Model Checking for Data-Based Concurrent Systems

A Dissertation Presented

by

Dezhuang Zhang

to

The Graduate School

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in

Computer Science

Stony Brook University

December 2005

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 3206503

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform 3206503

Copyright 2006 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

State University of New York

at Stony Brook

The Graduate School

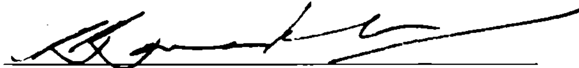
Dezhuang Zhang

We, the dissertation committee for the above candidate for

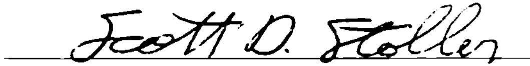
the degree of Doctor of Philosophy,

hereby recommend acceptance of this dissertation.

Professor Rance W. Cleaveland, Advisor
Computer Science Department



Professor C. R. Ramakrishnan, Chairman of Defense
Computer Science Department



Professor Scott Stoller
Computer Science Department

Professor Oleg Sokolsky
Department of Computer and Information Science
University of Pennsylvania

This dissertation is accepted by the Graduate School.

Graduate School

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abstract of Dissertation

Model Checking for Data-Based Concurrent Systems

by

Dezhuang Zhang

Doctor of Philosophy

in

Computer Science

Stony Brook University

2005

This dissertation introduces *predicate equation systems* (PESs) as a uniform symbolic basis for model checking of data-based concurrent systems. In contrast with the finite-state concurrent systems that most model-checking research is directed toward, data-based concurrent systems use data variables that may draw values from infinite sets. PESs generalize first-order logic by adding capabilities for recursively-defined predicates, and may be seen as a first-order generalization of the well-studied *boolean equation systems* used in finite-state model checking.

The dissertation also introduces a goal-directed, Gentzen-like proof system for proving PES formulas and shows how it may be used to define on-the-fly model checkers for data-based model-checking problems. Then the theory is used to develop model checkers for different data-based model-checking problems: real-time model checking, in both traditional and parametric forms; model-checking for Presburger systems, which feature the use of integer variables; and temporal-logic query checking for Presburger systems. In each case, implementations are presented, and extensive experimental data collected to compare these algorithms with existing approaches, when such exist. The general proof-search approach given here generally outperforms, in often startling fashion, the specialized routines found in the literature for these problems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To my wife Yue and our son Matthew with love

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Contents

Acknowledgements **ix**

1 Introduction **1**

1.1 Predicate Equation Systems 2

1.2 The First-Order Modal Mu-Calculus 4

1.3 Real-Time Model Checking Problems 4

1.4 Temporal-logic Query Checking 7

2 Fixpoint Equation Systems **10**

2.1 Lattices and Fixpoints 10

2.2 Fixpoint Equation Systems 12

2.3 Boolean Equation Systems 14

3 Predicate Equation Systems **16**

3.1 Basic Data Theories 16

3.2 The Predicate Calculus 18

3.3 Predicate Equation Systems 19

3.4 Global Approaches to Tautology Checking 21

3.5 A Gentzen-Like Proof System 22

4 Transition Systems and the Modal Mu-Calculus **28**

4.1 Concrete Transition Systems 28

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2 The First-Order Modal Mu-Calculus 29
4.3 Symbolic Transition Graphs 32
4.4 From Model Checking to PESs 34
4.5 Finite-State Model Checking with PESs 38

5 Real-Time Model Checking with PESs 40
5.1 Parametric Timed Automata 40
5.2 The Real-Time Modal Mu-Calculus 43
5.3 From Real-Time Model Checking to PESs 45
5.4 On-the-Fly Real-Time Model Checking 48
5.5 Implementation 56
5.6 Experimental Results for Real-Time 65
5.7 Experimental Results for Parametric Real-Time 72

6 Model Checking Presburger Systems with PESs 78
6.1 Presburger Systems 78
6.2 The Presburger Modal Mu-Calculus 80
6.3 From Presburger Model Checking to PESs 82
6.4 Local Model Checking 83
6.5 Implementation and Performance Evaluation 85

7 Temporal-Logic Query Checking for Presburger Systems 86
7.1 A Simple Example 87
7.2 Existential Query Checking 88
7.3 Universal Query Checking 91
7.4 Implementation 92
7.5 Case Study : A Simple Thermostat 95
7.6 Performance Comparisons 98

8 Conclusion and Future Work 103

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

1 A Presburger system 8

2 A Gentzen-like proof system for PESs. 24

3 A simple symbolic transition graph 34

4 Translation function for PESs 36

5 The relationship between the two semantic functions 37

6 A parametric timed automaton with two clocks 42

7 Clock regions 47

8 The graphic representation of $pre_t(\phi)$ and $suc_t(\phi)$ 49

9 A local approach for parametric real-time model checking. . . 50

10 Example for rule $\vee$ 51

11 Example for rule $\exists_1$ 52

12 Examples for lemma 5.4.1 53

13 Representation of a clock zone 58

14 Example for CRD with upper bounds 61

15 A local approach for Presburger systems. 83

16 A simple transition graph 87

17 Automaton for $x - y \leq 0$ 94

18 SCR specification of a simple thermostat 96

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Tables

1 Pseudo-code for local real-time model-checking algorithm . . . 63
2 Pseudo-code for local real-time model-checking algorithm (cont) 64
3 Non-parametric real-time performance data when correct systems fail buggy (b) properties. 68
4 Non-parametric real-time performance data for buggy system specifications and correct (a) properties. 69
5 Non-parametric real-time performance data for correct systems and (a) properties. 70
6 Parametric real-time performance data with (a) conditions. . . 75
7 Parametric real-time performance data with (c) conditions. . . 76
8 Parametric real-time performance data with (b) conditions. . . 77
9 Query checking performance comparison with model checking. 99
10 Query checking performance comparison with ALV-0.3. 101
11 Query checking performance comparison with ALV-0.3 for buggy properties. 102

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgements

First of all, I want to thank my great mentor, Prof. Rance Cleaveland, for his support during my doctoral research. You give me enough freedom, wisdom and offer me valuable guidance to explore the research subject. The days we spend over the corner of the large dining table with "pencil and paper" in your friendly and spacious house, is one of the happiest days that I have ever had. I have enjoyed our meeting and learned a lot from the rigorous way you think, the encouraging way you speak and the elegant way you write. Not enough thanks can be given to you!

I also want to thank Prof. Eugene W. Stark, Prof. C. R. Ramakrishnan, Prof. Scott A. Smolka, Prof. Scott Stoller, Prof. Oleg Sokolsky and Prof. Tevfik Bultan for their (funding) help.

I would like to thank Edwina Osmanski for her enthusiastic service.

My thanks also go to my friends in Stony Brook : Fuxiang Yu, Haodong Hu, Dongdong Ge, Ping Yang, Bikram Sengupta and many others. I enjoyed the discussion with Fuxiang about math problems. The computing resource contributed by Haodong helped me to provide experimental results in time.

Special thanks to my wife Yue Gao. Without your consistent love, support, understanding, encouragement and patience, I would not have been able to complete this program. I especially want to thank my parents-in-law and my parents. Without their expectation and encouragement, I would have never been here to present this dissertation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

Introduction

Temporal-logic model checkers [36, 38, 87] 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 classes of real-time systems and systems involving integer-valued variables. (Of course, for arbitrary systems involving integers, model checking is not decidable.) A number of different temporal logics have also been studied, including LTL [74], CTL [37], CTL* [52] and the modal mu-calculus [72].

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 [11, 43] 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 [33]. It has also served as a basis for new algorithms, including efficient on-the-fly model-checkers for the mu-calculus [11] and symbolic algorithms based on Gaussian elimination [78], and algorithm optimizations, e.g. [23, 43, 58, 77, 102] etc.

The motivation of this dissertation is to develop a similar framework for

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

model checking of systems that manipulate values and thus may not be finite-state. The main results obtained are described below.

1.1 Predicate Equation Systems

This dissertation develops *predicate equation systems* (PESs) as a uniform basis for verifying data-based systems [111]. PESs generalize boolean equation systems to full first-order logic and may be seen as an extension of the predicate calculus with recursively-defined predicates. We show how PESs may be used to encode model-checking problems, including those for Presburger systems [32] and real-time model checking [64], may be cast in terms of PESs, and discuss generic model-checking techniques that immediately follow from the recursive form of PESs.

