

# Fast On-The-Fly Parametric Real-Time Model Checking

Dezhuang Zhang  
Department of Computer Science  
State University of New York  
Stony Brook, NY 11794, USA  
dezhuang@cs.sunysb.edu

Rance Cleaveland  
Department of Computer Science  
University of Maryland  
College Park, MD 20742, USA  
rance@cs.umd.edu

## Abstract

*This paper presents a local algorithm for solving the universal parametric real-time model-checking problem. The problem may be phrased as follows: given a real-time system and temporal formula, both of which may contain parameters, and a constraint over the parameters, does every allowed parameter assignment ensure that the real-time system satisfies the formula? Our approach relies on translating these model-checking problems into predicate equation systems, and then using an efficient proof-search algorithm to solve these systems. Experimental data shows that our method substantially outperforms existing approaches for systems that contain errors, while exhibiting comparable behavior for systems that are correct. This fast error-detection capability of our technique makes it especially interesting for design approaches in which model checkers are used “early and often” to detect design errors in an ongoing manner.*

## 1. Introduction

Real-time model checking [1, 5, 20] has received a lot of attention in the past 15 years. In the traditional formulation of the problem, one is given a real-time system modeled as a timed automaton, and a specification as a formula in temporal logic, and is required to determine whether or not the system satisfies the formula.

In practice, system models often contain parameters that can be adjusted to tune model behavior. In automotive and aerospace applications, these parameters are often referred to as *calibration parameters*. For example, in an automobile-engine controller one calibration parameter might describe the number of cylinders in the engine, while another might be the maximum allowed revolutions-per-minute the engine can undergo. Setting these parameters to different values allows the same model to be “deployed” for different engine models. These calibration parameters

are also usually equipped with constraints on their allowed values; the number of cylinders might be restricted to 4, 6 or 8, for example, while the maximum RPM setting might be constrained to fall in the interval [7000, 8000]. Model checking such a parameterized system would require checking model correctness for all parameter settings against a temporal formula that may also involve the same parameters.

The goal of this paper is to develop an efficient, general framework for checking a parameterized model against a formula. We call this problem the *universal parametric real-time model-checking* problem because, in contrast with other work on parametric real-time systems, our interest is in determining whether or not *every* parameter setting consistent with parameter constraints leads to correct behavior. A naive approach to the universality problem is to test each parameter assignment. For each parameter valuation, one needs to perform the model-checking algorithm once. Such a computation might be prohibitive, since the number of possible parameter valuations may be very large. We present a local parametric model-checking algorithm for solving such a general problem symbolically, which only needs one execution of the model-checking process.

Several parametric real-time analysis problems have been investigated. The *emptiness* problem is the following: given a parameterized real-time system having parametric bounds on delays, and a state in the real-time system, is there an assignment of values to the parameters such that the the desired state can be reached? This problem is known to be undecidable in general [6], although for dense time systems with single parametric clock, decision procedures exist. The *constraint synthesis* problem is related to our universality problem: given a parametric real-time system and formula, derive the most-general constraints over parameters that make the model-checking problem successful. This problem is studied in [4, 9, 10, 14, 15, 18, 21, 29, 30, 32]. Although the constraint-synthesis problem for timed CTL with parameters appearing only in formulas is decidable, the same does not hold for timed automaton with parameters

in general. For example, [15] showed that the model checking of parametric timed CTL is undecidable over timed automata with only one parametric clock. The *optimization* problem is to find the optimal valuations of parameters according to given criteria and is considered in [4, 33].

All of these problems differ from the one considered here in that no *a priori* constraints on parameters are considered as given. In our experience with a variety of automotive and aerospace companies, however, such constraints are always present, and indeed are often specified even before the parametric model is constructed. The current work is intended to initiate study into this problem and offer a solution in certain practically relevant cases.

The main contribution of the paper is a local parametric real-time model-checking algorithm. Our approach encodes the model-checking problem with predicate equation systems (PESs). The solving of PESs is performed by providing a valid proof (i.e. successful tableau) for an initial predicate. In contrast with other real-time model-checking techniques, which employ either "forward" or "backward" analysis techniques, our proof search technique works in a combined forward / backward style. Proofs are constructed in a goal-directed, "forward" manner, with information obtained in one branch of proof construction allowed to flow "backward" to improve proof construction in other branches. The forward component of our strategy supports early termination in case errors are detected, while the backward element enables efficient computation when no errors are present. Experimental data shows that our algorithm significantly outperforms other existing tools for erroneous specifications, while having comparable performance when there are no errors. Since the universal problem can be seen as the dual of the emptiness problem, it is impossible to provide an algorithm which could terminate with more than two parametric clocks (even the decidability of the case with two clocks is open). However, our solution procedure does terminate when parameter constraints take the form of finite sets: a restriction we impose in this paper.

Among existing real-time model-checking works, the algorithms of [25, 26] are most related to ours in the sense that theirs also works in a forward / backward way to refine regions. Our method is somewhat different in being based on proof search; this basis permitted us to identify situations, specifically in the checking of invariance properties, in which we can avoid clock-zone-splitting operations that their algorithm requires. Consequently, we conjecture that our algorithm will significantly outperform those, although the absence of publicly available implementations of these tools prevented us from assessing this empirically.

## 2. Parametric Timed Automata

To model parametric real-time systems, we use *parametric timed automata*. We began by introducing some terminology and notation.

Throughout let  $\mathcal{C}$  be a finite set of clock variables ranging over  $x, y, \dots$ ,  $\text{Act}$  a finite set of actions (transition labels),  $\mathcal{P}$  a finite set of parameter variables ranging over  $a, b, \dots$ , and  $\alpha, \beta$  linear terms defined over  $\mathcal{P}$  and integer constants in the usual way: each has form  $n + \sum_i n_i a_i$  where  $n$  and each  $n_i$  are integer constants and each  $a_i \in \mathcal{P}$ . The set of *state predicates* is defined by the grammar.

$$\varphi_s := \alpha \sim \beta \mid x \sim \alpha \mid x - y \sim \alpha \quad (1)$$

where  $\sim \in \{<, \leq, =, \geq, >\}$ . A parameter may assume any value in a fixed finite set of integers  $\mathcal{V}$  (in practice different parameters would have different domains, but for simplicity in this paper we assume a single domain of possible values for all parameters). We write the set of state predicates as  $\Phi$ . Throughout we let  $\mathcal{D} \stackrel{\text{def}}{=} \mathbb{R}^+ \cup \{0\}$  be the set of possible *durations* and  $\mathcal{X} \stackrel{\text{def}}{=} \mathcal{C} \cup \mathcal{P}$ .

