

Fast Generic Model-Checking for Data-Based Systems

Dezhuang Zhang and Rance Cleaveland

Department of Computer Science
State University of New York at Stony Brook, NY 11794-4400, USA
{dezhuang, rance}@cs.sunysb.edu

Abstract. This paper shows how *predicate equation systems* (PESs) may be used to solve model-checking problems for systems, such as those involving real-time or value passing, that manipulate data. PESs are first defined and the encoding of model-checking problems described; then generic global and local approaches for solving PESs are given. Real-time model checking is then considered in detail, and a new, efficient on-the-fly technique for real-time model checking based on proof search in PESs is developed and experimentally shown to significantly outperform existing approaches when system specifications or formula specifications contain errors and to be competitive when both are correct.

1 Introduction

Temporal-logic model checkers [8, 9, 24] automatically establish whether or not a system satisfies a specification given as a formula in temporal logic. The model-checking problem has been studied most intensively in the area of finite-state systems but also for real-time systems and systems involving integer-valued variables. An interesting insight to emerge in the area of finite-state model checking is that model-checking questions can be reduced to solving systems of propositional equations [2, 11] called *boolean equation systems*. This observation leads to a uniform framework for understanding a number of different model-checking techniques, including so-called *symbolic* approaches [7]. It has also served as a basis for new algorithms, including efficient on-the-fly model-checkers for the mu-calculus [2] and symbolic algorithms based on Gaussian elimination [22].

The goal of this paper is to develop a similar framework for model checking of *data-based* systems that manipulate values and thus may not be finite-state. The proposed formalism, *predicate equation systems* (PESs), generalizes boolean equation systems to full first-order logic. We indicate how different model-checking techniques for systems that process data, including Presburger systems [6] and real-time model checking [18], may be cast in terms of PESs, and we also discuss how generic model-checking techniques may be derived based on PESs. We then use a proof system for PESs as a basis for deriving a new, very efficient on-the-fly technique for model-checking real-time systems. Experimental data is presented suggesting that our prototype implementation significantly outperforms existing real-time model checkers when system specifications or formulas contain errors (the most likely scenario) and is competitive with these checkers when specifications and formulas are correct. Thus, in addition to serving as a uniform framework for describing existing model-checking routines, PESs also open new avenues for model checking as well.

2 Defining Predicate Equation Systems

Predicate equation systems consist of systems of simultaneous equations whose right-hand sides are first-order formulas. This section defines PESs and other terminology and notation used in the rest of the paper.

If \mathcal{Q} and \mathcal{X} are sets, then the set $\mathcal{Q}^{\mathcal{X}}$ consists of all functions mapping \mathcal{X} to \mathcal{Q} . We assume that if $f \in \mathcal{Q}^{\mathcal{X}}$ and $f \in \mathcal{Q}^{\mathcal{X}'}$ then $\mathcal{X} = \mathcal{X}'$, and we write $\text{dom}(f) = \mathcal{X}$ for the *domain* of f . We sometimes write $\mathcal{Q}^{\mathcal{X}}$ as $\mathcal{X} \rightarrow \mathcal{Q}$. If $f \in \mathcal{Q}^{\mathcal{X}}$ and $f' \in \mathcal{Q}^{\mathcal{X}'}$, then $f[f']$ represents the function in $(\mathcal{Q} \cup \mathcal{Q}')^{(\mathcal{X} \cup \mathcal{X}'})$ defined as follows.

$$(f[f'])(x) = \begin{cases} f'(x) & \text{if } x \in \mathcal{X}' \\ f(x) & \text{otherwise} \end{cases}$$

Also, if $f \in \mathcal{Q}^{\mathcal{X}}$ and $\mathcal{X}' \subseteq \mathcal{X}$, then $f[\mathcal{X}']$ denotes the function in $\mathcal{Q}^{\mathcal{X}'}$ defined by $(f[\mathcal{X}'])(x) = f(x)$ if $x \in \mathcal{X}'$. Finally, if $\mathcal{X} = \{x_1, \dots, x_n\}$ and $\{q_1, \dots, q_n\} \subseteq \mathcal{Q}$ then $(x_1 := q_1, \dots, x_n := q_n)$ represents the function that maps each x_i to q_i .

2.1 Basic Data Theories

The predicate calculus we consider is parameterized with respect to the *basic data theory* used to specialize the domain of discourse.

Definition 1. Let \mathcal{D} be a set of data values and \mathcal{X} a set of data variables. A basic data theory over \mathcal{X} and \mathcal{D} is a tuple $\langle \mathcal{BExp}, \mathcal{DExp}, fv, \langle - \rangle, \models, |- \rangle$, where:

1. \mathcal{BExp} is a set of data predicates;
2. \mathcal{DExp} is a set of data expressions;
3. $fv : (\mathcal{BExp} \cup \mathcal{DExp}) \rightarrow 2^{\mathcal{X}}$ is the free-variable mapping;
4. $\langle - \rangle : (\mathcal{BExp} \cup \mathcal{DExp}) \times \mathcal{DExp}^{\mathcal{X}} \rightarrow (\mathcal{BExp} \cup \mathcal{DExp})$ is the substitution function (notation: $b\langle f \rangle$ for $\langle - \rangle(b, f)$);
5. $\models \subseteq \mathcal{D}^{\mathcal{X}} \times \mathcal{BExp}$ is the interpretation relation (notation: $\rho \models b$ for $\models(\rho, b)$);
6. $|- : \mathcal{DExp} \times \mathcal{D}^{\mathcal{X}} \rightarrow \mathcal{D}$ is the evaluation function (notation: $|b|_{\rho}$ for $|-|(b, \rho)$)

and such that the following hold.

1. $(b\langle f \rangle)\langle g \rangle = b\langle f \triangleleft g \rangle$, where $(f \triangleleft g)(x) = \begin{cases} g(x) & \text{if } x \in \text{dom}(g) - \text{dom}(f) \\ f(x)\langle g \rangle & \text{otherwise} \end{cases}$
2. $|e\langle f \rangle|_{\rho} = |e|_{\rho[|f|_{\rho}]}$, where $|f|_{\rho}$ is defined by: $(|f|_{\rho})(x) = |f(x)|_{\rho}$.

In $\langle \mathcal{BExp}, \mathcal{DExp}, fv, \langle - \rangle, \models, |- \rangle$, \mathcal{BExp} is a set of atomic predicates about data values; \mathcal{DExp} is a set of data-valued expressions; $fv(b)$ the set of free data variables in b ; and $b\langle f \rangle$ is the result applying substitution f to expression b . If $\rho \models b$ then ρ makes b true, while $|e|_{\rho}$ is the result of evaluating e in ρ . If $\mathcal{I} = \{x_1, \dots, x_n\} \subseteq \mathcal{X}$ is a finite subset of \mathcal{X} we use the term *assignment* for the function $(x_1 := e_1, \dots, x_n := e_n)$ in $\mathcal{DExp}^{\mathcal{I}}$. We often use $\bar{x} := \bar{e}$ to represent an assignment and call elements of $\mathcal{D}^{\mathcal{X}}$ *data states*.