We also define a goal-directed, Gentzen-like proof system for establishing that formulas defined in the context of a PES are valid (i.e. are tautologies). This proof system is shown to provide a generic basis for *on-the-fly* model checking of data-based systems.

Related Work A number of model checking frameworks have been proposed for (infinite-state) data-based systems. These various approaches can be characterized along several axes.

Boolean equation systems (BESs) have received a lot of attention since the model checking algorithms with CTL [37] and with modal mu-calculus [53] were introduced. A number of finite-state model-checking algorithms were developed directly over BESs, including [11, 24, 40, 43, 76, 78, 82, 100]. The first-order boolean equation systems [57, 59] is used for model-checking infinite-state value-passing systems. Predicate equation systems provide a more general framework in the sense to encode real-time model checking, and we focus on algorithmic issues and are devoted to develop new applications, e.g. query

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

checking.

Logic programming has been used to solve model-checking problems e.g. see [88, 89] etc. A logic program is a sequence of clauses, called *Horn clauses*, each of which has the form $A \leftarrow B_1 \wedge \ldots \wedge B_n$ where $B_1, \ldots, B_n$ are atomic formulas. A constraint logic program (CLP) [67, 68] is a first order extension of logic program. A constraint is a finite conjunction of atomic formulas built on a given set of constraint constructors. Constraints will be interpreted over a fixed domain and handled via a *constraint solver*. The least model of a CLP program can be defined as the least fixpoint of an operator that computes the direct logical consequences of the program and of a given set of atomic formulas. Several papers [48, 51, 55] have demonstrated the potentiality of CLP as a symbolic model checker for infinite-state systems. Both PESs and CLPs provide the capability for fixpoint computations. CLPs use resolution based method while PESs allows us to algebraically reason the model checking problem.

Local model checking tries to avoid constructing the global state space of the system, and access as few states as possible and only build fragments of the state space as needed. Algorithms in this category include tableau-based model-checking procedures for infinite-state systems [12, 69], value-passing systems [90], deductive model checking (see e.g. [26, 79, 95]) and attempts to combine theorem prover and model checking (see e.g. [22, 84]). These works consider the general termination condition for fixpoint computations and provides relative completeness. We have also provided a novel Gentzen-like proof system for PESs which could be customized for different applications.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.2 The First-Order Modal Mu-Calculus

This thesis also develops a first-order generalization of the modal mu-calculus [72] and presents general strategies for translating model-checking problems for this logic into PESs. It is also shown how existing temporal logics for data-based systems, including the real-time modal mu-calculus [96] and Presburger CTL [32], may be translated into this logic.

1.3 Real-Time Model Checking Problems

Real-time model checking [2, 7, 64] 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 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.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We show how the general framework PESs may be used for both non-parametric real-time model checking and checking a parameterized model against a parameterized formula. We call the latter problem the *universal parametric real-time model-checking* problem [112] because, in contrast with other work on parametric real-time systems, our interest consists 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.

Related Work 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 [9], 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 [5, 13, 16, 30, 31, 54, 66, 103, 104, 106]. 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, [31] showed that the model checking of parametric timed CTL is undecidable over timed automata with only one parametric clock. The *optimization* problem, i.e.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

find the optimal valuations of parameters according to some given criteria, is considered in [5, 107] etc.

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 given, 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.

Our local parametric real-time model-checking algorithm encodes the model-checking problem with 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 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 to detect errors 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 dissertation.

Among existing real-time model-checking works, the algorithms of [96, 97] are most related to ours in the sense that theirs also works in a forward /

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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 required. 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.

1.4 Temporal-logic Query Checking

Temporal-logic query checking [34] has emerged as a useful extension to model checking for supporting requirements and design understanding. The query-checking problem may be formulated as follows: given a model and a temporal logic formula with placeholders (i.e. a *query*), compute a set of assignments of formulas to placeholders such that the resulting temporal formula is satisfied by the given model. For example, solving the CTL query $\mathsf{AG}\ ?_X$, where $?_X$ is a placeholder, for the strongest formula making the query true yields the invariant $(2 \leq x \leq 5) \wedge (3 \leq y \leq 8)$ for the *Presburger* system in Figure 1. In the figure, the system is given as a state machine that can modify and test the values of integer variables x, y. Each transition includes a conditional guard determining whether or not the transition may fire; and ";"; and an optional update action to be performed when the transition fires. The "start state" arrow also contains the initial conditions on the values of x, y; here, $x = 2$ and $y = 3$ when the system begins execution.

Temporal-logic query checking has proved valuable as a means for model understanding. For example, given an early attempt at a specification for a system, one would want to validate some desired temporal-logic properties with a model checker. Some of the properties might fail to hold, in which

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

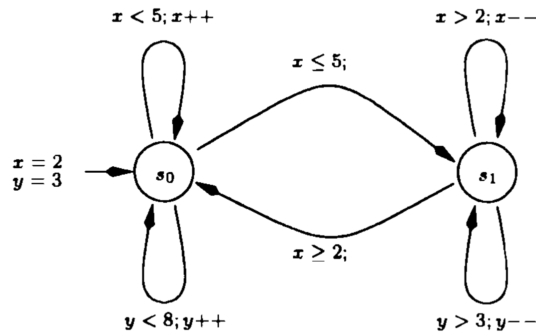


Figure 1: A Presburger system

case one might infer either that the specification requires revision or that the properties are faulty. To determine which situation holds, one can modify formulas into queries in order to retrieve the strongest formulas that makes the query true, and obtain more diagnostic information to help improve the design. Even if a property is proved to hold in the model, one can still use a query checking to obtain much stronger properties and thus understand more precisely the behavior of the system.

We develop query-checking techniques for a class of system models that use integer-valued variables (so-called *Presburger* systems, in which Presburger formulas are used to define system behavior) [110]. Our method uses the symbolic model-checking technique that relies on proof search. Solutions to a placeholder are inferred at the leaves of a proof tree in order to ensure that the resulting proof is valid.

The principal contributions of our query-checker, which we call CWB-QC (Concurrency Workbench [41] - Query Checking), are the following.

(1) CWB-QC is the first query checker for the class of (infinite-state) Presburger systems. Existing query checkers [61] only deal with finite-state systems. With CWB-QC, formulas and systems can manipulate integer-valued variables and may thus be infinite-state.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(2) Our solution focuses on the *existential* ("find a solution") query-checking problem, as opposed to the *universal* one ("find all solutions"). The latter problem is the usual one studied, but its double exponential time complexity limits its application [29, 61]. However, the applications of query checking that are most often cited [61], existential query checkers can equally well be used, and at much lower computational cost.

(3) Our existential query checker runs as fast as our model checker, and faster with more precise results than the Action Language Verifier [17], the state-of-the-art model checker for Presburger systems.

Related Work As originally proposed by Chan [34], query checking concentrated on valid queries, i.e. queries that always have a unique strongest solution for every system. Recent work has extended this seminal research in several ways. Bruns and Godefroid [29] studied how to adapt the automata-theoretic model-checking approach to solve the query-checking problem. Gurfinkel *et al.* [61] enriched the query language with multiple placeholders and implemented query checking using a multi-valued model checker. The problem of deciding whether a given query has a unique strongest solution over a given system and how to compute this solution is studied in [65]. The valid-query problem is revisited by [92]. All these works focus on propositional temporal logic and finite-state systems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

Fixpoint Equation Systems

This chapter introduces a general account of fixpoint equation systems [100] over complete lattices.

2.1 Lattices and Fixpoints

Let Ω be a set and $\sqsubseteq \subseteq \Omega \times \Omega$ be partial order on Ω, where a partial order is a reflexive, antisymmetric and transitive relation. Then $\langle \Omega, \sqsubseteq \rangle$ is a lattice if every pair of elements $p, q \in \Omega$ has a greatest lower bound $p \sqcap q \in \Omega$ and a least upper bound $p \sqcup q \in \Omega$. If for every subset $S \sqsubseteq \Omega$, there exists a least upper bound $\sqcup S$ and a greatest lower bound $\sqcap S$, $\langle \Omega, \sqsubseteq \rangle$ is called *complete lattice*. Note that every complete lattice has a maximum element $\top = \sqcup \emptyset$ and minimum element $\bot = \sqcap \emptyset$, and that every finite lattice is complete.