A *parameter valuation* is a mapping  $\omega \in \mathcal{V}^{\mathcal{P}}$  (recall that  $\mathcal{V}^{\mathcal{P}}$  is the set of mappings from  $\mathcal{P}$  to  $\mathcal{V}$ ) that assigns a value to each parameter. Given a parameter valuation  $\omega$ , a *system state*  $\rho \in \mathcal{D}^{\mathcal{X}}$  satisfies:  $\rho(a) = \omega(a)$  if  $a \in \mathcal{P}$ . If  $\rho$  is a system state and  $\delta \in \mathcal{D}$  then  $\rho + \delta$  is the new state  $\rho[x_1 := \rho(x_1) + \delta, \dots, x_n := \rho(x_n) + \delta]$ , which updates each clock variable  $x_i$  with a new value  $\rho(x_i) + \delta$  and agrees with  $\rho$  otherwise. State predicates are interpreted with respect to system states in the usual fashion; we write  $\rho \models \varphi$  when this is the case.

**Definition 2.1** A parametric timed automaton (PTA) is a tuple  $T = \langle S, R, L, S_I \rangle$ , where:

1.  $S$  is a finite set of control locations;
2.  $R \subseteq S \times \Phi \times 2^{\mathcal{C}} \times \text{Act} \times S$  is a finite set of transitions,
3.  $L \in \Phi^S$  is a mapping that assigns to each location a state predicate, called the invariant for that location, in  $\Phi$ ;
4.  $S_I \subseteq S$  are the initial locations

Intuitively, time can elapse in a location only as long as its invariant remains true; when the current location is  $s$ , a transition  $\langle s, \varphi, C, \lambda, s' \rangle$  may be executed when the trigger condition  $\varphi$  is satisfied, with the clocks in  $C$  being reset to 0 and control location switched to  $s'$ .

Semantically, given a parameter valuation  $\omega \in \mathcal{V}^{\mathcal{P}}$ , a parametric timed automaton  $T = \langle S, R, L, S_I \rangle$  can be interpreted as a *concrete transition system*.

**Definition 2.2** A concrete transition system (CTS) is a tuple  $\langle \Sigma, V, \rightarrow_c, \Sigma_I \rangle$ , where  $\Sigma$  is the set of states,  $V : \Sigma \rightarrow \mathcal{D}^X$  the valuation function,  $\rightarrow_c \subseteq \Sigma \times (\text{Act} \cup \mathcal{D}) \times \Sigma$  the transition relation, and  $\Sigma_I \subseteq \Sigma$  the set of start states.

Given  $\omega$  and  $T$ , CTS  $C_{\omega, T} = \langle \Sigma, V, \rightarrow_c, \Sigma_I \rangle$  is defined as follows.

1.  $\Sigma = \{ \langle s, \rho \rangle \in S \times \mathcal{D}^X \mid \text{for each } a \in \mathcal{P}, \rho(a) = \omega(a) \}$ .
2.  $V(\langle s, \rho \rangle) = \rho$ .
3. There are two types of transitions in  $C_T$ .
  - (a) Time advance:  $\langle s, \rho \rangle \xrightarrow{\delta}_c \langle s, \rho' \rangle$  for  $\delta \in \mathcal{D}$  iff for all  $0 \leq \delta' \leq \delta$ ,  $\rho + \delta' \models L(s)$ .
  - (b) Transition firing:  $\langle s, \rho \rangle \xrightarrow{\lambda}_c \langle s', \rho' \rangle$  iff there is  $\langle s, \varphi, C, \lambda, s' \rangle \in R$  with:  $\rho \models L(s)$  and  $\rho \models \varphi$  and  $\rho' = \rho[C := 0]$
4.  $\sigma_I = \{ \langle s_I, \rho \rangle \mid s_I \in S_I, \rho \models L(s_I) \text{ and } \rho(x) = 0 \text{ for each } x \in \mathcal{C} \}$

### 3. The Real-Time Modal Mu-Calculus

To specify system properties, we use a real-time extension of the modal mu-calculus derived from [25]. Our formulas are defined using *modal equation systems* (MESs), which consist of blocks of equations of the form  $X = \phi$ , where  $X \in \mathbb{X}$  is a formula variable and  $\phi$  is a formula defined by the following grammar.

$$\begin{aligned} \phi ::= & \varphi_s \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \langle \lambda \rangle \phi \mid [\lambda] \phi \\ & \mid \exists \phi \mid \forall \phi \mid x.\phi \mid X \end{aligned}$$

In the above,  $\lambda \in \text{Act}$  is an action while  $x \in \mathcal{C}$  is clock. Operators  $\langle \lambda \rangle \phi$  and  $[\lambda] \phi$  are called modal operators; these, together with  $\vee$  and  $\wedge$ , are standard from the propositional modal mu-calculus [23].  $\varphi_s$  is a state predicate;  $x.\phi$  is a *reset* operator; and  $\exists \phi$  and  $\forall \phi$  allow us to reason about time successors of a state. This logic is expressive enough to include many timed temporal logics, including TCTL.

The definition  $\text{fpv}(\phi)$  of free formula variables in  $\phi$  is standard. We call a formula  $\phi$  *formula-closed* if  $\text{fpv}(\phi) = \emptyset$ . While negation is restricted in the logic, every formula-closed formula  $\phi$  has a formula  $\text{not}(\phi)$  that is semantically equivalent to  $\phi$ 's negation.

MESs define a mutually recursive family of formulas, one for each equation. Since a given equation can have several solutions, blocks in MESs are equipped with an indication as to whether the “least” (most restrictive) “greatest” (most permissive) solution is intended.

**Definition 3.1** An equation block has form  $\langle p, \overline{E} \rangle$ , where  $p \in \{ \mu, \nu \}$  is the parity indicator and  $\overline{E} = \langle E_1, \dots, E_n \rangle$  is a finite sequence of equations of form  $X_i = \phi_i$ , with the  $X_i$  distinct formula variables and each  $\phi_i$  a formula.

In an equation block  $\langle p, \overline{E} \rangle$ ,  $p$  determines whether the “greatest” ( $\nu$ ) or “least” ( $\mu$ ) solution of the equations is intended. The parity  $p$  is also called the parity of the variables defined in the block. An equation may also be written in the form of  $X \stackrel{p}{=} \phi$ , where  $p$  indicates the parity. We write  $\text{lhs}(B) = \{ X_1, \dots, X_n \}$  for the left-hand-side variables in block  $B$  and  $\text{rhs}(B) = \{ \phi_1, \dots, \phi_n \}$  for the right-hand-side predicates.

**Definition 3.2** A modal equation system is a finite sequence  $\langle B_1, \dots, B_n \rangle$  of equation blocks with the property that if  $i \neq j$ , then  $\text{lhs}(B_i) \cap \text{lhs}(B_j) = \emptyset$ .