2.2 The Predicate Calculus

The predicate calculus is used to define the right-hand sides of predicate equation systems. Our account of the predicate calculus is parameterized with respect to a set \mathbb{X} of *predicate variables*, a set \mathcal{D} of data values, a set \mathcal{X} of data variables, and a basic data theory $B = \langle \mathcal{BExp}, \mathcal{DExp}, fv, \models, |- \rangle$ over \mathcal{X} and \mathcal{D} . The formulas are given as follows, where $b \in \mathcal{BExp}$, $X \in \mathbb{X}$, $x \in \mathcal{X}$, and A is an assignment.

$$\phi ::= b \mid \neg b \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid X \mid \phi[A] \mid \exists x.\phi \mid \forall x.\phi \quad (1)$$

The operators are standard, except for X and $\phi[A]$. As formulas may contain predicate variables, substitution, $\phi[A]$, which is usually a meta-operation, must be included as an operator in the language (since, e.g., $X[\bar{x} := \bar{c}]$ cannot be rewritten). The definition $fdv(\phi)$ of free (data) variables in ϕ is given in the usual manner, based on the definition of fv given in the basic data theory; the definition $fpv(\phi)$ of free predicate variables is standard. We call a formula ϕ *predicate-closed* if $fpv(\phi) = \emptyset$ and *closed* if $fpv(\phi) = fdv(\phi) = \emptyset$. We often call formulas generated by the above grammar *predicates*.

Predicates are interpreted with respect to a data state ρ and a *predicate state* $\theta \in (2^{\mathcal{D}^X})^X$ mapping predicate variables to sets of data states. We write $\rho \models_{\theta} \phi$ to denote that formula ϕ holds in data state ρ and predicate state θ . The definition is as follows.

$$\begin{array}{ll}
\rho \models_{\theta} b & \text{iff } \rho \models b \text{ (i.e. wrt basic data theory)} \\
\rho \models_{\theta} \neg b & \text{iff } \rho \not\models b \\
\rho \models_{\theta} \phi_1 \vee \phi_2 & \text{iff } \rho \models_{\theta} \phi_1 \text{ or } \rho \models_{\theta} \phi_2 \\
\rho \models_{\theta} \phi_1 \wedge \phi_2 & \text{iff } \rho \models_{\theta} \phi_1 \text{ and } \rho \models_{\theta} \phi_2 \\
\rho \models_{\theta} X & \text{iff } \rho \in \theta(X) \\
\rho \models_{\theta} \phi[A] & \text{iff } \rho[A]_{\rho} \models_{\theta} \phi \\
\rho \models_{\theta} \exists x. \phi & \text{iff for some } d \in \mathcal{D}, \rho[x := d] \models_{\theta} \phi \\
\rho \models_{\theta} \forall x. \phi & \text{iff for all } d \in \mathcal{D}, \rho[x := d] \models_{\theta} \phi
\end{array}$$

We use $\llbracket \phi \rrbracket_{\theta}$ to represent the set $\{\rho \mid \rho \models_{\theta} \phi\}$. If a formula ϕ is predicate-closed, then $\llbracket \phi \rrbracket_{\theta} = \llbracket \phi \rrbracket_{\theta'}$ for any θ and θ' ; in this case we write $\llbracket \phi \rrbracket$ for this common value. Finally, while negation is restricted in the logic, every predicate-closed formula ϕ has a formula $\text{not}(\phi)$ that is semantically equivalent to ϕ 's negation.

2.3 Predicate Equation Systems

Predicate Equation Systems (PESs) consist of blocks of equations of the form $X = \phi$, where X is a predicate variable and ϕ is a predicate. Such a system is intended to define a mutually recursive family of predicates, one for each equation. Since a given equation can have several solutions, blocks in PESs are equipped with an indication as to whether the “least” (most restrictive) “greatest” (most permissive) solution is intended.

Definition 2. A predicate 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 predicate variables and each ϕ a predicate.

In predicate block $\langle p, \overline{E} \rangle$ p determines whether the “greatest” (ν) or “least” (μ) solution of the equations is intended. 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. A predicate equation system (PES) is a finite sequence $\langle B_1, \dots, B_n \rangle$ of predicate 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 PESs. We call a PES P *predicate-closed* if $\bigcup_{\phi \in \text{rhs}(P)} fpv(\phi) \subseteq \text{lhs}(P)$.

PESs are interpreted using fixpoints of monotonic functions defined over the complete lattice given by $2^{\mathcal{D}^X}$ (i.e. the lattice of sets of data states, ordered by set inclusion). Given a predicate environment θ , a predicate ϕ containing free predicate variable X may be seen as a function f_{θ} over this lattice as follows: $f_{\theta}(S) = \llbracket \phi \rrbracket_{\theta[X := S]}$. A complete account of fixpoint equation systems is given in [30], and the semantics of

PESs may be seen as an instance of this, where the lattice Q is taken to be $2^{\mathcal{D}^{\mathcal{X}}}$. Given a “starting” environment θ , the semantics, $\llbracket P \rrbracket_{\theta}$, of PES P is an environment θ' that, for any equation $X = E$ of P , satisfies: $\theta'(X) = |E|_{\theta'[X:=\theta'(X)]}$. 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 θ, θ' . Based on this observation, it follows that if ϕ is a predicate, P is predicate-closed, and $\text{fpv}(\phi) \subseteq \text{lhs}(P)$, then $\llbracket \phi \rrbracket_{\llbracket P \rrbracket_{\theta}} = \llbracket \phi \rrbracket_{\llbracket P \rrbracket_{\theta'}}$ for any θ, θ' . In this case we write $\llbracket \phi \rrbracket_P$ for this common value, and if $\sigma \in \llbracket \phi \rrbracket_P$ we represent this notationally as $\sigma \models_P \phi$.

3 Transition Systems and the Modal Mu-Calculus

A goal of this paper is to reduce model checking to computing solutions of PESs. The basic approach consists of showing how, given a symbolic system model and a formula in the first-order mu-calculus, a PES may be generated whose “solutions” are answers for the model-checking problem. This section lays the foundation for this approach by introducing our system model, *parameterized symbolic transition graphs with assignment* (PSTGAs), and our temporal logic, the first-order mu-calculus.

3.1 Concrete Transition Systems

Fix a set of data values \mathcal{D} , a set of data variables \mathcal{X} , and a set Λ of communication port names not containing a distinguished value τ . The set of *concrete actions* Act_c is given as $\text{Act}_c = \{\lambda!d \mid \lambda \in \Lambda, d \in \mathcal{D}\} \cup \{\lambda?d \mid \lambda \in \Lambda, d \in \mathcal{D}\} \cup \{\tau\}$. Actions have the usual interpretation: $\lambda!d$ represents the emission of value d on port λ , and $\lambda?d$ the receipt of value d on λ . τ denotes the internal action.

Definition 4. A concrete transition system (CTS) is a tuple $\langle \Sigma, V \rightarrow_c, \Sigma_I \rangle$, where Σ is the set of states, $V : \Sigma \rightarrow \mathcal{D}^{\mathcal{X}}$ the valuation function, $\rightarrow_c \subseteq \Sigma \times \text{Act}_c \times \Sigma$ the transition relation, and $\Sigma_I \subseteq \Sigma$ the set of start states.

A CTS models the behavior of a system. We write $\sigma \xrightarrow{a}_c \sigma'$ for $\langle \sigma, a, \sigma' \rangle \in \rightarrow_c$.

3.2 The First-Order Modal Mu-Calculus

To specify system properties, we use first-order modal mu-calculus [29] and modal equation systems (MESs). The former enhances the predicate calculus with modal operators; MESs are like PESs whose right-hand sides of MESs are mu-calculus formulas. Fix basic data theory $\langle \mathcal{B}\text{Exp}, \mathcal{D}\text{Exp}, \text{fv}, \langle -, \models, |- \rangle$ and set Λ of port names. Then first-order mu-calculus formulas have the following form, where $e \in \mathcal{D}\text{Exp}$ and $\lambda \in \Lambda$.

$$\phi ::= \langle \text{operators from Equation 1} \rangle \mid \langle \tau \rangle \phi \mid [\tau] \phi \mid \langle \lambda!e \rangle \phi \mid [\lambda!e] \phi \mid \langle \lambda?e \rangle \phi \mid [\lambda?e] \phi$$

The notions *fpv* and *fdv* of free formula / data variables may be adapted in the obvious manner. We call a mu-calculus formula ϕ *formula-closed* if $\text{fpv}(\phi) = \emptyset$.