A function $\phi : \Omega \rightarrow \Omega$ is called *monotone* if whenever $q \sqsubseteq q'$ then $\phi(q) \sqsubseteq \phi(q')$. It is *continuous* if for every subset $S \in \Omega$, $\phi(\sqcup S) = \sqcup \phi(S)$. An element $q \in \Omega$ is a *fixpoint* of ϕ if $\phi(q) = q$.

Let $\langle Q, \sqsubseteq \rangle$ be a complete lattice, according to Knaster-Tarski Fixpoint Theorem [101], every monotonic function $\phi \in \Omega^{\Omega}$ has a unique least fixpoint $\mu\phi \in \Omega$ and greatest fixpoint $\nu\phi \in \Omega$ defined by,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\mu\phi = \sqcap\{q \in \Omega \mid \phi(q) \sqsubseteq q\}$$
$$\nu\phi = \sqcup\{q \in \Omega \mid q \sqsubseteq \phi(q)\}$$

The greatest and least points of a continuous function ϕ over a complete lattice may be characterized as an infinite conjunction and disjunction of approximants respectively,

$$\nu\phi = \sqcap_{i=0}^{\infty}\phi_i$$
$$\mu\phi = \sqcup_{i=0}^{\infty}\hat{\phi}_i$$

where,

$$\phi_0 = \top$$
$$\phi_{i+1} = \phi(\phi_i)$$
$$\hat{\phi}_0 = \bot$$
$$\hat{\phi}_{i+1} = \phi(\hat{\phi}_i)$$

Let $\langle \Omega, \sqsubseteq \rangle$ be a complete lattice and $\mathcal{X}$ be a finite set of *variables*. The set $\Omega^{\mathcal{X}}$ consists of all functions mapping $\mathcal{X}$ to Ω. We call a function $\theta \in \Omega^{\mathcal{X}}$ as an environment over $\mathcal{X}$. Then $\Omega^{\mathcal{X}}$ represent the set of all environments over $\mathcal{X}$. Environments constitute a complete lattice under the pointwise extension of $\sqsubseteq$ to $\Omega^{\mathcal{X}}$: $\theta \sqsubseteq \theta'$ if and only if for all $X \in \mathcal{X}, \theta(X) \sqsubseteq \theta'(X)$.

We assume that if $\theta \in \Omega^{\mathcal{X}}$ and $\theta \in \Omega^{\mathcal{X}'}$ then $\mathcal{X} = \mathcal{X}'$, and we write $\mathrm{dom}(\theta) = \mathcal{X}$ for the *domain* of θ. We sometimes write $\Omega^{\mathcal{X}}$ as $\mathcal{X} \to \Omega$. If $\theta \in \Omega^{\mathcal{X}}$ and $\theta' \in \Omega'^{\mathcal{X}'}$, then $\theta[\theta']$ represents the function in $(\Omega \cup \Omega')^{(\mathcal{X} \cup \mathcal{X}')}$ defined as follows.

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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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

2.2 Fixpoint Equation Systems

Syntax An *equation block* B is a set of equations $\{X_1 = f_1, \ldots, X_l = f_l\}$, where f_i are monotonic functions with type $\Omega^{\mathcal{X}} \rightarrow \Omega$. Variables $X_i \in \mathcal{X}$ and are distinct. We use $\mathsf{lhs}(B)$ to denote the left-hand side variables in block B, and $\mathsf{rhs}(X_i)$ to refer to the right-hand side of the equation whose left-hand side variable is X_i. Function $\mathsf{vars}(f_i)$ denotes the set of free variables in f_i. We define $\mathsf{vars}(B) = \mathsf{lhs}(B) \cup \bigcup_{i=1}^{l} \mathsf{vars}(f_i)$ as the variables in equation block B and refer to variables in $\mathsf{lhs}(B)$ as *bound* and variables in $\mathsf{vars}(B) - \mathsf{lhs}(B)$ as *free*.

A *parity block* E has the form $\langle p, B\rangle$, where $p \in \{\mu, \nu\}$ is a parity indicator and B is an equation block. We lift the notions $\mathsf{lhs}, \mathsf{rhs}, \mathsf{vars}$, free variable and bound variable to parity block in the straightforward manner.

A *fixpoint equation system* is a nonempty sequence $\mathsf{E} = E_1 \ldots E_m$ of parity blocks whose left-hand sides are pairwise disjoint. If E' is an equation system and E is a parity block whose left-hand side variables are disjoint from those in E' then we write $E :: \mathsf{E}'$ for the equation system obtained by adding E to front of E'. We use $\mathsf{E}^k = E_k E_{k+1} \ldots$ to refer to the subsequence of E starting from the k-th parity block. Operations $\mathsf{lhs}, \mathsf{rhs}, \mathsf{vars}$, free variable and bound variable are generalized in the straightforward manner. We call E as *closed* if every $X \in \mathsf{vars}(\mathsf{E})$ is bound, i.e. an element of $\mathsf{lhs}(\mathsf{E})$.

Semantics We first consider the semantics of the p-blocks $B = \{X_1 = f_1, \ldots, X_l = f_l\}$. Let $\mathcal{X}' = \{X_1, \ldots, X_l\}$ and $\theta \in \Omega^{\mathcal{X}}$. Define a function $f_{B,\theta} : \Omega^{\mathcal{X}'} \rightarrow \Omega^{\mathcal{X}'}$ mapping environments over $\mathcal{X}'$ to environments over $\mathcal{X}'$ as

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

follows.

$$f_{B,\theta}(\theta') = (X_1 := f_1(\theta[\theta']), \cdots, X_l := f_l(\theta[\theta'])) \tag{1}$$

Intuitively, $f_{B,\theta}(\theta')$ returns an environment over $\mathfrak{X}'$ in which each X_i is mapped to the result returned by evaluating f_i on environment $\theta[\theta']$. Note that we use $\theta[\theta']$ to denote the environments updated by θ'. It follows from the monotonicity of the f_j that for any θ, $f_{B,\theta}$ is a monotonic function over $\mathcal{Q}^{\mathfrak{X}'}$. Tarski's fixpoint theorem then ensures the existence of least and greatest fixpoints, $\mu f_{B,\theta}$ and $\nu f_{B,\theta}$, which are environments over $\mathfrak{X}'$. Given $\theta \in \mathcal{Q}^{\mathfrak{X}}$, we define the semantics of a parity block in terms of these fixed points: $[\![\langle p, B\rangle]\!]\theta = pf_{B,\theta}$. So $[\![\langle p, B\rangle]\!]$ maps environments over $\mathfrak{X}$ to environments over $\mathfrak{X}'$, where $\mathfrak{X}'$ consists of the left-hand side variables in B.

For the semantics of an fixpoint equational system E, given an environment θ, we define a function $f_{\mathsf{E},\theta} : Q^{\mathfrak{X}} \rightarrow Q^{\mathfrak{X}}$ inductively on the structure of E and the above block semantic function $f_{B,\theta}$. When E contains a single parity block, we take $f_{\mathsf{E},\theta} = f_{B,\theta}$ and define $[\![\mathsf{E}]\!]\theta = [\![\langle p, B\rangle]\!]\theta$. When E contains more than one blocks, it may be written as $\mathsf{E} = \langle p, B\rangle :: \mathsf{E}'$, where E' is also an equational system. In this case $f_{\mathsf{E},\theta}$ is defined as

$$f_{\mathsf{E},\theta}(\theta') = ([\![\mathsf{E}']\!](\theta[\theta']))[f_{B,\theta[[\![\mathsf{E}']\!](\theta[\theta'])]}(\theta'\lceil\mathfrak{X}_B)] \tag{2}$$

where $\mathfrak{X}_B = \mathsf{lhs}(B)$. Intuitively, this function may be understood by inspecting its subexpressions. $[\![\mathsf{E}']\!](\theta[\theta'])$ is the environment over $\mathfrak{X}_{\mathsf{E}'}$ defined by E' in environment θ updated with bindings contained in θ'. This environment assigns a "fixpoint value" to every left-hand variable in E'. $f_{B,\theta[[\![\mathsf{E}']\!](\theta[\theta'])]}$ is the function on environments defined by block B and the environment obtained by updating θ with the bindings in E'. $\theta'\lceil\mathfrak{X}_B$ is the sub-environment of θ' obtained by restricting variables to those that appear as left-hand sides in B.