The notions of lhs and rhs can be extended in the obvious manner to MESs. We call an MES  $M$  *formula-closed* if  $\bigcup_{\phi \in \text{rhs}(M)} \text{fpv}(\phi) \subseteq \text{lhs}(M)$ .

The semantics of modal mu-calculus formulas is given with respect to a CTS  $C = \langle \Sigma, V, \rightarrow_c, \Sigma_I \rangle$ , and takes the form of a relation  $\sigma \models_{C, \theta} \phi$ , which, given an environment  $\theta : \mathbb{X} \mapsto 2^\Sigma$  mapping formula variables to sets of states, determines whether or not CTS state  $\sigma$  satisfies  $\phi$ . This relation is given as follows (obvious cases omitted).

$$\begin{aligned} \sigma \models_{C, \theta} \varphi_s & \quad \text{iff } V(\sigma) \models \varphi_s \\ \sigma \models_{C, \theta} X & \quad \text{iff } \sigma \in \theta(X) \\ \sigma \models_{C, \theta} \phi_1 \vee \phi_2 & \quad \text{iff } \sigma \models_{C, \theta} \phi_1 \text{ or } \sigma \models_{C, \theta} \phi_2 \\ \sigma \models_{C, \theta} \exists \phi & \quad \text{iff } \exists \delta \text{ s.t. } \sigma \xrightarrow{\delta}_c \sigma' \wedge \sigma' \models_{C, \theta} \phi \\ \sigma \models_{C, \theta} \langle \lambda \rangle \phi & \quad \text{iff } \exists \sigma' \text{ s.t. } \sigma \xrightarrow{\lambda}_c \sigma' \wedge \sigma' \models_{C, \theta} \phi \\ \sigma \models_{C, \theta} [\lambda] \phi & \quad \text{iff } \forall \sigma' \text{ s.t. } \sigma \xrightarrow{\lambda}_c \sigma', \sigma' \models_{C, \theta} \phi \\ \sigma \models_{C, \theta} x.\phi & \quad \text{iff } \sigma \models_{C, \theta} \text{pre}(\phi, x := 0) \end{aligned}$$

In the last clause  $\text{pre}(\phi, x := 0)$  is the weakest precondition transformer; in effect,  $\text{pre}(\phi, x := 0)$  transforms  $\phi$  by replacing each occurrence of  $x$  by 0. The definition is conceptually simple but notationally complex due to the presence of equations, so its formal definition is omitted here. We define  $\llbracket \phi \rrbracket_{C, \theta} = \{ \sigma \mid \sigma \models_{C, \theta} \phi \}$ . We may now apply the general fixpoint-equation system theory, as elaborated in [27] to define the semantics of MESs. The basis of the definition is the semantics of a single block in an MES. According to Knaster-Tarski fixpoint theorem [28], every monotonic function over a complete lattice has a unique least fixpoint and a unique greatest fixpoint. As shown, for example, in [27], these fixpoints may also be interpreted as the “least permissive” and “most permissive” solutions to systems of equations defined over lattices. In the case of an MES block, the lattice in question is the set of environments mapping left-hand side variables to subsets of  $\Sigma$ , with the

lattice ordering being the point-wise extension of set containment to environments. The MES induces a monotonic function on this lattice that evaluates each right-hand side in the MES in the context of the “argument environment” and constructs an environment assigning these values to the appropriate left-hand side variables. These ideas can then be used to give semantics to lists of blocks; the details are complicated but not deep and can be found in [27].

The greatest and least fixpoints of monotonic functions also have iterative characterizations that give rise to approximation-based approaches to computing them, when they can be computed. In the case of the least fixpoint of a monotonic function  $f$ , one starts with the least element in the lattice ( $\perp$ ) and repeatedly applies  $f$  ( $f(\perp)$ ,  $f(f(\perp))$ , etc.) until the result does not change; this “fixed” result is the least fixpoint. Characterizing the maximum fixpoint is the same, except that the initial value is the maximum element of the lattice. In the case of MES blocks, the least value is the environment assigning  $\emptyset$ , the empty set of states, to each variable, while the greatest value is the environment assigning  $\Sigma$ , the complete set of states, to each variable.

We use  $\llbracket M \rrbracket_{C,\theta}$  to represent the semantics of MES  $M$  with respect to CTS  $C$  and an environment  $\theta$  that is used to interpret free predicate variables in  $M$ . If  $M$  is formula-closed then  $\llbracket M \rrbracket_{C,\theta}(X) = \llbracket M \rrbracket_{C,\theta'}(X)$  for any  $X \in \text{lhs}(M)$  and  $\theta, \theta'$ , and we write  $\llbracket M \rrbracket_C$  for this value. It also follows that if  $M$  is formula-closed and  $\psi$  is such that  $\text{fpv}(\psi) \subseteq \text{lhs}(M)$ , then  $\llbracket \psi \rrbracket_{\llbracket M \rrbracket_{C,\theta}} = \llbracket \psi \rrbracket_{\llbracket M \rrbracket_{C,\theta'}}$  for any  $\theta, \theta'$ . When this holds we use  $\llbracket \psi \rrbracket_{C,M}$  for this value, and we write  $\sigma \models_{C,M} \psi$  if  $\sigma \in \llbracket \psi \rrbracket_{C,M}$ .

As an example, the following MES states that “after performing the gate *down* action, it is always possible to raise *up* the gate within 5 units of time” [25].

$$\begin{cases} Y \stackrel{\nu}{=} \forall[-]Y \wedge \forall[\text{down}]z.X \\ X \stackrel{\mu}{=} \exists(\text{up})(z \leq 5) \vee (\forall[-\text{up}]X \wedge \exists\langle-\text{up}\rangle tt) \end{cases}$$

The notation  $[-]$  is a shorthand for  $\bigwedge_{a \in \text{Act}}[a]$ , while  $[-\text{up}]$  stands for  $\bigwedge_{a \in \text{Act} - \{\text{up}\}}[a]$  and  $\langle-\text{up}\rangle$  for  $\bigvee_{a \in \text{Act} - \{\text{up}\}}\langle a \rangle$ . Intuitively,  $[-\text{up}]\phi$  holds of a state if every action transition labeled by something other than *up* leads to a state satisfying  $\phi$ .