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 $\langle \sigma, \rho \rangle \models_{C, \theta} \phi$, which, given an environment $\theta \in (2^{\langle \Sigma \times \mathcal{D}^{\mathcal{X}} \rangle})^{\mathcal{X}}$ mapping predicate variables to sets consisting of states paired with *alternative data assignments* (used to handle bound variables), determines

whether or not CTS state σ paired with ρ satisfies ϕ . This relation is given as follows (obvious cases omitted).

$$\begin{aligned}
\langle \sigma, \rho \rangle \models_{C, \theta} X & \quad \text{iff } \langle \sigma, \rho \rangle \in \theta(X) \\
\langle \sigma, \rho \rangle \models_{C, \theta} \exists x. \phi & \quad \text{iff for some } d, \langle \sigma, \rho[x := d] \rangle \models_{C, \theta} \phi \\
\langle \sigma, \rho \rangle \models_{C, \theta} \langle \tau \rangle \phi & \quad \text{iff there is } \sigma' \text{ s.t. } \sigma \xrightarrow{\tau}_c \sigma' \text{ and } \langle \sigma', \rho \rangle \models_{C, \theta} \phi \\
\langle \sigma, \rho \rangle \models_{C, \theta} [\tau] \phi & \quad \text{iff for all } \sigma' \text{ s.t. } \sigma \xrightarrow{\tau}_c \sigma', \langle \sigma', \rho \rangle \models_{C, \theta} \phi \\
\langle \sigma, \rho \rangle \models_{C, \theta} \langle \lambda!e \rangle \phi & \quad \text{iff there is } \sigma' \text{ s.t. } \sigma \xrightarrow{\lambda!d}_c \sigma', |e|_{V(\sigma)[\rho]} = d, \text{ and } \langle \sigma', \rho \rangle \models_{C, \theta} \phi \\
\langle \sigma, \rho \rangle \models_{C, \theta} \langle \lambda?e \rangle \phi & \quad \text{iff there is } \sigma' \text{ s.t. } \sigma \xrightarrow{\lambda?d}_c \sigma', |e|_{V(\sigma)[\rho]} = d, \text{ and } \langle \sigma', \rho \rangle \models_{C, \theta} \phi
\end{aligned}$$

Note that the semantics of the modal operators are different from the ones given in [21, 26]. Here, in $\langle \lambda?x \rangle \phi$ the x in ϕ is not bound, while in the other work this is the case. Our logic only permits variables to be bound using \forall and \exists .

We sometimes write $\sigma \models_{C, \theta} \phi$ if $\langle \sigma, \emptyset \rangle \models_{C, \theta} \phi$, where \emptyset is the empty data assignment. We also define $\llbracket \phi \rrbracket_{C, \theta} = \{ \langle \sigma, \rho \rangle \mid \langle \sigma, \rho \rangle \models_{C, \theta} \phi \}$. We may now apply the general fixpoint-equation system theory in [30] to define the semantics of mu-calculus equation systems (MESSs). The lattice in question is $2^{(\Sigma \times \mathbb{A}(\mathcal{D}, \mathcal{X}))}$ ordered by set inclusion, where $\mathbb{A}(\mathcal{D}, \mathcal{X}) = \bigcup_{\mathcal{I} \subseteq \mathcal{X}} \mathcal{D}^{\mathcal{I}}$ is the set of assignments; the semantics, $\llbracket M \rrbracket_{C, \theta}$, of mu-calculus equation system M is an environment mapping each $X \in \text{lhs}(M)$ to a set of state / assignment pairs that is the appropriate solution for the equation defining X .

We also adapt the definitions of formula/predicate-closed-ness from PESs in the obvious manner. If MES 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 θ, θ' , and we write $\llbracket M \rrbracket_C$ for this value. It also follows that if M is formula-closed and ϕ is such that $\text{fpv}(\phi) \subseteq \text{lhs}(M)$, then $\llbracket \phi \rrbracket_{\llbracket M \rrbracket_{C, \theta}} = \llbracket \phi \rrbracket_{\llbracket M \rrbracket_{C, \theta'}}$ for any θ, θ' . When this holds we use $\llbracket \phi \rrbracket_{C, M}$ for this value, and we write $\langle \sigma, \rho \rangle \models_{C, M} \phi$ if $\langle \sigma, \rho \rangle \in \llbracket \phi \rrbracket_{C, M}$, and $\sigma \models_{C, M} \phi$ if $\langle \sigma, V(\sigma) \rangle \in \llbracket \phi \rrbracket_{C, M}$.

3.3 Parameterized Symbolic Transition Graphs with Assignment

Our symbolic system model, Parameterized Symbolic Transition Graphs with Assignment (PSTGAs), extends the STGA formalism of [21] with a facility for *parameterized transitions*. This enables them to encode a range of other symbolic system formats, including the value-passing CCS in [10], Linear Process Equations [16], the event-action language in [6], and timed automata [18].

Fix value set \mathcal{D} , variable set \mathcal{X} , and data theory $\langle \mathcal{B}\text{Exp}, \mathcal{D}\text{Exp}, \text{fv}, \langle - \rangle, \models, | - | \rangle$ over \mathcal{D} and \mathcal{X} . Let Φ be the associated set of predicate-calculus formulas. Also fix a set Λ of communication port names not containing the distinguished name τ , and define the set of symbolic actions $\text{Act}_s = \{ \lambda?x \mid c \in \Lambda, x \in \mathcal{X} \} \cup \{ \lambda!e \mid c \in \Lambda, e \in \mathcal{D}\text{Exp} \} \cup \{ \tau \}$.

Definition 5. A PSTGA is a tuple $G = \langle S, \mathcal{I}, R, S_I, \text{InitC} \rangle$, where:

1. S is a finite set of control locations;
2. $\mathcal{I} \subseteq \mathcal{X}$ is a finite set of assignable data variables;
3. $R \subseteq S \times (\mathcal{X} - \mathcal{I}) \times \Phi \times \Phi \times \mathbb{A}(B, \mathcal{I}) \times \text{Act}_s \times S$ is a finite set of parameterized transitions satisfying: if $\langle s, k, \kappa, \beta, A, \lambda?x, s' \rangle \in R$ then $x \in \mathcal{I}$;
4. $S_I \subseteq S$ are the initial locations; and
5. $\text{InitC} \in \mathcal{B}\text{Exp}$ is the initial condition.

In PSTGA $G = \langle S, \mathcal{I}, R, S_I, \text{InitC} \rangle$, S_I contains the possible starting locations and InitC the initial condition on data variables. Based on the current control location and data state, transitions may fire, with data variables and control locations being updated.

With this intuition in mind, let us more closely examine the structure of parameterized transitions in a PSTGA. Each transition is a tuple $\langle s, k, \kappa, \beta, A, \alpha, s' \rangle$, where s and s' are the source and target control location, respectively. The k and κ are used to parameterize, or “index”, the transition. Roughly speaking, each value d , that, when substituted for k , makes κ “true”, defines a transition, in the STGA sense, consisting of: a boolean guard $\beta\langle k := d \rangle$ determining when the transition can “fire”; an assignment $A_d^k = (A \triangleleft (k := d)) \upharpoonright_{\text{dom}(A)}$ to variables in \mathcal{I} ; and a communication action $\alpha[k := d]$ (defined as the replacement of free occurrences of data variable k in the action expression by d). STGA transitions, in contrast, omit k and κ . The utility of our more complex model will become apparent when we consider timed automata.