One may then evaluate $f_{\mathsf{E},\theta}(\theta')$ as follows.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(1) Update the global environment θ with bindings contained in θ'.

(2) Compute the meaning of E' in this new global environment to obtain new bindings for the left-hand side variables in E'.

(3) Update θ with these new bindings.

(4) Evaluate $f_{B,\dots}$ with respect to this new global environment and the bindings, using as input the bindings for the left-hand side variables of B that are given in θ'.

It is easy to show that $f_{\mathsf{E},\theta}(\theta')$ is monotonic over the lattice $\Omega^{\mathcal{X}}$ and hence has unique least and greatest fixpoints. We then define $[\![\mathsf{E}]\!]\theta$ as follows.

$$[\![\langle p, B\rangle :: \mathsf{E}']\!]\theta = p f_{\langle p,B\rangle::\mathsf{E}',\theta}$$

If E is closed then for any θ, θ' we have that $[\![\mathsf{E}]\!]\theta = [\![\mathsf{E}]\!]\theta'$. In this case we often omit reference to θ and write $[\![\mathsf{E}]\!]$ for this (unique) environment.

2.3 Boolean Equation Systems

As an example, we consider the Boolean equation systems defined over the Boolean lattice $\langle \mathbf{0}, \mathbf{1}, \sqsubseteq\rangle$, where $\mathbf{0}$ and $\mathbf{1}$ are the boolean values "false" and "true", respectively, with $\mathbf{0} \sqsubset \mathbf{1}$. In this setting environments may be viewed as characteristic functions of subsets of $\mathcal{X}$, so we allow the use of the standard set operators $\cup$, $\cap$, and $-$ on such environments. The right-hand sides of equations are the formulas given by the following, where $\mathcal{X}' \subseteq \mathcal{X}$.

$$f := \bigvee \mathcal{X}' \mid \bigwedge \mathcal{X}'$$

We often write $\mathbf{tt}$ for $\bigwedge \emptyset$ and $\mathbf{ff}$ for $\bigvee \emptyset$. The definition of $[\![f]\!]\theta$ is standard: $[\![\bigvee \mathcal{X}']\!]\theta = \mathbf{1}$ iff $\mathcal{X}' \cap \theta \neq \emptyset$, and $[\![\bigwedge \mathcal{X}']\!]\theta = 1$ iff $\mathcal{X}' \subseteq \theta$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

As we have pointed, Boolean equation systems have been widely used to develop and optimize algorithms for finite-state model checking. Refer to Mader's thesis [78] for an introduction on Boolean equation systems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

Predicate Equation Systems

Predicate equation systems consist of systems of simultaneous equations whose right-hand sides are first-order formulas. This chapter defines predicate equation systems and develops a Gentzen-like proof system.

3.1 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 3.1.1 *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* $\langle \mathcal{B}Exp, \mathcal{D}Exp, fv, \langle - \rangle, \models, |-| \rangle$, *where:*

1. $\mathcal{B}Exp$ *is a set of* data predicates*;*

2. $\mathcal{D}Exp$ *is a set of* data expressions*;*

3. $fv : (\mathcal{B}Exp \cup \mathcal{D}Exp) \rightarrow 2^{\mathcal{X}}$ *is the* free-variable mapping*;*

4. $\langle - \rangle : (\mathcal{B}Exp \cup \mathcal{D}Exp) \times \mathcal{D}Exp^{\mathcal{X}} \rightarrow (\mathcal{B}Exp \cup \mathcal{D}Exp)$ *is the* substitution function *(notation:* $b\langle f \rangle$ *for* $\langle - \rangle(b, f)$*);*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5. $\models \subseteq \mathcal{D}^{\mathcal{X}} \times \mathcal{B}Exp$ *is the* interpretation relation *(notation: $\rho \models b$ for $\models (\rho, b)$);*

6. $|-| : \mathcal{D}Exp \times \mathcal{D}^{\mathcal{X}} \rightarrow \mathcal{D}$ *is the* evaluation function *(notation: $|b|_\rho$ for $|-|(b, \rho)$)*

and such that the following hold.

1. $(b\langle f \rangle)\langle g \rangle = b\langle f \lhd g \rangle$, *where*

$$(f \lhd g)(x) = \begin{cases} g(x) & \text{if } x \in dom(g) - dom(f) \\ f(x)\langle g \rangle & \text{otherwise} \end{cases}$$

2. $|e\langle f \rangle|_\rho = |e|_{\rho[|f|_\rho]}$, *where $|f|_\rho$ is defined by:* $(|f|_\rho)(x) = |f(x)|_\rho$.

In $\langle \mathcal{B}\text{Exp}, \mathcal{D}\text{Exp}, fv, \langle - \rangle, \models, |-| \rangle$, $\mathcal{B}$Exp is a set of atomic predicates about data values; $\mathcal{D}$Exp 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 $\{x_1, \ldots, x_n\} \subseteq \mathcal{X}$ we use the term *assignment* for the function $(x_1 := e_1, \ldots x_n := e_n)$ in $\mathcal{D}\text{Exp}^{\mathcal{X}}$. We often use $\overline{x} := \overline{e}$ to represent an assignment and call elements of $\mathcal{D}^{\mathcal{X}}$ *data states.*

State-transformation formulas A state transformation specifies how current values of variables will be related to new values after the transformation. To formalize state transformations, let $\mathcal{X}' = \{x' \mid x \in \mathcal{X}\}$ represent the set of "primed" versions of data variables. Then a data predicate $A \in \mathcal{B}$Exp over the variable set $\mathcal{X} \cup \mathcal{X}'$ may be seen as the specification of a state transformation. We refer to formulas such as A as state-transition formulas and use $\mathbb{A}$ to represent the set of all such formulas.

Semantically, state-transformation formulas are interpreted with respect to pairs $\langle \rho, \rho' \rangle$ of data states, where ρ represents the "current" state and ρ' the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

"next" state. We write $\langle \rho, \rho' \rangle \models A$ when A is made true by taking the values for the variables of $\mathcal{X}$ from ρ and the variables of $\mathcal{X}'$ from ρ'.

Note that the assignment function $\overline{x} := \overline{e}$ actually defines a state-transformation formula. We sometime also write the assignment as $\overline{x'} = \overline{e}$.

3.2 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 $B = \langle \mathcal{B}\text{Exp}, \mathcal{D}\text{Exp}, fv, \models, |-| \rangle$ over $\mathcal{X}$ and $\mathcal{D}$. The formulas are given as follows, where $b \in \mathcal{B}\text{Exp}$, $X \in \mathbb{X}$, $x \in \mathcal{X}$, and A is a state transformation formula.

$$\phi \ ::= \ b \mid \neg b \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid X \mid \phi[A] \mid \exists x.\phi \mid \forall x.\phi \qquad (3)$$

The operators are standard, except for X and $\phi[A]$. As formulates may contain predicate variables, operator $\phi[A]$, which is usually a meta-operation, may be thought of as a generalization of the substitution operation. To define it precisely, if ρ is a data state then define $\textsf{post}(\rho, A) = \{\rho' \mid \langle \rho, \rho' \rangle \models A\}$ to be the "post-states" of ρ after A. Then $\rho \models \phi[A]$ holds exactly when every post-state $\rho' \in \textsf{post}(\rho, A)$ satisfies ϕ ($\rho' \models \phi$).

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* $\theta \in (2^{(\mathcal{D}^{\mathcal{X}})})^{\mathbb{X}}$ mapping predicate variables to sets of data states. We write

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$\rho \models_\theta \phi$ to denote that formula ϕ holds in data state ρ and predicate state θ. The definition is as follows.