The definition of  $C_{\omega,T}$  implies an immediate interpretation of the mu-calculus with respect to PTA  $T$ . In addition to the other notations defined for the mu-calculus, we also introduce the following. Let  $\phi$  be a mu-calculus formula, and  $s$  a control location in PTA  $T$ , and let  $\theta$  be a mapping of mu-calculus formula variables to sets of states in  $C_{\omega,T}$ . Then  $\llbracket \phi \rrbracket_{\theta}(s) = \{\rho \mid \langle s, \rho \rangle \in \llbracket \phi \rrbracket_{C_{\omega,T},\theta} \text{ for all } \omega\}$ . That is, the “semantics” of a control location  $s$  vis à vis a formula is the set of system states that, when combined with location  $s$ , make the formula “true”, regardless of the parameter assignment  $\omega$ . Similarly, if  $M$  is a formula-closed MES,

and  $\phi$  is a mu-calculus formula with  $\text{fpv}(\phi) \subseteq \text{lhs}(M)$ , we write  $\llbracket \phi \rrbracket_{T,M}(s)$  for  $\{\rho \mid \langle s, \rho \rangle \in \llbracket \phi \rrbracket_{C_{\omega,T},M} \text{ for all } \omega\}$ . In this case, we also say that PTA  $T$  satisfies a mu-calculus formula  $\phi$  with respect to equation system  $M$  under initial (state-predicate-specified) condition  $\varphi$  (written  $T \models_M^{\varphi} \phi$ ) if for all  $s_I \in S_I$ ,  $\{\rho \mid \rho \models \varphi\} \subseteq \llbracket \phi \rrbracket_{T,M}(s_I)$ .

## 4. Predicate Equation Systems

In this section we introduce predicate equation systems (PESs), which will be the vehicles we use for solving the universal parametric model-checking problem. PESs are like MESs whose right-hand sides of PESs are predicates. Each equation of PESs takes the form of  $X = \phi$ , where  $X \in \mathbb{X}$  is a predicate variable and  $\phi$  is predicate defined by the following grammar.

$$\phi ::= \varphi_s \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid X \mid \phi[A] \mid \exists x.\phi \mid \forall x.\phi$$

In the preceding,  $\varphi_s$  is a state predicate,  $x$  is a duration variable (i.e. can assume any value a clock can), and  $A$  is an assignment of form  $x_1 := e_1, \dots, x_n := e_n$ , where  $x_i$  is a clock and  $e_i$  is a (linear) term over clock variables. All operators are the usual ones, except for  $\phi[A]$ . As the account above allows for the use of predicate variables, substitution,  $\phi[A]$ , which is usually a derived operation, must be included as an operator in the language (since, e.g.,  $X[\bar{x} := \bar{e}]$  cannot be rewritten).

The notion *fpv* of free predicate variable can be adapted from MESs in the obvious manner. We call a predicate  $\phi$  *predicate-closed* if  $\text{fpv}(\phi) = \emptyset$ .

The semantics of predicates and PESs can be adapted from MESs in the obvious way. Now the lattice in question is  $2^{\mathcal{D}^x}$  (i.e. the lattice of sets of system states, ordered by set inclusion). A given predicate formula  $\phi$  containing free predicate variable  $X$ , in the context of a predicate environment  $\theta$ , may be seen as a monotonic function  $f_{\theta}$  over this lattice as follows:  $f_{\theta}(S) = \llbracket \phi \rrbracket_{\theta[X:=S]}$ . Given a “starting” environment  $\theta$ , the semantics,  $\llbracket P \rrbracket_{\theta}$ , of PES  $P$  is an environment  $\theta'$  that, for any equation  $X = \phi$  of  $P$ , satisfies:  $\theta'(X) = \llbracket \phi \rrbracket_{\theta'[X:=\theta'(X)]}$ . and is appropriately extremal. Note that if  $P$  is predicate-closed, then  $\llbracket P \rrbracket_{\theta}(X) = \llbracket P \rrbracket_{\theta'}(X)$  for any  $X \in \text{lhs}(P)$  and  $\theta, \theta'$ , and if  $\phi$  is additionally a predicate with  $\text{fpv}(\phi) \subseteq \text{lhs}(P)$ , then  $\llbracket \phi \rrbracket_{\llbracket P \rrbracket_{\theta}} = \llbracket \phi \rrbracket_{\llbracket P \rrbracket_{\theta'}}$  for any  $\theta, \theta'$ . In this case we write  $\llbracket \phi \rrbracket_P$  for this common value, and if  $\rho \in \llbracket \phi \rrbracket_P$  we represent this notationally as  $\rho \models_P \phi$ .

## 5. From Parametric Model Checking to Predicate Equation Systems

The universal parametric model-checking problem may be phrased as follows: given a PTA  $T$ , formula-closed MES

$M$  and  $X \in \text{lhs}(M)$ , and a constraint  $\varphi$  over parameter and clock variables, does  $T \models_M^\varphi X$ ? This section shows how to translate this question into an equivalent one involving PESs. The key problem to be addressed is the symbolic representation of the set  $\llbracket X \rrbracket_{T,M}(s_I)$  for every  $s_I \in S_I$ . This is achieved by constructing a PES equation for each location in  $T$  and equation in  $M$ . Formally, we define a function  $F$  that, given a PTA  $T$  and formula-closed mu-calculus equation system  $M$ , yields a predicate-closed PES  $F(T, M)$ .  $F$  is applied on a block-by-block basis; that is,  $F(T, \langle B_1, \dots, B_n \rangle) = \langle F(T, B_1), \dots, F(T, B_n) \rangle$ .  $F(T, B) = F(T, \langle p, \overline{E} \rangle)$  in turn yields a predicate equation block of form  $\langle p, \overline{E}' \rangle$ , where for each equation  $X = \phi$  in  $\overline{E}$  and control location  $s$  in  $T$ , there is an equation of form  $Y_{s,X} = F(s, \phi)$  in  $\overline{E}'$ .  $F(s, \phi)$  is defined in the following.

$$\begin{aligned}
F(s, \varphi_s) &= \varphi_s \\
F(s, \phi_1 \vee \phi_2) &= F(s, \phi_1) \vee F(s, \phi_2) \\
F(s, \phi_1 \wedge \phi_2) &= F(s, \phi_1) \wedge F(s, \phi_2) \\
F(s, X) &= Y_{s,X} \\
F(s, \exists \phi) &= \exists d \geq 0. (F(s, \phi)[\bar{x} := \bar{x} + d]) \\
F(s, \forall \phi) &= \forall d \geq 0. (F(s, \phi)[\bar{x} := \bar{x} + d]) \\
F(s, x.\phi) &= F(s, \phi)[x := 0] \\
F(s, \langle \lambda \rangle \phi) &= \bigvee \{ \varphi \wedge (F(s', \phi)[C := 0]) \mid \\
&\quad \langle s, \varphi, C, \lambda, s' \rangle \in R \} \\
F(s, [\lambda] \phi) &= \bigwedge \{ \varphi \rightarrow (F(s', \phi)[C := 0]) \mid \\
&\quad \langle s, \varphi, C, \lambda, s' \rangle \in R \}
\end{aligned}$$