Semantically, a PSTGA $G = \langle S, \mathcal{I}, R, S_I, \text{Init}\mathcal{C} \rangle$ is interpreted as a CTS $C_G = \langle \Sigma, V, \rightarrow_c, \Sigma_I \rangle$ as follows.

1. $\Sigma \subseteq S \times \mathcal{D}^{\mathcal{X}}$. Note that in $\langle s, \rho \rangle$, ρ provides values to the data variables.
2. $V(\langle s, \rho \rangle) = \rho$.
3. $\langle s, \rho \rangle \xrightarrow{a}_c \langle s', \rho' \rangle$, iff there is $\langle s, k, \kappa, \beta, A, \alpha, s' \rangle \in R$, $d' \in D$, and ρ'' with:
 - (a) $\rho \models \kappa[k := d']$, $\rho \models \beta[k := d']$, $\rho'' = \rho'[k := d'] \llbracket A_d^k \rrbracket_{\rho}$, and
 - (b) either: (i) $a = \tau$ and $\rho' = \rho''$; or (ii) $a = \lambda!d$, $\alpha[k := d'] = \lambda!e$, $|e|_{\rho} = d$, and $\rho' = \rho''$; or $a = \lambda?d$, $\alpha[k := d'] = \lambda?x$, and $\rho' = \rho''[x := d]$.
4. $\sigma_I = \{ \langle s_I, \rho \rangle \mid s_I \in S_I, \rho \models \text{Init}\mathcal{C} \}$

PSTGAs and the Mu-Calculus. The definition of C_G implies an immediate interpretation of the mu-calculus with respect to PSTGA G . In addition to the other notations defined for the mu-calculus, we also introduce the following. Let ϕ be a mu-calculus formula, and s a control location in PSTGA G , and let θ be a mapping of mu-calculus formula variables to sets of states in C_G paired with assignments. Then $\llbracket \phi \rrbracket_{\theta}(s) = \{ \rho \mid \langle \langle s, \rho \rangle, \rho \rangle \in \llbracket \phi \rrbracket_{C_G, \theta} \}$. That is, the “semantics” of a control location s vis à vis a formula is the set of data states that, when combined with s , make the formula “true”. Similarly, if M is a formula-closed MES, and ϕ is a mu-calculus formula with $\text{fpv}(\phi) \subseteq \text{lhs}(M)$, we write $\llbracket \phi \rrbracket_{G, M}(s)$ for $\{ \rho \mid \langle \langle s, \rho \rangle, \rho \rangle \in \llbracket \phi \rrbracket_{C_G, M} \}$. In this case, we also say that a PSTGA G satisfies a mu-calculus formula ϕ with respect to equation system M (written $G \models_M \phi$) if for all $s_I \in S_I$, $\{ \rho \mid \rho \models \text{Init}\mathcal{C} \} \subseteq \llbracket \phi \rrbracket_{G, M}(s_I)$.

4 From Model Checking to Predicate Equation Systems

The model-checking problem for PSTGAs is: given PSTGA G , formula-closed MES M and $X \in \text{lhs}(M)$, does $G \models_M X$? 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 $\llbracket X \rrbracket_{G, M}(s_I)$ for every $s_I \in S_I$. This is achieved by constructing a PES equation for each state in G and equation in M . Formally, we define a function F that, given a PSTGA G and formula-closed mu-calculus equation system M , yields a predicate-closed PES $F(G, M)$. F is applied on a block-by-block basis; that is, $F(G, \langle B_1, \dots, B_n \rangle) = \langle F(G, B_1), \dots, F(G, B_n) \rangle$. $F(G, B) = F(G, \langle p, \overline{E} \rangle)$ 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 G , there is an equation of form $Y_{s, X} = F(s, \phi)$ in \overline{E}' . $F(s, \phi)$ is defined in Figure 1.

Theorem 1. *Let $G = \langle S, \mathcal{I}, R, S_I, \text{Init}\mathcal{C} \rangle$ be a PSTGA, and let M be a closed MES. Then for any $s \in S$ and any $X \in \text{lhs}(M)$, $\llbracket X \rrbracket_{G, M}(s) = \llbracket Y_{s, X} \rrbracket_{F(G, M)}$.*

$F(s, b)$	$= b$	$F(s, \neg b)$	$= \neg b$
$F(s, \phi_1 \vee \phi_2)$	$= F(s, \phi_1) \vee F(s, \phi_2)$	$F(s, \phi_1 \wedge \phi_2)$	$= F(s, \phi_1) \wedge F(s, \phi_2)$
$F(s, X)$	$= Y_{s, X}$	$F(s, \exists x. \phi)$	$= \exists x. F(s, \phi)$
$F(s, \forall x. \phi)$	$= \forall x. F(s, \phi)$	$F(s, \phi[x := e])$	$= F(s, \phi)[x := e]$
$F(s, \langle \tau \rangle \phi)$	$= \bigvee \{ \exists k. \kappa \wedge \beta \wedge F(s', \phi)[A] \mid \langle s, k, \kappa, \beta, A, \tau, s' \rangle \in R \}$		
$F(s, [\tau] \phi)$	$= \bigwedge \{ \forall k. (\kappa \wedge \beta) \rightarrow F(s', \phi)[A] \mid \langle s, k, \kappa, \beta, A, \tau, s' \rangle \in R \}$		
$F(s, \langle c!e \rangle \phi)$	$= \bigvee \{ \exists k. \kappa \wedge \beta \wedge F(s', \phi)[A] \mid \langle s, k, \kappa, \beta, A, \alpha, s' \rangle \in R \wedge (\alpha = c!e) \}$		
$F(s, [c!e] \phi)$	$= \bigwedge \{ \forall k. (\kappa \wedge \beta) \rightarrow F(s', \phi)[A] \mid \langle s, k, \kappa, \beta, A, \alpha, s' \rangle \in R \wedge (\alpha = c!e) \}$		
$F(s, \langle c?e \rangle \phi)$	$= \bigvee \{ \exists k. \kappa \wedge \beta \wedge F(s', \phi)[A][x := e] \mid \langle s, k, \kappa, \beta, A, \alpha, s' \rangle \in R \wedge \alpha = c?x \}$		
$F(s, [c?e] \phi)$	$= \bigwedge \{ \forall k. \kappa \wedge \beta \rightarrow F(s', \phi)[A][x := e] \mid \langle s, k, \kappa, \beta, A, \alpha, s' \rangle \in R \wedge \alpha = c?x \}$		

Fig. 1. Translation Function for PESs

5 Encoding Real Time Model Checking Via PESs

Different model-checking problems may be uniformly captured in terms of PESs, from boolean equation systems (BESs) [2, 23] and Presburger systems [6] to real-time model checking [18]. To illustrate this we detail an encoding of real-time model checking.

The framework we consider is given in [18], which models real-time systems using so-called *guarded-command real-time programs* (which are expressively equivalent to the better-known timed-automaton formalism) and uses the timed modal mu-calculus to define properties. Let R^+ be the set of nonnegative real numbers, $C = \{x_1, \dots, x_n\}$ be a finite set of clock variables, and \mathcal{P} be a finite set of boolean variables. The set of state predicates is defined by the following grammar, where $p \in \mathcal{P}$, $x, y \in C$, and $c, d \in \mathbb{N}$ are nonnegative integer constants.

$$\phi ::= p \mid x \leq d \mid c \leq y \mid x + c \leq y + d \mid \neg \phi \mid \phi_1 \vee \phi_2$$

A (clock) state $\rho \in (R^+)^{(C \cup \mathcal{P})}$ satisfies: $\rho(p) \in \{0, 1\}$ if $p \in \mathcal{P}$ (here 0 is interpreted as “false”, and 1 as “true”). If ρ is a state and $\delta \in R^+$ then $\rho + \delta$ is the new state $\rho[x_1 := \rho(x_1) + \delta, \dots, x_n := \rho(x_n) + \delta]$. State predicates are interpreted with respect to states in the usual fashion; we write $\rho \models \phi$ when this is the case. Then a real-time program has form $R = \langle G, \phi \rangle$, where:

- G is a finite set of guarded commands of form $\psi \rightarrow A$, with ψ a state predicate and A an assignment of form $v_1 := e_1, \dots, v_n := e_n$. If $v_i \in \mathcal{P}$, then e_i must either be 0 or 1; if $v_i \in C$, then e_i must either be 0 (reset) or v (no change).
- State predicate ϕ , the *invariant*, is past-closed: if $\rho + \delta \models \phi$ then $\rho \models \phi$.