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

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

3.3 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 3.3.1 *A* predicate equation block *has form* $\langle p, \overline{E} \rangle$*, where* $p \in \{\mu, \nu\}$ *is the* parity indicator *and* $\overline{E} = \langle E_1, \ldots, E_n \rangle$ *is a finite sequence of equations of form* $X_i = \phi_i$*, with the* X_i *distinct predicate variables and each* ϕ *a predicate.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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 $\mathsf{lhs}(B) = \{X_1, \ldots, X_n\}$ for the left-hand-side variables in block B and $\mathsf{rhs}(B) = \{\phi_1, \ldots, \phi_n\}$ for the right-hand-side predicates.

Definition 3.3.2 *A* predicate equation system *(PES) is a finite sequence $\langle B_1, \ldots, B_n \rangle$ of predicate equation blocks with the property that if $i \neq j$, then $\mathit{lhs}(B_i) \cap \mathit{lhs}(B_j) = \emptyset$.*

The notions of lhs and rhs can be extended in the obvious manner to PESs. We call a PES P *predicate-closed* if $\bigcup_{\phi \in \mathsf{rhs}(P)} \mathit{fpv}(\phi) \subseteq \mathsf{lhs}(P)$.

Example 3.3.3 *As an example, we consider the following PES, where the set of data variables $\mathcal{X} \stackrel{def}{=} \{x, y\}$*

$$\begin{cases} X_{A,1} \stackrel{\mu}{=} & X_{A,2} \vee ((x < 8 \rightarrow X_{A,2}[\ \exists \delta.\ \delta > 0 \wedge x' = x + \delta \wedge y' = y + \delta \\ & \wedge x' < 8 \wedge y' < 8\ ]) \wedge (y > 5 \rightarrow X_{B,1}[x' = 0])) \\ X_{A,2} \stackrel{\mu}{=} & x > 7 \\ X_{B,1} \stackrel{\mu}{=} & X_{B,2} \\ X_{B,2} \stackrel{\mu}{=} & x > 7 \end{cases}$$

PESs are interpreted using fixpoints of monotonic functions defined over the complete lattice given by $2^{(\mathcal{D}^{\mathcal{X}})}$ (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:

$$f_\theta(S) = [\![\phi]\!]_{\theta[X:=S]}.$$

A complete account of fixpoint equation systems is given in Chapter 2, and the semantics of PESs may be seen as an instance of this, where the lattice Q is taken to be $2^{(\mathcal{D}^{\mathcal{X}})}$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Given a "starting" environment θ, the semantics, $[\![P]\!]_\theta$, 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 $[\![P]\!]_\theta(X) = [\![P]\!]_{\theta'}(X)$ for any $X \in \mathsf{lhs}(P)$ and θ, θ'. Based on this observation, it follows that if ϕ is a predicate, P is predicate-closed, and $fpv(\phi) \subseteq \mathsf{lhs}(P)$, then

$$[\![\phi]\!]_{[\![P]\!]_\theta} = [\![\phi]\!]_{[\![P]\!]_{\theta'}}$$

for any θ, θ'. In this case we write $[\![\phi]\!]_P$ for this common value, and if $\sigma \in [\![\phi]\!]_P$ we represent this notationally as $\sigma \models_P \phi$.

3.4 Global Approaches to Tautology Checking

Generally speaking, model-checking problems can be solved by proving the tautologiness of a logical formula which contains predicate variables from PESs (We will show this later). The global approach usually involves computing solutions for all predicate variables of the PES.

The iterative strategy for computing the solution to fixpoint equation systems, of course, implies a general approach for model checking. The strategy is based on the following technique for computing solutions to basic blocks.

1. Assign each lhs variable the correct extremal value ($\top$ for ν, $\bot$ 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 may be realized symbolically in the obvious manner: for a ν-block, start by assigning each lhs predicate variable $\mathbf{tt}$, then iterate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

by replacing each occurrence of a lhs variable in a right-hand side and comparing the new expressions with the previous ones. Terminate when there is no change. Note that in general, this strategy might not terminate. First, the basic data theory may not decidable, so mechanically testing formula equivalence cannot be done. Second, the number of iterations needed may not be finite.

Traditional global finite-state model checkers use this strategy, as do both [32] and [64]. In [32], the authors note that, even though Presburger arithmetic is decidable, their procedure is not guaranteed to terminate, owing to the second condition above. In contrast, the restrictions in state predicates in [64] do guarantee termination.

The paper [45] restricts the allowed form of predicates mentioned in [32] so that the only basic comparisons allowed mirror those of [64], 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 approach to discrete-time model checking. Rather than expand a discrete-time model into a concrete transition system (which is detailed in Section 4.1) 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 [45] to conduct model checking symbolically.

3.5 A Gentzen-Like Proof System

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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

received little attention, although in the case of real-time model checking the subject is discussed in [96]. 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 [22, 69].

The proof rules operates on *sequents* of the form: $\Phi \vdash \psi$, where $\Phi = \{\phi_1, \ldots \phi_n\}$ 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 [98], 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 definition of strongest postcondition **post** can be lifted to a set of states defined by the constraint Φ as follows.

$$\mathsf{post}(\Phi, A) = \{\rho' \mid \langle \rho, \rho' \rangle \models A \text{ and } \rho \models \Phi\}$$

and the weakest precondition is defined as the following,

$$\mathsf{pre}(\Phi, A) = \{\rho \mid \langle \rho, \rho' \rangle \models A \;\Rightarrow\; \rho' \models \Phi\}$$

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

Definition 3.5.1 *Let σ be a sequent of form $\Phi \vdash_P \psi$. σ is a* (successful) leaf *if one of the following conditions holds.*

(1) $\psi \in \mathcal{B}Exp$ or $\psi = \neg b$ for some $b \in \mathcal{B}Exp$ (successful if $[\![\bigwedge \Phi \rightarrow \psi]\!] = \mathcal{D}^{\mathcal{X}}$).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$\vee_1$ $\dfrac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi \vdash_P \psi_1}$ $\vee_2$ $\dfrac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi \vdash_P \psi_2}$

$\vee_3$ $\dfrac{\Phi \vdash_P \phi \vee \psi}{\Phi, \mathsf{not}(\phi) \vdash_P \psi}$ $\vee_4$ $\dfrac{\Phi \vdash_P \psi \vee \phi}{\Phi, \mathsf{not}(\phi) \vdash_P \psi}$

$\wedge$ $\dfrac{\Phi \vdash_P \psi_1 \wedge \psi_2}{\Phi \vdash_P \psi_1 \quad ; \quad \Phi \vdash_P \psi_2}$

$[]$ $\dfrac{\Phi \vdash_P \psi[A]}{\mathsf{post}(\Phi, A) \vdash_P \psi}$

S $\dfrac{\Phi, \phi \vdash_P \psi}{\Phi \vdash_P \mathsf{not}(\phi) \vee \psi}$

T $\dfrac{\Phi, \phi \vdash_P \psi}{\Phi \vdash_P \psi}$

$\exists$ $\dfrac{\Phi \vdash_P \exists x.\psi}{\Phi \vdash_P \psi[x' = t]}$ $(t \in \mathfrak{D}\text{Exp})$

$\forall$ $\dfrac{\Phi \vdash_P \forall x.\psi}{\Phi \vdash_P \psi[x' = y]}$ (y a fresh data variable)

CALL $\dfrac{\Phi \vdash_P X}{\Phi \vdash_P \psi}$ ($X = \psi$ is an equation in P)

Figure 2: A Gentzen-like proof system for PESs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(2) $\psi \in \Phi$ *(always successful).*

(3) $\psi = X$ *with parity p, and there is another sequent* σ' *of form* $\Phi' \vdash_P X$ *on the path from the root node of the proof to* σ *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* Φ *logically implies* Φ'*. Such a leaf is successful if the parity of* X *is* ν*.*

The definition of (successful) leaf is based on the one given in [69], which also gives a success criterion for leaves involving μ-formulas. This criterion is not needed in this work, so we omit further mention of it.
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 3.5.2 *(Soundness) The proof rules in Figure 2 are sound: if* $\Phi \vdash_P \psi$ *has a valid proof wrt PES P then* $[\![\Phi \to \psi]\!]_P = \mathcal{D}^{\mathcal{X}}$*.*

Proof: By induction over the derivation of $\Phi \vdash_P \psi$. The inductive step involves proving soundness of each rule in the proof system. And for each rule of the form

$$\frac{\sigma}{\sigma_1, \ldots, \sigma_n}$$

we have to show that if the soundness hold for each subgoal $\sigma_1, \ldots, \sigma_n$, then the goal σ is also sound. During the proof, we will also point out the completeness if applicable.

