implementation Details

5.1 Addressing Non-convexity: Zone Unions

For a subset of properties including safety properties, *clock zones*, or convex sets of clock valuations, are used to make the model-checking as coarse-grained as possible. However, as shown in [24], certain automata with certain formulas require non-convex sets of clock valuations (unions of clock zones) to be model-checked correctly. For simplicity, we use a list of Difference Bound Matrices

(DBMs) to implement unions of clock zones. Other more complex data structures have been developed which include the Clock Difference Diagram (CDD) [5] and Clock Restriction Diagram (CRD) [22].

5.2 Addressing Performance: Simpler PES Formulas

When writing safety and liveness properties, we can use the formulas from [14]. However, in the common case where there are no nested temporal operators and the formula does not involve clock constraints, we can simplify the formulations considerably. In these cases, the subformula is a conjunction and disjunction of atomic propositions, and is represented by p or q. Here are some simplifications:

$$AG\,[p] \equiv Y \overset{\nu}{=}\; p \wedge \forall([\,-\,](Y)) \tag{3}$$

$$AF\,[p] \equiv Y \overset{\mu}{=}\; p \vee \Big(\forall([\,-\,](Y)) \wedge \exists(z.(\forall(z<1)))\Big) \tag{4}$$

$$EF\,[p] \equiv Y \overset{\mu}{=}\; p \vee \exists(\langle - \rangle(Y)) \tag{5}$$

$$EG\,[p] \equiv Y \overset{\nu}{=}\; p \wedge \Big(\exists(\langle - \rangle(Y)) \vee \forall(z.(\exists(z \geq 1)))\Big) \tag{6}$$

The correctness proofs for these simplified formulations are in Appendix C.3 of the supplement [16].

The TCTL operators here are: $AG\,[p]$ (always p), $AF\,[p]$ (inevitably p), $EG\,[p]$ (there exists a path where always p), and $EF\,[p]$ (possibly p). One noticeable feature is that these simplified liveness properties do not require relativization. Another noticeable feature is that the $\vee$ can be simplified to not use placeholders; consequently, $AG\,[p]$ and $AF\,[p]$ do not require placeholders. Additionally, our tool directly computes $\exists(z.(\forall(z<1)))$, time can elapse forever without an action transition, and its dual, $\forall(z.(\exists(z \geq 1)))$.

5.3 Placeholder Implementation Complexities

Consider the two placeholder premises in the $\forall(\psi)$ and $\exists_{\psi_1}(\psi_2)$ proof rules in Figures 2 and 3. The placeholder sequents are given here:

$$\mathit{succ}((l,cc),\phi_\forall) \vdash \mathit{succ}((l,cc)) \wedge \phi_s \text{ and } \mathit{succ}((l,cc)),\mathit{pred}_<(\phi_s) \vdash \psi_1 \tag{7}$$

In soundness and completeness proofs, we use soundness to give us a placeholder to show that the formula holds, and with completeness, we argue that some placeholder exists. Given the complexities of the formulas, the tool needs to find the *largest* such placeholder. The rules are designed for the tool to implement them in a left-to-right fashion, where placeholders are tightened by right-hand rules. However, as the placeholders are tightened, we need to make sure that the tightened placeholder still satisfies the left-hand premise. For instance, consider the second of the above placeholders. As we tighten the placeholder to satisfy ψ_1, we need to check that this placeholder is the predecessor$_<$ of the placeholder that satisfies ψ_2. These checks take extra algorithmic work.

6 Performance Evaluation

We present the results of an experimental evaluation of our method that demonstrates the types of timed automata and specifications the system can model check. Furthermore, on the subset of specifications that UPPAAL supports, we compare our tool's time performance to their tools's time performance.

6.1 Methods: Evaluation Design

In our case study, we use four different models: Carrier Sense, Multiple Access with Collision Detection (CSMA); Fischer's Mutual Exclusion (FISCHER); Generalized Railroad Crossing (GRC); and Leader election (LEADER). For more information on these models, see Appendix D.1 of the supplement [16] or [17, 26, 27].

For each model, we start at 4 processes and scale the model up by adding more processes (up to 8 processes). For each model we model-checked one valid safety (always) specification (as), one invalid safety specification (bs), one valid liveness (inevitably) specification (al), and one invalid liveness specification (bl). Each of these cases involves only one temporal operator: ψ_1 involves conjunctions and disjunctions of atomic propositions and clock constraints. In addition we tested 4 additional specifications on each property ($M1$, $M2$, $M3$, and $M4$), some of which are the leads to property $p \rightsquigarrow q$. Out of these specifications, at least one (usually $M4$) is a property with no known equivalent TCTL formula. The specifications checked are listed in Appendix D.2 of the supplement [16]. The experiments were run on an Intel Mac with 8GB ram and a quad-core 2 GHz Intel Core i7 processor running OS 10.7. Times were measured with the UNIX utility `time`.

6.2 Data and Results

The data is provided in Tables 1 and 2. Table 1 contains the remaining specifications that are not supported by UPPAAL. Table 2 contains the examples that

Table 1. Examples that UPPAAL does not support. All times are in seconds (s).