In state program R executes as follows. If a $\psi \rightarrow A$ is such that $\rho \models \psi$, then the assignment A is “executed” in the usual manner, updating ρ to ρ' , provided that $\rho' \models \phi$ also. In addition, provided $\rho + \delta \models \phi$, R may “idle”.

The syntax of the timed mu-calculus extends the state-predicate language.

$$\phi ::= \langle \text{state-predicate operators} \rangle \mid X \mid \phi_1 \triangleright \phi_2 \mid z.\phi \mid \mu X.\phi$$

The new operators include capabilities for recursive definition ($X, \mu X.\phi$), a modal operator ($\phi_1 \triangleright \phi_2$), and the *freeze quantifier* ($z.\phi$), which sets “specification clock” z to 0. The formula $\phi_1 \triangleright \phi_2$ has the following meaning: $\phi_1 \vee \phi_2$ holds as time elapses, until a guarded-command fires, at which point ϕ_2 holds.

Our translation of timed model-checking into PESs converts a real-time-program / timed-mu-calculus formula into a PSTGA (the formula is needed for the specification

clocks). Then the timed mu-calculus formula is translated into a MES, and our generic generator applied. Let $R = \langle G, \phi \rangle$ be a real-time program and γ be a timed mu-calculus formula, with $\S(\gamma) = \{s_1, \dots, s_m\}$ the names of the specification clocks used in γ . To define a PSTGA, we introduce the following basic data theory (take $\mathcal{D} = \mathbb{R}^+$).

- $\mathcal{X} = C \cup \S(\gamma) \cup \mathcal{P} \cup \{k\}$, where $k \notin C \cup \S(\phi) \cup \mathcal{P}$
- $\mathcal{BExp} = \{p = 1 \mid p \in \mathcal{P}\} \cup \{x + c \leq y + d \mid x, y \in C, c, d \in \mathbb{N}\}$
- $\mathcal{DExp} = \{0, 1\} \cup \{x + k \mid x \in C \cup \S(\gamma)\}$
- $fv, \langle -, \rangle, \models, \dashv$ defined in the usual manner

We now define the PSTGA $G_{R,\gamma} = \langle S, \mathcal{I}, R, S_I, \text{InitC} \rangle$ associated with R and ϕ as follows. We take $S = \{s\}$ (i.e. there is one control location) and $\mathcal{I} = C \cup \S(\gamma)$. For each $\psi \rightarrow A \in G$, R contains a transition $\langle s, k, k = 0, \psi, A, \tau, s \rangle$, a τ -labeled transition corresponding to each guarded command in R . Note that k cannot appear free in ψ or A . There is also a parameterized transition of form $\langle s, k, \phi[x_1 := x_1 + k, \dots, x_n := x_n + k], tt, [\bar{x} := \bar{x} + k], t!k, s \rangle$. The notation $\bar{x} := \bar{x} + k$ is shorthand for $x_1 := x_1 + k, \dots, x_n := x_n + k$, etc. This transition models the ability of time to advance, so long as the state invariant remains true. The action label $t!k$ uses a special port t on which the delay k is written. Finally, we set $S_I = \{s\}$ and $\text{InitC} = \phi \wedge \bigwedge_{x \in \mathcal{X}} x = 0$.

The translation of the timed mu-calculus formula γ is omitted; we only comment on $\phi_1 \triangleright \phi_2$ and $z.\phi$. The former is given as $\exists \delta. \langle t!\delta \rangle \langle \tau \rangle \phi_2 \wedge \forall \epsilon. \epsilon \leq \delta \rightarrow \langle t!\epsilon \rangle (\phi_1 \vee \phi_2)$. The latter is $\phi[z := 0]$. Once all instances of these operators have been eliminated, the resulting formula can be easily converted into an MES.

6 Generic Algorithmic Approaches

The previous section suggested how model-checking problems can be encoded as PESs. This section discusses algorithmic issues involved in computing solutions to PESs and their relation to the specific algorithms given by researchers studying the aforementioned problems.

Global Approaches. The paper [30] gives an iterative strategy for computing the solution to fixpoint equation systems that is based on the following technique for computing solutions to basic blocks.

1. Assign each lhs variable the correct extremal value (\top for ν , \perp for μ).
2. ‘‘Iterate’’ by evaluating the right-hand side of each equation using the current assignment to derive a new assignment. Terminate when there is no change.

For PESs, this strategy has an obvious symbolic implementation: for a ν -block, initialize each lhs predicate variable to tt , then iterate by replacing the lhs variables by their definitions in rhs predicates and checking if the old predicates imply the new ones; terminate if they do. Note that in general, this strategy might not terminate. First, the basic data theory may not be decidable, so checking formula equivalence cannot be automated. Second, the number of iterations needed may not be finite. Traditional global finite-state model checkers use this strategy, as do both [6] and [18]. In [6], the authors note that, even though Presburger arithmetic is decidable, their procedure may not terminate. In contrast, the restrictions in state predicates in [18] do guarantee termination.

The paper [12] restricts the allowed form of predicates mentioned in [6] so that the only basic comparisons allowed mirror those of [18], albeit for integers rather than real numbers. In this case, the iterative fixpoint calculation is guaranteed to terminate. This fact, together with the PES formulation of real-time model-checking, therefore suggests a novel symbolic approach to discrete-time model checking. Rather than expand

$\vee_1 \frac{\Phi \vdash \psi_1 \vee \psi_2}{\Phi \vdash \psi_1}$	$\vee_2 \frac{\Phi \vdash \psi_1 \vee \psi_2}{\Phi \vdash \psi_2}$	$\vee_3 \frac{\Phi \vdash \phi \vee \psi}{\Phi, \text{not}(\phi) \vdash \psi}$	$\vee_4 \frac{\Phi \vdash \psi \vee \phi}{\Phi, \text{not}(\phi) \vdash \psi}$
$\exists \frac{\Phi \vdash \exists x.\psi}{\Phi \vdash \psi[x := t]} (t \in \mathcal{D}\text{Exp})$	$\forall \frac{\Phi \vdash \forall x.\psi}{\Phi \vdash \psi[x := x']} (x' \text{ a fresh data variable})$		
$\text{Cut} \frac{\Phi \vdash \psi}{\Phi \vdash \psi \vee \phi ; \Phi, \phi \vdash \psi}$	$S \frac{\Phi, \phi \vdash \psi}{\Phi \vdash \text{not}(\phi) \vee \psi}$	$T \frac{\Phi, \phi \vdash \psi}{\Phi \vdash \psi}$	
$\wedge \frac{\Phi \vdash \psi_1 \wedge \psi_2}{\Phi \vdash \psi_1 ; \Phi \vdash \psi_2}$	$\vee \frac{\Phi, \phi_1 \vee \phi_2 \vdash_P \psi}{\Phi, \phi_1 \vdash_P \psi ; \Phi, \phi_2 \vdash_P \psi}$	$\square \square \frac{\Phi \vdash \psi[A_1][A_2]}{\Phi \vdash \psi[A_1 \triangleleft A_2]}$	
$[b] \frac{\Phi \vdash b[A]}{\Phi \vdash b\langle A \rangle} (b \in \mathcal{B}\text{Exp})$	$[-b] \frac{\Phi \vdash (-b)[A]}{\Phi \vdash \neg(b\langle A \rangle)} (b \in \mathcal{B}\text{Exp})$		
$[\vee] \frac{\Phi \vdash (\psi_1 \vee \psi_2)[A]}{\Phi \vdash \psi_1[A] \vee \psi_2[A]}$	$[\wedge] \frac{\Phi \vdash (\psi_1 \wedge \psi_2)[A]}{\Phi \vdash \psi_1[A] \wedge \psi_2[A]}$		
$[\exists] \frac{\Phi \vdash (\exists x.\psi)[A]}{\Phi \vdash \psi[x := t][A]} (t \in \mathcal{D}\text{Exp})$	$[\forall] \frac{\Phi \vdash (\forall x.\psi)[A]}{\Phi \vdash \psi[x := x'][A]} (x' \text{ a fresh data variable})$		
$C \frac{\Phi \vdash X}{\Phi \vdash \psi} (X = \psi \text{ an equation})$	$[C] \frac{\Phi \vdash X[A]}{\Phi \vdash \psi[A]} (X = \psi \text{ an equation})$		