The soundness of most of the rules is straightforward;

- Rule $\vee_4$ is sound and complete since $(\mathsf{not}(\Phi \wedge \mathsf{not}(\phi)) \vee \psi) \equiv (\mathsf{not}(\Phi) \vee (\phi \vee \psi))$, so are rule $\vee_1, \vee_2, \vee_3$.

- Rule **S** is sound and complete since $(\mathsf{not}(\Phi) \vee \mathsf{not}(\phi) \vee \psi) \equiv (\mathsf{not}(\Phi \wedge \phi) \vee \psi)$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Rule $\mathbf{T}$ strengthens the proof obligation by weakening the left-hand side. It is sound since $\mathsf{not}(\Phi) \vee \psi$ implies $\mathsf{not}(\Phi) \vee \mathsf{not}(\phi) \vee \psi$; note that this rule is not complete.

- Rule [] is sound and complete; Note that the operator $\psi[A]$ actually defines the weakest precondition $\mathsf{pre}(\psi, A)$; The proof follows the definition of predicate transformers, and can be found in [94].

- Rule $\exists$ is the standard skolemization technique to eliminate existential first-order quantifiers. Skolemization does preserve the satisfiability of formulas. This rule is sound and complete.

- Rule $\forall$ eliminates the universal quantifiers by introducing a fresh free variable. It is sound and complete.

- The termination condition in definition 3.5.1(3) is similar to the case of boolean equation systems [39, 78]. The requirement that there is no such a X'' with alternative parity is used to guarantee the monotonicity of the underlying semantic function. $\Phi \Rightarrow \Phi'$ indicates that we have reached a loop in the proof. Such a loop for predicate variable with greatest fixpoint can be identified as a successful leaf according to the fixpoint definition. While a loop for least fixpoint means that the proof need to continue until condition 3.5.1(1) or (2) is reached. The proof for the least fixpoint variable needs the well-founded induction. Interested readers are referred to a similar proof detailed in [69]. Note that the state space consists of data states in their model checking problems.

■

In general, the proof rules are not complete; proofs may require the application of $\mathbf{T}$ rule, and the data theory may not be expressive enough to define

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

Transition Systems and the Modal Mu-Calculus

In this chapter, we show how model checking can be reduced 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 chapter lays the foundation for this approach by introducing our system model, *symbolic transition graphs*, and our temporal logic, the first-order mu-calculus.

4.1 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* $\mathcal{A}\mathrm{ct}_c$ is given as

$$\mathcal{A}\mathrm{ct}_c = \{\lambda!d \mid \lambda \in \Lambda, d \in \mathcal{D}\} \cup \{\lambda?d \mid \lambda \in \Lambda, d \in \mathcal{D}\} \cup \{d \mid d \in \mathcal{D}\} \cup \{\tau\}.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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 4.1.1 *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,* $\rightarrow_c \subseteq \Sigma \times \mathcal{A}ct_c \times \Sigma$ *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 $\langle \sigma, a, \sigma' \rangle \in \rightarrow_c$.

4.2 The First-Order Modal Mu-Calculus

To specify system properties, we use first-order modal mu-calculus [99] 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 $\langle \mathcal{B}\text{Exp}, \mathcal{D}\text{Exp}, fv, \langle - \rangle, \models, |-| \rangle$ and set Λ of port names. Then first-order mu-calculus formulas have the following form, where $e \in \mathcal{D}\text{Exp}$ and $\lambda \in \Lambda$.

$$
\begin{aligned}
\phi \ ::= \ & \langle \text{operators from Equation } 3 \rangle \mid \\
& \langle \tau \rangle \phi \mid [\tau]\phi \mid \langle \lambda!e \rangle \phi \mid [\lambda!e]\phi \mid \langle \lambda?e \rangle \phi \mid [\lambda?e]\phi
\end{aligned}
$$

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 $fpv(\phi) = \emptyset$.

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 $\sigma \models_{C,\theta} \phi$, which, given an environment $\theta \in (2^{\Sigma})^{\mathbf{X}}$ mapping formula variables to sets of CTS states, determines whether or not a CTS state satisfies ϕ. This relation is given as follows.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$
\begin{array}{llll}
\sigma & \models_{C,\theta} & b & \text{iff } V(\sigma) \models b \\
\sigma & \models_{C,\theta} & \neg b & \text{iff } V(\sigma) \not\models b \\
\sigma & \models_{C,\theta} & \phi_1 \vee \phi_2 & \text{iff } \sigma \models_{C,\theta} \phi_1 \text{ or } \sigma \models_{C,\theta} \phi_2 \\
\sigma & \models_{C,\theta} & \phi_1 \wedge \phi_2 & \text{iff } \sigma \models_{C,\theta} \phi_1 \text{ and } \sigma \models_{C,\theta} \phi_2 \\
\sigma & \models_{C,\theta} & X & \text{iff } \sigma \in \theta(X) \\
\sigma & \models_{C,\theta} & \phi[A] & \text{iff } \sigma \models_{C,\theta} \mathsf{pre}(\phi, A) \\
\sigma & \models_{C,\theta} & \exists x.\phi & \text{iff there is some } d \in \mathcal{D},\ \sigma \models_{C,\theta} \phi[x' = d] \\
\sigma & \models_{C,\theta} & \forall x.\phi & \text{iff for all } d \in \mathcal{D},\ \sigma \models_{C,\theta} \phi[x' = d] \\
\sigma & \models_{C,\theta} & \langle\tau\rangle\phi & \text{iff there is } \sigma' \text{ s.t. } \sigma \xrightarrow{\tau}_c \sigma' \text{ and } \sigma' \models_{C,\theta} \phi \\
\sigma & \models_{C,\theta} & [\tau]\phi & \text{iff for all } \sigma' \text{ s.t. } \sigma \xrightarrow{\tau}_c \sigma',\ \sigma' \models_{C,\theta} \phi \\
\sigma & \models_{C,\theta} & \langle\lambda!e\rangle\phi & \text{iff there is } \sigma' \text{ s.t. } \sigma \xrightarrow{\lambda!d}_c \sigma',\ |e|_{V(\sigma)} = d, \text{ and } \sigma' \models_{C,\theta} \phi \\
\sigma & \models_{C,\theta} & \langle\lambda?e\rangle\phi & \text{iff there is } \sigma' \text{ s.t. } \sigma \xrightarrow{\lambda?d}_c \sigma',\ |e|_{V(\sigma)} = d, \text{ and } \sigma' \models_{C,\theta} \phi \\
\sigma & \models_{C,\theta} & [\lambda!e]\phi & \text{iff for all } \sigma' \text{ s.t. } \sigma \xrightarrow{\lambda!d}_c \sigma',\ |e|_{V(\sigma)} = d, \text{ and } \sigma' \models_{C,\theta} \phi \\
\sigma & \models_{C,\theta} & [\lambda?e]\phi & \text{iff for all } \sigma' \text{ s.t. } \sigma \xrightarrow{\lambda?d}_c \sigma',\ |e|_{V(\sigma)} = d, \text{ and } \sigma' \models_{C,\theta} \phi
\end{array}
$$

Note that the semantics of the modal operators are different from the ones given in [75, 91]. 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 $\exists$. Also note that $\mathsf{pre}(\phi, A)$ defines the weakest precondition of ϕ with respect to the state-transformation formula A.

We define $[\![\phi]\!]_{C,\theta} = \{\sigma \mid \sigma \models_{C,\theta} \phi\}$. We may now apply the general fixpoint equation system theory to define the semantics of mu-calculus equation systems. The lattice in question is 2^{Σ} ordered by set inclusion, the semantics, $[\![M]\!]_{C,\theta}$, of mu-calculus equation system M is an environment mapping each $X \in \mathsf{lhs}(M)$ to a set of states that is the appropriate solution for the equation defining X.