**Theorem 5.1** *Let  $T = \langle S, R, L, S_I \rangle$  be a PTA and let  $M$  be a formula-closed MES. Then for any  $s \in S$  and any  $X \in \text{lhs}(M)$ ,  $\llbracket X \rrbracket_{T,M}(s) = \llbracket Y_{s,X} \rrbracket_{F(T,M)}$ .*

It follows that  $T \models_M^\varphi X$  iff the statement  $\mathcal{D}^X = \llbracket \varphi \rightarrow \bigwedge_{s_I \in S_I} Y_{s_I, X} \rrbracket_{F(T,M)}$  is true;

## 6. On-the-Fly Parametric Real-Time Model Checking

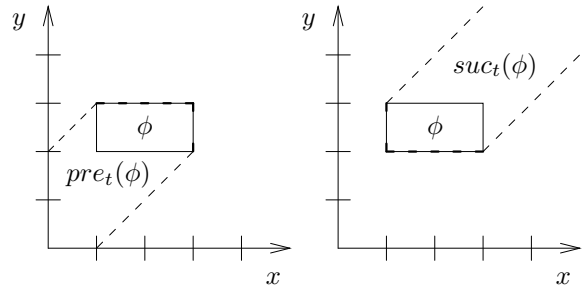
This section introduces a goal-directed proof system for solving universal parametric real-time model-checking problems encoded as PESs. The proof system is intended to establish when a set of predicate-closed formulas  $\Phi \stackrel{\text{def}}{=} \{\phi_1, \dots, \phi_n\}$  implies a formula  $\psi$  containing predicate variables from a PES. The proof rules operate on *sequents* of the form  $\Phi \vdash_P \psi$ ; a valid proof of such a sequent indicates that  $\llbracket \bigwedge \Phi \rightarrow \psi \rrbracket_P = \mathcal{D}^X$  (i.e. the implication is a tautology). The rules are given in Fig. 2 and use the following syntactic conventions: conclusions are also written above subgoals, which are separated by a “;”.  $\phi, \phi_i, \varphi, s, s'$  are predicate closed, while  $\psi, \psi_i$  need not be, and  $\Phi, \phi$  is short-hand for  $\Phi \cup \{\phi\}$ . Also note that  $s, s'$  are *placeholders* whose meaning will be clear later.

Let  $\phi$  be a predicate-closed formula and  $A \stackrel{\text{def}}{=} [\bar{x} := \bar{e}]$  an assignment. Then the *strongest postcondition*,  $\text{post}(\phi, A)$ , of  $\phi$  wrt  $A$  is defined as

$$\text{post}(\phi, \bar{x} := \bar{e}) \stackrel{\text{def}}{=} \exists \bar{v}. (\bar{x} = (\bar{e}[\bar{x} := \bar{v}]) \wedge \phi[\bar{x} := \bar{v}])$$

Note that  $\phi \vdash_P \psi[\bar{x} := \bar{e}]$  is valid if and only if  $\text{post}(\phi, \bar{x} := \bar{e})$  implies  $\psi$ . We also have two derived operators, where  $\text{pre}(\phi, A) \stackrel{\text{def}}{=} \phi[A]$  is the weakest precondition operator.

- $\text{succ}_t(\phi) \stackrel{\text{def}}{=} \exists \delta. \text{post}(\phi, \bar{x} := \bar{x} + \delta)$ , time successor of  $\phi$ .
- $\text{pre}_t(\phi) \stackrel{\text{def}}{=} \exists \delta. \text{pre}(\phi, \bar{x} := \bar{x} + \delta)$ , time predecessor of  $\phi$ .



**Figure 1.** The graphic representation of  $\text{pre}_t(\phi)$  and  $\text{succ}_t(\phi)$ .

Rules  $\vee$ – $\wedge$  are familiar from the predicate calculus; Rule C is for predicate variables. Instead of a cut rule [12] for the reasoning of case splittings like the following,

$$\frac{\Phi \vdash_P \psi}{\Phi \vdash_P \phi ; \Phi, \phi \vdash_P \psi}$$

(which in general can not be automated), we defer the computation of these predicates by introducing placeholders for them in Rule  $\vee$  and Rule  $\exists_1$  and using a “backward” analysis of the proof tree to infer values for these placeholders (see Rules  $\exists_2, \forall_2, \square_2$ ). This strategy is inspired by the *splitting* technique used in [25].

Rule  $\vee$  distributes the proof obligation into two subgoals by introducing a placeholder  $s$  in the left subgoal. The splitting constraint  $s$  is first computed through the left subtree and then the negation of  $s$  is fed into the right subsequence. For example, in Figure 3(b),  $s$  might be  $x \leq 4$ .

Rule  $\exists_1$  eliminates the existential quantifier by introducing a placeholder. The right subsequence is used to tell the validity of the splitting  $s$  derived from the left subtree. For example, in Figure 3(a),  $s$  might be the shaded region.

Rules  $\forall_2, \exists_2$  and  $\square_2$  all have a right subgoal which is used to compute the weakest splitting constraints  $s$  from  $s'$ .

$$\begin{array}{c}
\forall_1 \frac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi \vdash_P \psi_1} \quad \forall_2 \frac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi \vdash_P \psi_2} \quad \forall_3 \frac{\Phi \vdash_P \phi \vee \psi}{\Phi, \text{not}(\phi) \vdash_P \psi} \quad \forall_4 \frac{\Phi \vdash_P \psi \vee \phi}{\Phi, \text{not}(\phi) \vdash_P \psi} \\
\vee \frac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi, s \vdash_P \psi_1 ; \Phi, \neg s \vdash_P \psi_2} \quad \wedge \frac{\Phi \vdash_P \psi_1 \wedge \psi_2}{\Phi \vdash_P \psi_1 ; \Phi \vdash_P \psi_2} \quad \text{CALL} \frac{\Phi \vdash_P X}{\Phi \vdash_P \psi} \quad (X \stackrel{\text{def}}{=} \psi \in P) \\
\forall_1 \frac{\Phi \vdash_P \forall \delta. \psi[\bar{x} := \bar{x} + \delta]}{\text{suc}_t(\Phi) \vdash_P \psi} \quad \forall_2 \frac{\Phi, s \vdash_P \forall \delta. \psi[\bar{x} := \bar{x} + \delta]}{\text{suc}_t(\Phi), s' \vdash_P \psi ; \text{suc}_t(\Phi \wedge s) \vdash_P \text{suc}_t(\Phi) \wedge s'} \\
\exists_1 \frac{\Phi \vdash_P \text{pre}_t(\psi)}{\text{suc}_t(\Phi), s \vdash_P \psi ; \Phi \vdash_P \text{pre}_t(s)} \quad \exists_2 \frac{\Phi, s \vdash_P \text{pre}_t(\psi)}{\text{suc}_t(\Phi), s' \vdash_P \psi ; s \vdash_P \text{pre}_t(s')} \quad \exists_1 \frac{\Phi \vdash_P \psi[A]}{\text{post}(\Phi, A) \vdash_P \psi} \\
\exists_2 \frac{\Phi, s \vdash_P \psi[A]}{\text{post}(\Phi, A), s' \vdash_P \psi ; s \vdash_P \text{pre}(s', A)} \quad \text{LEAF} \quad \Phi, s \vdash_P \varphi \begin{cases} \text{if } \Phi \rightarrow \varphi \text{ a tautology,} & s \stackrel{\text{def}}{=} \text{true} \\ \text{if } \Phi \rightarrow \varphi \text{ a contradiction,} & s \stackrel{\text{def}}{=} \text{false} \\ \text{otherwise} & s \stackrel{\text{def}}{=} \varphi \end{cases}
\end{array}$$

Figure 2. A local approach for parametric real-time model checking.