Fig. 2. A Gentzen-like Proof System for PESs.

a discrete-time model into a CTS by “exploding” delays into sequence of clock ticks, mirror the definitions of timed-automata / real-time programs, albeit in the setting of integers, then use the symbolic approach here combined with the observation of [12].

Local Approaches. Significant attention has been paid to local, or on-the-fly, approaches to finite-state model checking. In the setting of BESs, this amounts to computing the solution of a single (propositional) variable rather than the values of all variables. In the case of data-based model checking, on-the-fly techniques have received little attention, although in the case of real-time model checking the subject is discussed in [27]. In the remainder of this section, we present a local model-checking framework for PESs that is based on a Gentzen-style, goal-directed proof system related to ones given in [5, 19].

The proof rules operates on *sequents* of the form: $\Phi \vdash \psi$, where $\Phi = \{\phi_1, \dots, \phi_n\}$ is a set of predicate-closed formulas, and ψ is a predicate. We interpret $\Phi \vdash \psi$ as the formula $\bigwedge \Phi \rightarrow \psi$. The rules for the proof system are given in Figure 2 and follow the following syntactic conventions: ϕ, ϕ_i are predicate closed, while ψ, ψ_i need not be; Φ, ϕ is short-hand for $\Phi \cup \{\phi\}$. Conclusions are also written above subgoals, which are separated by a “;”. Rules $\vee_1 - \wedge$ are familiar from the predicate calculus; note that instead of left- and right- rules for each construct as in [28], we rely on rule S combined with the fact that the not function “drives” negations inside. The remaining rules are for the substitution operator and predicate variables.

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

Definition 6. Let σ be a sequent of form $\Phi \vdash_P \psi$.

1. Let ϕ 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 ϕ wrt A is defined as

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

2. σ is a (successful) leaf if one of the following conditions holds. (a). $\psi \in \mathcal{BExp}$ or $\psi = \neg b$ for some $b \in \mathcal{BExp}$ (successful if $\llbracket \bigwedge \Phi \rightarrow \psi \rrbracket = \mathcal{D}^X$). (b). $\psi \in \Phi$ (always successful). (c). $\psi = X[A]$ (A may be empty) with parity p , and there is another sequent σ' of form $\Phi' \vdash_P X[A']$ on the path from the root node of the proof to σ with the property that no $\sigma'' \vdash_P X''[A'']$ such that X'' has parity different than p and X'' is defined in an earlier block in the PES than X , and $\text{post}(\Phi, A)$ logically implies $\text{post}(\Phi', A')$. Such a leaf is successful if the parity of X is ν .

A proof 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 2. *The proof rules in Figure 2 are sound: if $\Phi \vdash \psi$ has a valid proof wrt PES P then $\llbracket \Phi \rightarrow \psi \rrbracket_P = \mathcal{D}^X$.*

In general, the proof rules are not complete; proofs may require the *Cut* rule, and the data theory may not be expressive enough to define the necessary property. One must also be able to determine the validity of implications in the basic data theory. One may identify data theories for which completeness does hold. For example, we conjecture the following: If any predicate is semantically equivalent to a finite boolean combination of data predicates, the proof system for this data theory is complete.

7 On-the-Fly Real-Time Model Checking

This section shows how ideas from the previous section may be used to develop a novel efficient on-the-fly model-checking algorithm for real-time systems. The essential idea is to search for proofs of PESs using the proof rules given there.

Three key challenges exist for efficient proof search. The first involves computing implications in the basic data theory (here, the theory of clock constraints). For real-time model checking, efficient data structures for this problem have been proposed, including difference-bound matrices (DBMs) [14], constraint-decision diagrams (CDDs) [20] and clock restriction diagrams (CRDs) [31]. Because of ease of implementation, our prototype uses DBMs. The second challenge is to reuse sequents whose truth has been previously established, which we achieve by sequent caching. The third challenge is to devise derived proof rules from the generic ones to afford efficient proof generation. The generic rules are intended for arbitrary predicates; for specific applications, like real-time model checking, special forms of predicates predominate, and developing special-purpose proof rules can speed proof search significantly. In the case of real-time model checking, for example, quantifiers only appear in formulas of the form $\exists k. \phi[x_i := x_i + k]$ or $\forall k. \phi[x_i := x_i + k]$, where the x_i are clocks. Instead of using the generic rules for quantifiers, we use rules specialized for these formulas. Note that these rules are derived from the generic ones, and hence are guaranteed to be sound. This, plus an additional argument (omitted here) establishing the completeness of these derived rules for real-time, ensures the correctness of our algorithm.

A final observation is in order. A significant difficulty in automating proof search involves the *Cut* rule: automatically inferring the “cutting predicates” is non-obvious. In our approach, we defer the computation of these predicates by introducing placeholders for them and using a “backward” analysis of the proof tree to infer values for these placeholders. This strategy is inspired by the *splitting* technique used in [27].

We now give the derived rules used in our algorithm by first introducing two derived operators, where $\text{pre}(\phi, A) \stackrel{\text{def}}{=} \phi[A]$ is the weakest precondition operator.

- $\text{suc}_t(\phi) \stackrel{\text{def}}{=} \exists k. \text{post}(\phi, \bar{x} := \bar{x} + k)$, time successor of ϕ .

– $pre_t(\phi) \stackrel{\text{def}}{=} \exists k. pre(\phi, \bar{x} := \bar{x} + k)$, time predecessor of ϕ .