File	PES4	PES5	PES6	PES7	PES8
CSMA-as	0.29	4.62	139.16	6696.08	TO
CSMA-M3	0.01	0.03	0.14	0.80	3.99
CSMA-M4	0.01	0.03	0.14	0.71	3.66
FISCHER-M3	0.14	2.51	79.17	TO	TOsm
FISCHER-M4	0.00	0.00	0.00	2.04	2.42
GRC-M2	0.01	0.01	0.01	0.02	0.03
GRC-M4	0.00	0.00	0.01	0.02	0.01
GRC-M4ap	0.00	0.00	0.01	0.01	0.01
LEADER-M1	0.00	0.00	0.00	0.01	0.01
LEADER-M3	0.01	0.08	2.12	79.05	4242.97
LEADER-M4	0.00	0.00	0.04	0.03	0.01

Table 2. Time performance in seconds (s) on examples comparing PES and UPPAAL

File	**PES4**	**UPP4**	**PES5**	**UPP5**	**PES6**	**UPP6**	**PES7**	**UPP7**	**PES8**	**UPP8**
CSMA-al	0.01	1.45	0.03	0.24	0.13	0.25	0.72	0.26	3.65	0.26
CSMA-bl	0.01	0.26	0.03	0.27	0.13	0.27	0.73	0.28	3.53	0.33
CSMA-bs	0.01	0.33	0.05	0.27	0.22	0.27	1.14	1.33	5.09	4.66
CSMA-M1	0.01	0.29	0.03	0.27	0.14	0.28	0.73	0.27	3.69	0.27
CSMA-M2	0.33	0.35	5.21	7.00	154.56	1194.74	TO	TO	TOsm	TOsm
FISCHER-al	0.00	0.51	0.00	0.27	0.00	0.28	0.00	0.40	0.00	0.27
FISCHER-as	0.07	0.27	0.51	0.28	13.44	0.67	864.04	0.96	TO	4.26
FISCHER-bl	0.00	0.26	0.00	0.26	0.00	0.28	0.00	0.34	0.00	0.26
FISCHER-bs	0.04	0.28	0.01	0.27	0.02	0.32	0.39	0.47	0.39	0.90
FISCHER-M1	0.00	0.26	0.00	0.26	0.00	0.28	0.00	0.28	0.00	0.25
FISCHER-M2	0.00	0.26	0.00	0.26	0.00	0.27	0.00	0.30	0.03	0.28
GRC-al	0.00	0.27	0.01	0.28	0.47	0.59	0.07	0.44	0.08	5.45
GRC-as	53.09	0.36	TO	7.11	TOsm	940.51	TOsm	3433.14	TOsm	TO
GRC-bl	0.00	0.27	0.00	0.27	0.01	0.27	0.01	0.61	0.01	0.66
GRC-bs	0.11	0.41	1.91	0.41	433.59	1.76	O/M	16.19	O/M	52.03
GRC-M1	0.01	0.27	0.04	0.27	0.01	0.29	0.05	0.35	0.03	0.32
GRC-M3	0.00	0.27	0.00	0.31	0.01	0.56	0.04	1.23	0.01	3.85
LEADER-al	0.00	0.28	0.01	0.33	0.17	4.30	5.80	747.82	573.84	TO
LEADER-as	0.00	0.27	0.01	0.27	0.22	0.33	6.23	0.86	649.52	8.21
LEADER-bl	0.00	0.28	0.00	0.27	0.01	0.28	0.17	0.32	4.25	0.29
LEADER-bs	0.00	0.27	0.00	0.28	0.01	0.28	0.03	4.99	0.40	1.57
LEADER-M2	0.00	0.28	0.02	0.31	0.38	3.05	13.53	504.89	1570.37	TO

are supported both by our tool (PES) and by UPPAAL (UPP), with the number indicating the number of processes used in the model. We use the following abbreviations: TO (timeout: the example took longer than 2 hours), TOsm (the example timed out with fewer process), and O/M (out of memory). Since our tool supports a superset of the specifications that UPPAAL can support, there are specifications that our tool supports that UPPAAL does not. A scatter plot of the data in Table 2 is given in Appendix D.3 of the supplement [16].

6.3 Analysis and Discussion

After analyzing the data, we may draw three conclusions. First, on the examples that both our PES tool and UPPAAL support, we see that UPPAAL's performance is generally faster than ours, although, our tool performs faster on some examples. Additionally, while our tool does time out more often than UPPAAL does, most examples are verified quickly by both tools. Second, our tool can reasonably efficiently verify specifications that UPPAAL cannot. Third, for these examples, the performance bottleneck seems to be safety properties. Even with the additional complexity of supporting the more complicated specifications (in both tables), liveness was often verified more quickly than safety properties. Here is one possible explanation: while the verifier must check the entire state space for a valid safety property, often only a subset of the state space must be checked for a liveness property.