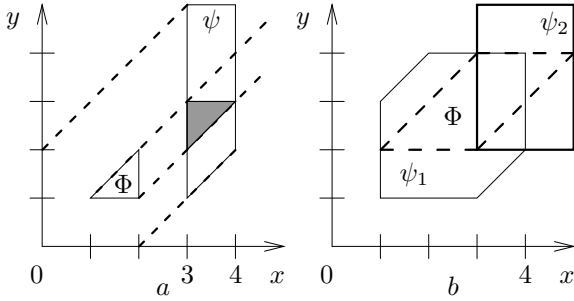


Figure 3. Examples for Rule  $\exists_1$  and  $\vee$

The rules also share an implicit side condition: they may only be applied to *non-leaf* sequents. These are defined as follows.

**Definition 6.1** Let  $\sigma$  be a sequent of form  $\Phi \vdash_P \psi$ . Then  $\sigma$  is a (successful) leaf if one of the following conditions holds. (a).  $\psi$  is a state predicate. (successful if  $\wedge \Phi \rightarrow \psi$  is a tautology). (b).  $\psi \in \Phi$  (always successful). (c).  $\psi = X$  with parity  $p$ , and there is another sequent  $\sigma'$  of form  $\Phi' \vdash_P X$  on the path from the root node of the proof to  $\sigma$  with the property that no  $\sigma'' : \Phi'' \vdash_P X''$  such that  $X''$  has parity different than  $p$  and  $X''$  is defined in an earlier block in the PES than  $X$ , and  $\Phi$  logically implies  $\Phi'$ . Such a leaf is successful if the parity of  $X$  is  $\nu$ .

The definition of (successful) leaf is based on one given in [22], which also gives a success criterion for leaves involving  $\mu$ -formulas. This criterion is not needed in this paper, so we omit further mention of it.

A proof (called *tableau*) built using these rules is *valid* if and only if it is finite, every path ends in a leaf, and every leaf is successful. The following is true.

**Theorem 6.2 (Soundness)** If  $\Phi \vdash_P \psi$  has a valid proof, then  $\llbracket \wedge \Phi \rightarrow \psi \rrbracket_P = \mathcal{D}^X$ .

In general, the proof rules will not be complete for arbitrary parametric model-checking problems. However, when the PES is generated from a (parametric) timed automaton and a (parametric) timed mu-calculus, and all parameters take values from finite sets, then completeness does hold. The proof follows from the finiteness of the number of different regions in the systems [2] and an argument for fix-point approximation similar to [20, 22].

## 7. Experimental Results

We have implemented a prototype, which we call CWB-PRT, of the above-mentioned algorithm. The effort took approximately one month, with two weeks devoted to an implementation of a *parametric difference bound matrices* (PDBMs) [21] package and the rest to building the proof-search infrastructure. C++ was used as the implementation language. PDBMs are a data type extending difference bound matrices (DBMs) [17], which are used to record clock differences for (non-parametric) real-time model checking. PDBMs are basically DBMs, where the matrix entries are linear parameter terms rather than constants. We used the Omega Library [24] as our decision procedure for inclusion checking between these linear terms.

The algorithm uses a depth-first search technique with caching. The proof rules in Figure 2 are used to generate sequents needed to be proved next in order for the goal sequent to be true. The cache contains sequents that have either been proved or disproved, or are currently assumed to be true. When a sequent is generated, the cache is first checked to see if it is implied by something in the cache; if this is the case, then no more searching is necessary for this

sequent. If the sequent is not in the cache, it is added into the cache, and rules are then recursively applied to it. The precise details of cache management are similar to those for on-the-fly propositional model checkers [8, 13], so we omit further discussion here.

To assess the performance of CWB-PRT, we ran it on several examples taken from the literature and compared the results with those from the most recent available versions of TReX-1.4 [9], HyTech-1.04f [19] and RED5.3 [32] with both forward and backward analysis. (All these tools solve the constraint-synthesis version of the problem: they compute the most general constraints on parameters that guarantee the property in question will hold. It is easy to use these results to solve the universal problem, however.) The tool TReX [9] can deal with non-linear parameter constraints. It was implemented with PDBMs and also supports the Omega Library as an external decision procedure. Both HyTech [19] and RED [32] are tools for linear hybrid automata, which are more general than parametric timed automata. While HyTech-1.04f was implemented with polyhedra as its data structures, RED5.3 was released with HRDs (Hybrid-Restriction Diagrams), a BDD-like data structure, which is more compact and efficient than PDBMs and polyhedra, see [32] for the experimental results. Due to the absence of publicly available implementations, other constraint-synthesis tools that are capable of parametric analysis, namely LPMC [26] and the extension of UPPAAL for linear parametric model checking [21], are not considered here; see [16] for the performance reports.

The experimental platform used was an Intel Pentium IV 2.8GHz with 2GB memory running Linux. The systems are listed below together with different constraints over parameters. Note that for any parameter valuation that satisfies condition (a), the model-checking problem is unsuccessful, while successful under condition (b); condition (c) is the mixed case, i.e. some parameter valuations make the problem successful, others unsuccessful.

1. *Fischer’s timed Mutual Exclusion (MUX)* [3, 32]. There are  $n$  processes trying to access a critical section. Initially each process is idle, but at any time it may begin executing the protocol provided the value of a global variable  $p$  is 0. It then delays for up to  $\Delta_B$  seconds before assigning its identifier to  $p$ . It may enter the critical section within  $\Delta_C$  seconds provided  $p$  still equal to its identifier. It reinitializes  $p$  to 0 upon leaving the critical section. We verify that at any time, no more than one process is in its critical section, when (a).  $\Delta_B \geq \Delta_C$ ; (b).  $\Delta_B < \Delta_C$ ; (c).  $\Delta_B > 0, \Delta_C > 0$ .
2. *Nuclear reactor controller (REACTOR)* [7, 32]. The goal of the system is to maintain the reactor temperature between a minimal threshold  $L$  and a maximal

threshold  $U$  by inserting control rods. A rod must stay outside for at least  $T$  time units after it is removed. We verify that whenever the temperature reaches  $U$ , one of the rods can be put in, with condition (a).  $T \geq 16 + (m - 1) * 21$ ; (b).  $T < 16 + (m - 1) * 21$ ; (c).  $T \geq (m - 1) * 21$ , where  $m$  is the number of rods in the system.