We also adapt the definitions of predicate-closed-ness from PESs to formula-closed-ness in the obvious manner. If MES M is formula-closed then

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$[\![M]\!]_{C,\theta}(X) = [\![M]\!]_{C,\theta'}(X)$ for any $X \in \mathsf{lhs}(M)$ and θ, θ', and we write $[\![M]\!]_C$ for this value. It also follows that if M is formula-closed and ϕ is such that $fpv(\phi) \subseteq \mathsf{lhs}(M)$, then

$$[\![\phi]\!]_{[M]_{C,\theta}} = [\![\phi]\!]_{[M]_{C,\theta'}}$$

for any θ, θ'. When this holds we use $[\![\phi]\!]_{C,M}$ for this value, and we write $\sigma \models_{C,M} \phi$ if $\sigma \in [\![\phi]\!]_{C,M}$.

From CTL to MES The first-order modal mu-calculus we introduced are expressive enough to encode the CTL-style temporal formulas [85]. To provide the translation, we can take use of the standard fixpoint characteristics of temporal operators. Assume the CTL formulas are in positive normal form, i.e. all negations have been "pushed" inside formulas until they reach atomic formulas. Then it is sufficient to give accounts of the following formulas [43].

$$\begin{aligned}
A(\phi_1 W \phi_2) &= \nu X.(\phi_2 \vee (\phi_1 \wedge [\tau]X)) \\
E(\phi_1 W \phi_2) &= \nu X.(\phi_2 \vee (\phi_1 \wedge \langle\tau\rangle X)) \\
A(\phi_1 U \phi_2) &= \mu X.(\phi_2 \vee (\phi_1 \wedge [\tau]X \wedge \langle\tau\rangle \mathbf{tt})) \\
E(\phi_1 U \phi_2) &= \mu X.(\phi_2 \vee (\phi_1 \wedge \langle\tau\rangle X))
\end{aligned}$$

In the above, A and E are the universal path quantifier and the existential path quantifier respectively; W is the "weak" path operator, and U is the "strong" path operator. A state s satisfy $A(\phi_1 U \phi_2)$ if along every computation path beginning with s, ϕ_1 holds until ϕ_2 does; A state s satisfy $A(\phi_1 W \phi_2)$ if either it satisfy $A(\phi_1 U \phi_2)$, or when ϕ_2 does not necessarily hold, ϕ_1 holds everywhere.

Note that the translation is linear-time and does contain only internal communication actions since CTL does not distinguish communication events.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Example 4.2.1 *As an example, we consider the first-order CTL formula* $AF(x > 7)$, *the corresponding MES is the following.*

$$\begin{cases} X_1 \stackrel{\mu}{=} & X_2 \vee [\tau]X_1 \\ X_2 \stackrel{\mu}{=} & x > 7 \end{cases}$$

4.3 Symbolic Transition Graphs

Our symbolic system model, Symbolic Transition Graphs(STGs), extends the STGA formalism of [75] with state transformation formulas. This extension enables STGs to encode a range of other symbolic system formats, including the value-passing CCS in [42], Linear Process Equations [59], the event-action language in [32], and timed automata [64].

Fix value set $\mathcal{D}$, variable set $\mathcal{X}$, and data theory $\langle \mathcal{B}\text{Exp}, \mathcal{D}\text{Exp}, fv, \langle - \rangle, \models, |-| \rangle$ 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

$$\mathcal{A}\text{ct}_s = \{\lambda?x \mid c \in \Lambda, x \in \mathcal{X}\} \cup \{\lambda!e \mid c \in \Lambda, e \in \mathcal{D}\text{Exp}\} \cup \{\tau\}.$$

Then STGs are defined as follows.

Definition 4.3.1 *An STG is a tuple* $\mathcal{G} = \langle S, R, S_I, Init\mathcal{C} \rangle$, *where:*

1. S *is a finite set of control locations;*
2. $R \subseteq S \times \Phi \times \mathbb{A} \times \mathcal{A}ct_s \times S$ *is a finite set of transitions.*
3. $S_I \subseteq S$ *are the initial locations; and*
4. $Init\mathcal{C} \in \mathcal{B}Exp$ *is the initial condition.*

In STG $\mathcal{G} = \langle S, R, S_I, \text{Init}\mathcal{C} \rangle$, S_I contains the possible starting locations and Init$\mathcal{C}$ the initial condition on data variables. Based on the current control

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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 transitions in an STG. Each transition is a tuple $\langle s, \beta, A, \alpha, s' \rangle$, where s and s' are the source and target control location, respectively. β determines when the transition can "fire"; a state transformation formula A update data variables; and a communication action α.

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

1. $\Sigma \stackrel{\text{def}}{=} S \times \mathcal{D}^{\mathcal{X}}$. 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 $\langle s, \beta, A, \alpha, s' \rangle \in R$, and ρ'' with:

   (a) $\rho \models \beta$, $\rho'' \in \mathsf{post}(\rho, A)$, and

   (b) either:

      i. $a = \tau$ and $\rho' = \rho''$; or

      ii. $a = \lambda!d$, $\alpha = \lambda!e$, $|e|_\rho = d$, and $\rho' = \rho''$; or

      iii. $a = \lambda?d$, $\alpha = \lambda?x$, and $\rho' = \rho''[x' = d]$.

4. $\sigma_I = \{\langle s_I, \rho \rangle \mid s_I \in S_I, \rho \models \mathrm{Init}\mathcal{C}\}$

Example 4.3.2 *As an example, considering the simple STG in Figure 3, where the set of data variables* $X \stackrel{\text{def}}{=} \{x, y\}$.

STGs and the Mu-Calculus. The definition of C_G implies an immediate interpretation of the mu-calculus with respect to STG G. In addition to the other notations defined for the mu-calculus, we also introduce the following.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

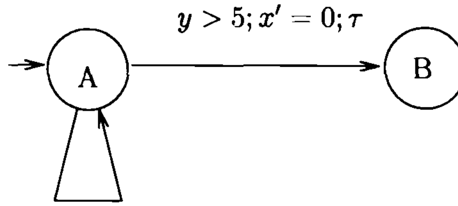


Figure 3: A simple symbolic transition graph

Let ϕ be a mu-calculus formula, and s a control location in STG G, and let θ be a mapping of mu-calculus formula variables to sets of states in C_G. Then

$$[\![\phi]\!]_\theta(s) = \{\rho \mid \langle s, \rho\rangle \in [\![\phi]\!]_{C_G,\theta}\}.$$

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 $fpv(\phi) \subseteq \mathsf{lhs}(M)$, we write $[\![\phi]\!]_{G,M}(s)$ for $\{\rho \mid \langle s, \rho\rangle \in [\![\phi]\!]_{C_G,M}\}$. In this case, we also say that a STG G satisfies a mu-calculus formula ϕ with respect to equation system M (written $G \models_M \phi$) if for all $s_I \in S_I$, $\{\rho \mid \rho \models \text{Init}\mathcal{C}\} \subseteq [\![\phi]\!]_{G,M}(s_I)$.

4.4 From Model Checking to PESs

The model-checking problem for STGs is: given STG G, formula-closed MES M and $X \in \mathsf{lhs}(M)$, 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 $[\![X]\!]_{G,M}(s_I)$ 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 STG 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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is,

$$F(G, \langle B_1, \ldots, B_n\rangle) = \langle F(G, B_1), \ldots, F(G, B_n)\rangle.$$

While $F(G, B) = F(G, \langle p, \overline{E}\rangle)$ 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 4.

Example 4.4.1 *An as example, the PES given in 3.3.3 is generated from the STG 4.3.2 and the MES 4.2.1.*

Lemma 4.4.2 *Let* $G = \langle S, R, S_I, Init\mathfrak{C}\rangle$ *be an STG with the interpretation* $C_G = \langle \Sigma, V, \rightarrow_c, \Sigma_I\rangle$, *and* M *a closed MES; Let* θ *be a mapping of mu-calculus formula variables to sets of states in* C_G, θ' *be a mapping of predicate variables to sets of data states; If for any* $X \in \mathsf{lhs}(M)$ *and any* $s \in S$, $\theta(X)(s) = \theta'(Y_{s,X})$, *then for any* ψ, *we have* $[\![\psi]\!]_\theta(s) = [\![F(s, \psi)]\!]_{\theta'}$

Proof: The proof proceeds by structural induction on the formula ψ.