7 Conclusion

We provide the first implementation of a $L_{\nu,\mu}^{rel,af}$ timed automata model checker. Additionally, this model checker is on-the-fly, allowing for verification to explore both the timed automaton and the $L_{\nu,\mu}^{rel,af}$ formula incrementally. To support the full fragment of this logic, we extended the proof-rule framework of [26, 27] to support the relativization operators, and we optimize the tool's performance using derived proof rules. We also provided simpler $L_{\nu,\mu}^{rel,af}$ formulas for common safety and liveness formulas. While these may seem to be straightforward extensions, the rules and the extensions were *designed* to be straightforward, designing the proof rules to be both easy to implement efficiently.

We then compared our tool to UPPAAL. While UPPAAL seems to perform faster more often, our tool is competitive for many of those examples, including liveness formulas. Additionally, our tool was able to quickly verify specifications that UPPAAL does not currently support.

Future work is to both further optimize the performance of our tool and to augment our tool to provide more information than just a yes or no answer. Potential information includes providing answers to these questions: Was the formula true because the premise of an implication was always false? Was the formula true because certain states were never reached?

Acknowledgements. We thank Dezhuang Zhang for providing the code base [26] and for his insights.

References

[1] Aceto, L., Laroussinie, F.: Is your model checker on time? on the complexity of model checking for timed modal logics. Journal of Logic and Algebraic Programming 52-53, 7–51 (2002)

[2] Alur, R.: Timed Automata. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 8–22. Springer, Heidelberg (1999)

[3] Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Information and Computation 104(1), 2–34 (1993)

[4] Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)

[5] Behrmann, G., Larsen, K.G., Pearson, J., Weise, C., Yi, W.: Efficient Timed Reachability Analysis Using Clock Difference Diagrams. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 341–353. Springer, Heidelberg (1999)

[6] Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)

[7] Bouyer, P., Cassez, F., Laroussinie, F.: Timed modal logics for real-time systems. Journal of Logic, Language and Information 20(2), 169–203 (2011)

[8] Bowman, H., Gomez, R.: How to stop time stopping. Formal Aspects of Computing 18(4), 459–493 (2006)

[9] Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. TOPLAS 8(2), 244–263 (1986)

[10] Cleaveland, R.: Tableau-Based Model Checking in the Propositional Mu-Calculus. Acta Informatica 27(9), 725–747 (1990)

[11] Cleaveland, R., Steffen, B.: A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus. Formal Methods in System Design 2(2), 121–147 (1993)

[12] Emerson, E.A., Lei, C.L.: Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In: LICS 1986, pp. 267–278. IEEE Computer Society (1986)

[13] Fontana, P., Cleaveland, R.: Data Structure Choices for On-the-Fly Model Checking of Real-Time Systems. In: DIFTS 2011, pp. 13–21 (2011)

[14] Fontana, P., Cleaveland, R.: Expressiveness results for timed modal-mu calculi (2014) (in Preparation Preprint available upon request)

[15] Fontana, P., Cleaveland, R.: A menagerie of timed automata. ACM Computing Surveys 46(3), 40:1–40:56 (2014)

[16] Fontana, P., Cleaveland, R.: The power of proofs: New algorithms for timed automata model checking (appendix). arXiv.org (2014)

[17] Heitmeyer, C., Lynch, N.: The generalized railroad crossing: a case study in formal verification of real-time systems. In: RTSS 1994, pp. 120–131 (December 1994)

[18] Henzinger, T., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Information and Computation 111(2), 193–244 (1994)

[19] Laroussinie, F., Larsen, K.G.: CMC: A tool for compositional model-checking of real-time systems. In: Budkowski, S., Cavalli, A., Najm, E. (eds.) Formal Description Techniques and Protocol Specification, Testing and Verification. IFIP, pp. 439–456. Springer, US (1998)

[20] Peter, H.J., Ehlers, R., Mattmüller, R.: Synthia: Verification and synthesis for timed automata. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 649–655. Springer, Heidelberg (2011)

[21] Sokolsky, O.V., Smolka, S.A.: Local model checking for real-time systems. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 211–224. Springer, Heidelberg (1995)

[22] Wang, F.: Efficient verification of timed automata with BDD-like data structures. STTT 6(1), 77–97 (2004)

[23] Wang, F.: Redlib for the formal verification of embedded systems. In: ISoLA 2006, pp. 341–346. IEEE Computer Society, Piscataway (2006)

[24] Wang, F., Huang, G.D., Yu, F.: TCTL inevitability analysis of dense-time systems: From theory to engineering. IEEE Transactions on Software Engineering 32(7), 510–526 (2006)

[25] Yovine, S.: KRONOS: a verification tool for real-time systems. STTT 1(1), 123–133 (1997)

[26] Zhang, D., Cleaveland, W.R.: Fast generic model-checking for data-based systems. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 83–97. Springer, Heidelberg (2005)

[27] Zhang, D., Cleaveland, R.: Fast on-the-fly parametric real-time model checking. In: RTSS 2005, pp. 157–166. IEEE Computer Society, Washington, DC (2005)