3. *Generic Railroad Crossing (GRC)*. We use the real-time version of the protocol adapted from [32]. A system operates a gate at a railroad crossing. The railroad crossing  $I$  lies in a region of interest  $R$ . A set of trains travel through  $R$  on multiple tracks in both directions. A constant parameter  $\delta$  is used to determine the controller actions. The safety property is to ensure the system will not enter an unsafe state where a train is in the crossing but the crossing gate is not down. We check with (a).  $\delta > 20$ ; (b).  $\delta \leq 20$ ; (c).  $\delta > 10$ .
4. *CSMA/CD benchmark* [32, 34]. This is the Ethernet bus arbitration protocol with the idea of collision-and-retry. A typical worst case round trip propagation is  $\delta$  time units, and it need  $\sigma$  time units to detect a collision. One safety property requires that at any moment, at most one process is in the transmission mode for no less than  $\delta$  time units. We check (a).  $\delta \leq \sigma$ ; (b).  $\delta > 52, \sigma \leq 26$ ; (c)  $\delta > 52, \sigma > 0$ .

One of the motivations for on-the-fly model checking is that bugs can be caught much more quickly than with global approaches since computation can be short-circuited when errors are found. We tested this hypothesis with buggy conditions (a) and (c) over parameters and collected comparative performance data for the model checkers in question. These figures in Table 1 and Table 2 indicate that CWB-PRT significantly outperforms the other tools in this case. We conjecture that CWB-PRT’s superior performance here is due to the combined forward / backward analysis of our algorithm. The logical infrastructure of our algorithm is useful to detect errors quickly while most of other tools are must compute a fixpoint before finding errors.

An often-mentioned criticism of on-the-fly model checking is that when system specifications and formulas are both correct, these algorithms perform very poorly. To test the validity of this statement, we ran CWB-PRT on all (b) properties for these case studies. The performance figures are given in Table 3 and tend to refute the assertion just given. Specifically, it can be seen that CWB-PRT generally outperforms TReX-1.4 and HyTech-1.04f. RED-5.3 generally outperforms CWB-PRT on these examples, although it should be noted that while TReX [34] was implemented with PDBMs, as CWB-PRT is, HyTech [11] use polyhedra and RED5.3 HRDs [32]. Since data structures have been one of the key challenges for efficient real-time model

**Table 1. Performance data with (a) conditions. The numbers in the names of the systems refer to the numbers of processes in the models. Times represent CPU time in seconds, “O/M” means “out-of-memory”, or “does not finish in two hours”. “N/A” means “not available” (especially for TReX-1.4, a segmentation fault occurs).**

	CWB-PRT	TReX1.4		HyTech1.04f		RED5.3	
		fw	bw	fw	bw	fw	bw
GRC-5-a	0.04s	O/M	O/M	O/M	O/M	334.26s	O/M
GRC-6-a	0.07s	O/M	O/M	O/M	O/M	5403.17s	O/M
GRC-10-a	0.34s	O/M	O/M	O/M	O/M	O/M	O/M
GRC-40-a	125.44s	O/M	O/M	O/M	O/M	O/M	O/M
MUX-4-a	0.06s	O/M	N/A	O/M	56.09s	2.90s	3.22s
MUX-7-a	0.13s	O/M	N/A	O/M	O/M	363.76s	1190.26s
MUX-10-a	0.20s	O/M	N/A	O/M	O/M	O/M	O/M
MUX-70-a	6.69s	O/M	N/A	O/M	O/M	O/M	O/M
CSMACD-6-a	0.01s	O/M	O/M	1354.50s	O/M	4.42s	48.68s
CSMACD-7-a	0.01s	O/M	O/M	O/M	O/M	33.35s	O/M
CSMACD-10-a	0.03s	O/M	O/M	O/M	O/M	O/M	O/M
CSMACD-70-a	4.26s	O/M	O/M	O/M	O/M	O/M	O/M
REACTOR-5-a	0.07s	O/M	O/M	1.83s	3.14s	92.05s	15.07s
REACTOR-6-a	0.11s	O/M	O/M	16.79s	49.47s	O/M	412.45s
REACTOR-7-a	0.19s	O/M	O/M	O/M	O/M	O/M	O/M
REACTOR-40-a	145.95s	O/M	O/M	O/M	O/M	O/M	O/M

**Table 2. Performance data with (c) conditions.**

	CWB-PRT	TReX1.4		HyTech1.04f		RED5.3	
		fw	bw	fw	bw	fw	bw
GRC-5-c	0.05s	O/M	O/M	O/M	O/M	338.45s	O/M
GRC-6-c	0.07s	O/M	O/M	O/M	O/M	5598.72s	O/M
GRC-10-c	0.34s	O/M	O/M	O/M	O/M	O/M	O/M
GRC-40-c	124.51s	O/M	O/M	O/M	O/M	O/M	O/M
MUX-4-c	0.08s	O/M	N/A	O/M	56.78s	23.61s	3.36s
MUX-5-c	0.13s	O/M	N/A	O/M	O/M	818.26s	29.22s
MUX-10-c	0.26s	O/M	N/A	O/M	O/M	O/M	O/M
MUX-70-c	12.26s	O/M	N/A	O/M	O/M	O/M	O/M
CSMACD-4-c	0.01s	O/M	O/M	4.17s	O/M	24.07s	2.05s
CSMACD-6-c	0.01s	O/M	O/M	775.62s	O/M	3206.95s	36.88s
CSMACD-10-c	0.01s	O/M	O/M	O/M	O/M	O/M	O/M
CSMACD-70-c	2.20s	O/M	O/M	O/M	O/M	O/M	O/M
REACTOR-5-c	0.09s	O/M	O/M	4.32s	3.68s	207.31s	18.68s
REACTOR-6-c	0.14s	O/M	O/M	68.92s	41.04s	O/M	389.22s
REACTOR-7-c	0.19s	O/M	O/M	O/M	O/M	O/M	O/M
REACTOR-40-c	144.91s	O/M	O/M	O/M	O/M	O/M	O/M