In the rules that follow, s, s' are placeholders.

$$\begin{array}{ll}
(1) \frac{\Phi \vdash_P \forall k. \psi[\bar{x} := \bar{x} + k]}{suc_t(\Phi) \vdash_P \psi} & (2) \frac{\Phi, s \vdash_P \forall k. \psi[\bar{x} := \bar{x} + k]}{suc_t(\Phi), s' \vdash_P \psi ; suc_t(\Phi \wedge s) \vdash_P suc_t(\Phi) \wedge s'} \\
(3) \frac{\Phi \vdash_P pre_t(\psi)}{suc_t(\Phi), s \vdash_P \psi ; \Phi \vdash_P pre_t(s)} & (4) \frac{\Phi, s \vdash_P pre_t(\psi)}{suc_t(\Phi), s' \vdash_P \psi ; s \vdash_P pre_t(s')} \\
(5) \frac{\Phi \vdash_P \psi[A]}{post(\Phi, A) \vdash_P \psi} & (6) \frac{\Phi, s \vdash_P \psi[A]}{post(\Phi, A), s' \vdash_P \psi ; s \vdash_P pre(s', A)} \\
(7) \Phi, s \vdash_P \varphi \begin{cases} \text{if } \Phi \rightarrow \varphi \text{ a tautology,} & s \stackrel{\text{def}}{=} \text{true} \\ \text{if } \Phi \rightarrow \varphi \text{ a contradiction,} & s \stackrel{\text{def}}{=} \text{false} \\ \text{otherwise} & s \stackrel{\text{def}}{=} \varphi \end{cases}
\end{array}$$

Rules (1)–(4) are specialized for real-time from the general quantifier elimination rules. Rules (5) and (6) are used for the time reset operation. Rule (7) is used to determine the values of placeholders. Recall that Φ, ϕ, φ are predicate-closed, while ψ need not to be.

The algorithm uses a generic, depth-first search technique, with caching; the proof rules above are used to generate sequents needed to be proved next in order for the goal sequent to be true. When a sequent is generated, the cache is first checked to see if it is implied by something in the cache; then the leaf is examined to see if it is a leaf, and if not, rules are then recursively applied to it. The same basic algorithm can easily be adapted to other settings by changing the proof rules uses.

Experimental Evaluation. We have implemented a prototype, which we call CWB-RT, of the abovementioned algorithm. The effort took approximately one month, with a week devoted to DBM implementations and the rest to building the proof-search infrastructure. C++ was used as the implementation language. To assess the performance of CWB-RT, we ran it on several examples taken from the literature and compared the results with those from the most recent available versions of Kronos (2.5i.2), UPPAAL (3.4.7, with both breadth-first (-b) and depth-first (-d) search options) and RED (5.3, with both forward and backward analysis). The experimental platform used was an Intel Pentium IV 2.8GHz with 2GB memory running Linux. The systems are listed below, together with properties (a) that should hold of correct implementations and properties (b) containing a bug that should not hold of correct implementations. The “formula bugs” include both logical errors and errors that could result from typographical mistakes (i.e. typing “2” rather than “1” by accident).

1. *Fischer’s timed Mutual Exclusion (MUX)* [1, 31]. We verify that (a). at any time, no more than one process is in its critical section. (b). at most four processes could be in their waiting states at the same time.
2. *FDDI token-ring mutual exclusion protocol* [13, 31]. We want to verify that (a). at any moment, at most one station is holding the token. (b). station i is in its asynchronous mode at time $20 * i$ of the network clock.
3. *Scheduling problem of real-time operating system (PATHOS)* [3, 31]. The property verified is that (a). no deadlines will be missed. (b). no new deadlines (2 units ahead of time) will be missed.
4. *Safeness of a leader-election algorithm (LEADER)* [31]. We check that (a). at any time there is at least one process who is a child to no other processes. (b). at any time there is at least three processes, each of which is a child to no other processes.

5. *Bounded liveness of a leader-election algorithm (LBOUND)* [31]. We verify that (a). after $2\lceil \log_2 m \rceil$ time units, where m is the number of processes, the algorithm will terminate. (b). after 3 time units, the algorithm will terminate.
6. *CSMA/CD benchmark* [31,32]. We check that (a). at any moment, at most one process is in the transmission mode for no less than 52 time units. (b). a third process could retry to send while two are already in the transmission status.

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 in two ways. First, for each buggy formula (b) and correct system specification, we collected comparative performance data for the model checkers in question. These figures in Table 1 indicate that CWB-RT performs much better than the other tools in this case.

Table 1. Performance data when correct systems fail buggy (i.e. (b)) properties. 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”.

Example	CWB-RT	Kronos 2.5i.2	UPPAAL 3.4.7 (-b)	UPPAAL 3.4.7 (-d)	RED 5.3 (forward)	RED 5.3 (backward)
MUX-20-b	7.83s	O/M	O/M	24.55s	O/M	O/M
MUX-40-b	372.81s	O/M	O/M	1139.57s	O/M	O/M
MUX-50-b	2653.00s	O/M	O/M	O/M	O/M	O/M
FDDI-30-b	0.20s	O/M	O/M	O/M	22.85s	15.96s
FDDI-40-b	0.58s	O/M	O/M	O/M	92.92s	78.57s
FDDI-60-b	2.76s	O/M	O/M	O/M	1788.43s	1053.06s
PATHOS-7-b	10.58s	O/M	O/M	O/M	O/M	3582.55s
PATHOS-8-b	48.32s	O/M	O/M	O/M	O/M	O/M
PATHOS-9-b	212.66s	O/M	O/M	O/M	O/M	O/M
LEADER-10-b	0.00s	O/M	O/M	O/M	21.32s	264.46s
LEADER-20-b	0.03s	O/M	O/M	O/M	O/M	O/M
LEADER-120-b	26.50s	O/M	O/M	O/M	O/M	O/M
LBOUND-10-b	0.01s	O/M	O/M	O/M	O/M	O/M
LBOUND-40-b	1.92s	O/M	O/M	O/M	O/M	O/M
LBOUND-120-b	284.42s	O/M	O/M	O/M	O/M	O/M
CSMA/CD-20-b	0.02s	O/M	6.11s	0.12s	O/M	O/M
CSMA/CD-40-b	0.15s	O/M	O/M	2.41s	O/M	O/M
CSMA/CD-100-b	3.81s	O/M	O/M	232.32s	O/M	O/M

We then studied situations in which correct formulas were used but buggy system specifications given. The data we obtained is given in Table 2, where the error for MUX originates in a misassignment to the global lock with the difference between the number of processes and the process identifier; the destination of the transition from the asynchronous state is misset to itself for the first station in FDDI; the error in PATHOS involves an omitted clock reset, which would be a typical programming error one might observe; and the error in CSMA/CD is caused by missing a collision signal, thus it leads to an incomplete system specification; the error in LBOUND is caused by setting the parent to NULL in the requester-responder pair, and to the identifier complemented by the number of processes in LEADER.

Again, the figures show that CWB-RT significantly outperforms the other tools on these case studies. We conjecture that CWB-RT’s superior performance in this and the

Table 2. Performance data for buggy system specifications and correct (i.e. (a)) properties.

Example	CWB-RT	Kronos 2.5i.2	UPPAAL 3.4.7 (-b)	UPPAAL 3.4.7 (-d)	RED 5.3 (forward)	RED 5.3 (backward)
MUX-14-e	1.32s	O/M	O/M	O/M	O/M	O/M
MUX-16-e	13.00s	O/M	O/M	O/M	O/M	O/M
MUX-18-e	257.02s	O/M	O/M	O/M	O/M	O/M
FDDI-30-e	0.24s	O/M	1.81s	2.54s	67.09s	14.15s
FDDI-40-e	0.70s	O/M	6.09s	9.39s	351.09s	39.37s
FDDI-60-e	3.16s	O/M	44.43s	63.26s	7066.18s	308.60s
PATHOS-5-e	0.51s	O/M	1.02s	109.56s	215.04s	24.33s
PATHOS-6-e	19.71s	O/M	354.40s	O/M	O/M	250.64s
PATHOS-7-e	2283.13s	O/M	O/M	O/M	O/M	O/M
LEADER-60-e	0.02s	O/M	21.18s	21.04s	O/M	O/M
LEADER-70-e	0.03s	O/M	O/M	O/M	O/M	O/M
LEADER-150-e	0.26s	O/M	O/M	O/M	O/M	O/M
LBOUND-10-e	0.00s	O/M	O/M	62.33s	O/M	O/M
LBOUND-20-e	0.02s	O/M	O/M	O/M	O/M	O/M
LBOUND-120-e	1.16s	O/M	O/M	O/M	O/M	O/M
CSMA/CD-10-e	65.19s	O/M	O/M	O/M	2057.94s	2389.87s
CSMA/CD-11-e	200.50s	O/M	O/M	O/M	O/M	O/M
CSMA/CD-12-e	670.95s	O/M	O/M	O/M	O/M	O/M

preceding case 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 devoted to compute a fixpoint before it could find an error.