**Table 3. Performance data with (b) conditions.**

	CWB-PRT	TReX1.4		HyTech1.04f		RED5.3	
		fw	bw	fw	bw	fw	bw
GRC-2-b	2.24s	0.85s	0.50s	0.58s	1.35s	1.05s	0.31s
GRC-3-b	27.25s	O/M	O/M	22.75s	301.71s	2.75s	2.96s
GRC-4-b	271.52s	O/M	O/M	O/M	O/M	28.56s	5.47s
GRC-5-b	2263.06s	O/M	O/M	O/M	O/M	260.12s	54.66s
MUX-2-b	0.10s	0.08s	N/A	0.10s	0.09s	0.10s	0.07s
MUX-3-b	2.72s	4.40s	N/A	2.80s	2.86s	1.02s	0.45s
MUX-4-b	66.79s	648.32s	N/A	217.85s	56.20s	23.79s	3.12s
MUX-5-b	2546.32s	O/M	N/A	O/M	O/M	865.44s	9.82s
CSMACD-2-b	0.59s	O/M	O/M	0.07s	O/M	0.18s	0.18s
CSMACD-3-b	15.94s	O/M	O/M	0.39s	O/M	1.76s	1.76s
CSMACD-4-b	310.70s	O/M	O/M	4.02s	O/M	17.77s	2.07s
CSMACD-5-b	4019.83s	O/M	O/M	37.35s	O/M	212.25s	8.75s
REACTOR-6-b	1.45s	O/M	O/M	O/M	41.97s	O/M	331.70s
REACTOR-10-b	6.63s	O/M	O/M	O/M	O/M	O/M	O/M
REACTOR-20-b	53.82s	O/M	O/M	O/M	O/M	O/M	O/M
REACTOR-30-b	186.99s	O/M	O/M	O/M	O/M	O/M	O/M

checking [31, 32], we conjecture that CWB-PRT would see considerable performance improvement if we used a BDD-like data structure in place of PDBMs (our prototype uses PDBMs because of the ease of the implementation). Also, CWB-PRT’s competitiveness does suggest that our proof-search strategy, which combines forward (proof search) and backward (sequent caching) analysis, offers performance improvements over the “pure forward” or “pure backward” strategies favored by these tools.

## 8. Conclusion

We have presented an on-the-fly algorithm for solving the universal parametric real-time model-checking problem. Experimental results demonstrate that our proof-theoretic method significantly outperforms existing approaches for systems containing errors, while exhibiting competitive behavior for systems that are correct. This fast error-detection capability of our technique makes it especially interesting for industrial design flows in which model checkers are used “early and often” to detect design errors in an ongoing manner.

Our algorithm works for parameter constraints taking the form of finite sets on allowed values. We would like to investigate whether it also terminates with a more constraints over parameters in the future. Studying the constraint synthesis problem with our forward / backward approach would also be very interesting.

## References

- [1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of LICS’90*. IEEE Computer Society Press, 1990.
- [2] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
- [3] R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *RTSS*, 1992.
- [4] R. Alur, K. Etessami, S. La Torre, and D. Peled. Parametric temporal logic for model measuring. In *Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, LNCS 1644*, pages 159–168, 1999.
- [5] R. Alur and T. A. Henzinger. A really temporal logic. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, 1989.
- [6] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *ACM Symposium on Theory of Computing*, pages 592–601, 1993.
- [7] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. on Software Engineering*, 22(3):181–201, 1996.

- [8] H. R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1), 1994.
- [9] A. Annichini, A. Bouajjani, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In *CAV 2001, LNCS 2102*, pages 368–372, 2001.
- [10] G. Bandini, R. F. Lutje Spelberg, Ruud C. H. de Rooij, and Hans Toetenel. Application of parametric model checking - the root contention protocol. In *34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, 2001.
- [11] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi. Uppaal implementation secrets. In *the 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'02)*, 2002.
- [12] S. Berezin. *Model Checking and Theorem Proving: a Unified Framework*. PhD thesis, Carnegie Mellon University, 2002.
- [13] G. S. Bhat, R. Cleaveland, and O. Grumberg. Efficient on-the-fly model checking for ctl\*. In *Proceedings of the 10th Annual Symposium on Logic in Computer Science (LICS '95)*, pages 388–397, San Diego, July 1995. IEEE Computer Society Press.
- [14] V. Bruyere, E. Dall'Olio, and J.F. Raskin. Durations, parametric model-checking in timed automata with presburger arithmetic. In *STACS'03*, pages 687–698, 2003.
- [15] V. Bruyere and J.F. Raskin. Real-time model-checking: Parameters everywhere. In *FSTTCS'03*, pages 100–111, 2003.
- [16] A. Collomb-Annichini and M. Sighireanu. Parameterized reachability analysis of the iee 1394 root contention protocol using trex. In *Proceedings of Workshop on Real-Time Tools (RT-TOOLS'2001)*, Aalborg, Denmark, August 2001.
- [17] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of CAV'89*. LNCS 407, 1989.
- [18] E. A. Emerson and R. J. Treffler. Parametric quantitative temporal reasoning. In *LICS 99*, pages 336–343, 1999.
- [19] T. A. Henzinger, P. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.
- [20] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2), 1994.
- [21] T. Hune, J. Romijn, M. Stoelinga, and F. W. Vaandrager. Linear parametric model checking of timed automata. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 189–203, 2001.
- [22] J. Bradfield and C. Stirling. Local model checking for infinite state spaces. *Theoretical Computer Science*, 96:157–174, 1992.
- [23] D. Kozen. Results on the propositional  $\mu$ -calculus. *Theoretical Computer Science*, 27(3):333–354, December 1983.
- [24] W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:102–104, 1992.
- [25] O. Sokolsky and S. Smolka. Local model checking for real-time systems. In *CAV*, volume 939 of *LNCS*, July 1995. Springer-Verlag.
- [26] R. F. Lutje Spelberg and W. J. Toetenel. Parametric real-time model checking using splitting trees. *Nordic Journal of Computing*, 8(1):88–120, 2001.
- [27] L. Tan and R. Cleaveland. Evidence-based model checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*. Springer, 2002.
- [28] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pac. J. Math.*, 1955.
- [29] F. Wang. Parametric timing analysis for real-time systems. *Information and Computation*, 130:131–150, 1996.
- [30] F. Wang. Parametric analysis of computer systems. *Formal Methods in System Design*, 17:39–60, 2000.
- [31] F. Wang. Efficient verification of timed automata with bdd-like data-structures. In *VMCAI 2003, LNCS 2575*, pages 189–205, 2003.
- [32] F. Wang. Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures. In *CAV 2004, LNCS 3114*, pages 295–307, 2004.
- [33] F. Wang and H. Yen. Parametric optimization of open real-time systems. In *Static Analysis, 8th International Symposium*, pages 299–318, 2001.
- [34] S. Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1:123–133, 1997.