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-RT on all (a) properties for correct versions of the 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-RT generally outperforms Kronos and is often better, though sometimes worse, than UPPAAL3.4.7. RED5.3 generally outperforms CWB-RT on these examples, although it should be noted that while Kronos [32] was implemented with DBMs, as CWB-RT is, UPPAAL [4] use CDDs and RED5.3 CRDs [31]. We conjecture that CWB-RT would see considerable performance improvement if we used CDDs / CRDs in place of DBMs. Also, CWB-RT’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 Conclusions

In this paper we have presented predicate equation systems (PESs) as a generic model-checking framework for data-based systems. We illustrated the flexibility of PESs by showing model-checking problems for real time may be captured uniformly using them, and we developed generic global and local approaches to computing solutions of PESs. Finally, we developed a new model-checking algorithm for real-time based on proof search in the setting of PESs and gave experimental data showing that the algorithm is competitive with, and often superior to, existing approaches.

Table 3. Performance data for correct systems and properties (i.e. (a) properties).

Example	CWB-RT	Kronos	UPPAAL	UPPAAL	RED 5.3	RED 5.3
		2.5i.2	3.4.7 (-b)	3.4.7 (-d)	(forward)	(backward)
MUX-5-a	0.23s	0.48s	0.77s	4.12s	4.67s	1.36s
MUX-6-a	4.03s	O/M	68.87s	927.79s	66.89s	3.92s
MUX-7-a	115.53s	O/M	O/M	O/M	778.48s	10.32s
FDDI-20-a	0.21s	O/M	O/M	O/M	2.02s	2.25s
FDDI-40-a	2.29s	O/M	O/M	O/M	16.91s	24.39s
FDDI-60-a	11.03s	O/M	O/M	O/M	60.07s	85.99s
PATHOS-4-a	4.19s	O/M	0.21s	0.14s	10.15s	6.07s
PATHOS-5-a	2824.96s	O/M	2.14s	55.27s	353.98s	360.06s
PATHOS-6-a	O/M	O/M	O/M	O/M	12053.26s	31190.21s
LEADER-6-a	0.24s	O/M	1.32s	1.53s	0.43s	1.28s
LEADER-7-a	12.74s	O/M	136.29s	142.02s	1.18s	3.73s
LEADER-8-a	1888.35s	O/M	O/M	O/M	2.97s	9.80s
LBOUND-6-a	0.35s	O/M	2.53s	1.64s	67.70s	33.17s
LBOUND-7-a	15.22s	O/M	145.86s	153.59s	453.58s	193.68s
LBOUND-8-a	2431.69s	O/M	O/M	O/M	2933.81s	892.97s
CSMA/CD-6-a	3.89s	0.32s	2.55s	5.15s	709.12s	0.52s
CSMA/CD-7-a	56.62s	O/M	218.81s	182.49s	12109.23s	1.26s
CSMA/CD-8-a	1584.76s	O/M	O/M	O/M	O/M	3.15s

Related Work. Both [23] and [25] propose model checkers for the value-passing mu-calculus, although these algorithms only work for finite systems. Groote *et al.* [16] define first-order boolean equation systems, which are very similar to our PESs. That work does not consider proof systems or on-the-fly algorithms, however, and it did not study the real-time model-checking problem. Tableau methods [26, 17] can be cast into a local algorithms for PESs in a way similar to [22]. Finally, [27] also gives an on-the-fly algorithm for model-checking real-time systems and [15] proposes a computational framework based on logic programming for verifying real-time systems. Our method is 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 that one for checking safety properties.

References

1. 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 *RTSS92*, 1992.
2. H. R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1), 1994.
3. F. Balarin. Approximate reachability analysis of timed automata. In *IEEE RTSS96*, 1996.
4. G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi. Uppaal implementation secrets. In *FTRFT02*, 2002.
5. S. Berezin. *Model Checking and Theorem Proving: a Unified Framework*. PhD thesis, Carnegie Mellon University, 2002.
6. T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using presburger arithmetic. In O. Grumberg, editor, *CAV97*, 1997, LNCS 1254, Springer-Verlag.

7. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the 5th IEEE Symposium on Logic in Computer Science*, pages 428–439, Philadelphia, PA, 1990.
8. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logic of Programs*, Yorktown Heights, LNCS 131, pages 52–71, 1981.
9. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
10. R. Cleaveland and J. Riely. Testing-based abstractions for concurrent systems. In *CONCUR94*, LNCS 836, pages 417–432, 1994.
11. R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In *CAV91*, LNCS 575, pages 48–58, 1991.
12. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV98*, LNCS 1427, pages 268–279. Springer-Verlag, 1998.
13. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool kronos. In *DIMACS Workshop on Verification and Control of Hybrid Systems*, LNCS 1066. Springer-Verlag, 1995.
14. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *CAV89*. LNCS 407, 1989.
15. X. Du, C.R. Ramakrishnan, and S.A. Smolka. Tabled resolution + constraints: A recipe for model checking real-time systems. In *RTSS00*, 2000.
16. J.F. Groote and T.A.C. Willemse. A checker for modal formulas for processes with data. Technical report, Technische Universiteit Eindhoven, The Neitherlands, 2002.
17. M. Hennessy and X. Liu. A modal logic for message passing processes. *Acta Informatica*, 32(4):375–393, 1995.
18. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2), 1994.
19. J. Bradfield and C. Stirling. Local model checking for infinite state spaces. *Theoretical Computer Science*, 96:157–174, 1992.
20. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Software Tools for Technology Transfer*, 1:134–152, 1997.
21. H. Lin. Symbolic graphs with assignment. In *CONCUR96*, LNCS 1119, pages 50–65, 1996.
22. A. Mader. *Verification of Modal Properties Using Boolean Equation Systems*. PhD thesis, München, Techn-Univ., 1997.
23. R. Mateescu. Local model-checking of an alternation-free value-based modal mu-calculus. In *VMCAI98*, September 1998.
24. J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proceedings of the International Symposium in Programming*, LNCS 137, Springer-Verlag, 1982.
25. C.R. Ramakrishnan. A model checker for value-passing mu-calculus using logic programming. In *PADL01*, LNCS 1990, Las Vegas, Nevada, March 2001. Springer-Verlag.
26. J. Rathke and M. Hennessy. Local model checking for value-passing processes. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Software (TACS '97)*, LNCS 1281, Springer-Verlag, 1997.
27. O. Sokolsky and S. Smolka. Local model checking for real-time systems. In P. Wolper, editor, *CAV 95*, LNCS 939, Liège, Belgium, July 1995.
28. A. Szalas. Logic for computer science. lecture notes. URL <http://www.ida.liu.se/~andsz>.
29. A. Szalas. On natural deduction in first-order fixpoint logics. *Fundamenta Informaticae*, 26:81–94, 1996.
30. L. Tan and R. Cleaveland. Evidence-based model checking. In *CAV*, LNCS 2404, 2002.
31. F. Wang. Efficient verification of timed automata with bdd-like data-structures. In *VMCAI 2003*, LNCS 2575, pages 189–205, 2003.
32. S. Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1:123–133, 1997.