For the base case, $[\![X]\!]_\theta = [\![Y_{s,X}]\!]_{\theta'} = [\![F(s, \psi)]\!]_{\theta'}$.

Most cases are routine; We consider here the case when ψ is $\langle\tau\rangle\psi$.

A data state $\rho \in [\![\langle\tau\rangle\psi]\!]_\theta(s)$ if only if there is a transition $\langle s, \beta, A, \tau, s'\rangle \in R$, $\rho \models \beta$ and $\mathsf{post}(\rho, A) \in [\![\psi]\!]_\theta(s')$, since $[\![\psi]\!]_\theta(s') = [\![F(s', \psi)]\!]_{\theta'}$ (the inductive assumption). we have

$$\rho \models_{\theta'} \bigvee\{\beta \wedge F(s', \psi)[A] \mid \langle s, \beta, A, \tau, s'\rangle \in R\}.$$

It follows that

$$\begin{aligned}[\![\langle\tau\rangle\psi]\!]_\theta(s) &= [\![\bigvee\{\beta \wedge F(s', \psi)[A] \mid \langle s, \beta, A, \tau, s'\rangle \in R\}]\!]_{\theta'} \\ &= [\![F(s, \langle\tau\rangle\psi)]\!]_{\theta'}\end{aligned}$$

■

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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

Figure 4: Translation function for PESs

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Theorem 4.4.3 *Let $G = \langle S, R, S_I, Init\mathcal{C}\rangle$ be an STG, and let M be a closed MES. Then for any $s \in S$ and any $X \in lhs(M)$, $[\![X]\!]_{G,M}(s) = [\![Y_{s,X}]\!]_{F(G,M)}$.*

Proof: The proof proceeds by establishing the connection between the semantic function for MES and the generated PES.

Suppose $\mathbb{X} \stackrel{\text{def}}{=} \{X_1, \ldots, X_n\}$ are the formula variables in M, $\mathcal{X}$ is the set of data variables and $\mathcal{D}$ is the data domain. Let $S \stackrel{\text{def}}{=} \{s_1, \ldots, s_m\}$. Then the semantic used to interpret M is given by $f : (S \times \mathcal{D}^{\mathcal{X}})^{\mathbb{X}} \rightarrow (S \times \mathcal{D}^{\mathcal{X}})^{\mathbb{X}}$; Since for each $X \in \mathbb{X}$ and $s \in S$, $F(G, M)$ will generate a predicate variable $Y_{s,X}$ for the PES, the semantic function for the PES is given by $g : (\mathcal{D}^{\mathcal{X}})^{\mathbb{Y}} \rightarrow (\mathcal{D}^{\mathcal{X}})^{\mathbb{Y}}$, where $\mathbb{Y} \stackrel{\text{def}}{=} \{Y_{s,X} \mid s \in S, X \in \mathbb{X}\}$. Let σ be the parity of the block, then the fixpoint σf encodes the solution to each formula variable in the MES and σg contains the solution to each predicate variable in the PES.

For each semantic set $X_i : S \times \mathcal{D}^{\mathcal{X}}$, we construct a set $Y_{s,X_i} : \{d \mid \langle x, d\rangle \in X_i\}$. Then the input $(X_1, \ldots, X_n)$ to function f becomes the input $(Y_{s_1,X_1}, \ldots, Y_{s_1,X_n}, \ldots, Y_{s_m,X_1}, \ldots, Y_{s_m,X_n})$ to function g. Given one solution of f, we can construct the solution for g and the vice versa. We can conclude that the relationship between the solutions of the two semantic functions as illustrated by Figure 5.

$$\begin{array}{ccc}
\sigma f(X_1, \ldots, X_n) & = & \langle \sigma X_1, \ldots, \sigma X_n \rangle \\
\updownarrow & & \updownarrow \\
\sigma g(Y_{s_1,X_1}, \ldots, Y_{s_m,X_n}) & = & \langle \sigma Y_{s_1,X_1}, \ldots, \sigma Y_{s_m,X_n} \rangle
\end{array}$$

Figure 5: The relationship between the two semantic functions

■

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.5 Finite-State Model Checking with PESs

As mentioned in the introduction, BESs play a central role in model checking of finite-state systems. These equation systems resemble PESs; the key difference is that BESs contain propositional, rather than predicate variables. Each such variable represents whether or not a given state in a system satisfies a given temporal formula.

For finite-state systems, our PES-generation procedure returns BESs. To illustrate this, we consider system descriptions given as CTSs as described in Section 4.1, where there is exactly one data value, d, and formulas are given in the equational propositional mu-calculus (MESs without quantification whose only data expression is the constant d).

Formally, define the sets $\mathcal{D} = \{d\}$ and $\mathcal{X} = \{x\}$; the basic data theory we use takes $\mathcal{B}\text{Exp} = \{\mathbf{tt}\}$ and $\mathcal{D}\text{Exp} = \{d\}$; the function fv returns $\emptyset$ for any argument, $\langle - \rangle$ is trivial, and $\sigma \models \mathbf{tt}$ holds always and $|d|_\sigma = d$ for any σ. Note that the set $\mathcal{D}^{\mathcal{X}} = \{\sigma\}$ contains exactly one element.

Given these definitions, a CTS C may be encoded as a STG P_C. The control locations of P_C are exactly the states of C, and data variable x is needed because input transitions need an assignable variable. For each transition $s \xrightarrow{a} s'$ in C, we generate a transition of the form $\langle s, \mathbf{tt}, x' = x, a, s' \rangle$, if $a \in \{\tau, \lambda!d\}$, and $\langle s, \mathbf{tt}, x' = d, \lambda?x, s' \rangle$, if $a = \lambda?d$. The initial control locations are the initial states of C, and $\text{Init}\mathcal{C} = \mathbf{tt}$. It is easy to see that the CTS where the semantics of STGs associates to P_C is isomorphic to C.

Since the equational propositional mu-calculus is a sublanguage of MESs, our PES generation procedure may be applied to P_C and an equational system M. All quantifiers can easily be eliminated from the resulting PES. Moreover, since $\mathcal{D}^{\mathcal{X}}$ contains only σ, the semantic lattice for predicates consists of two elements: $\{\sigma\}$ ("true"), and $\emptyset$ ("false"). Each predicate variable may thus be seen as a propositional variable, and the PES is isomorphic to a BES.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Using a similar encoding, Mateescu's parameterized boolean equation systems [81] can be regarded as instances of our PESs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

Real-Time Model Checking with PESs

This chapter shows how PESs can be used to encode (parametric) real-time model-checking problems. In this setting, systems are modeled as (parametric) timed automata and properties are specified with real-time modal mu-calculus. An efficient on-the-fly (parametric) model-checking algorithm is developed by providing specialized proof rules.

5.1 Parametric Timed Automata

Real-time systems are often modeled as timed automata [2], For the convenience of statement, we use parametric timed automata to model both non-parametric and parametric real-time systems. We began by introducing some terminology and notation.

Throughout let C be a finite set of clock variables ranging over $x, y, \ldots$, $\mathcal{A}ct$ a finite set of actions (transition labels), $\mathcal{P}$ a finite set of parameter variables ranging over $a, b, \ldots$, and α, β 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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

constants and each $a_i \in \mathcal{P}$. The set of *state predicates* is defined by the grammar.

$$\varphi_s := \alpha \sim \beta \mid x \sim \alpha \mid x - y \sim \alpha \tag{4}$$

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 Φ. Throughout we let $\mathcal{D} \stackrel{\text{def}}{=} \mathbb{R}^+ \cup \{0\}$ be the set of possible *durations* and $\mathcal{X} \stackrel{\text{def}}{=} C \cup \mathcal{P}$.

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 ω, a *system state* $\rho \in \mathcal{D}^{\mathcal{X}}$ satisfies: $\rho(a) = \omega(a)$ if $a \in \mathcal{P}$. If ρ is a system state and $\delta \in \mathcal{D}$ then $\rho + \delta$ is the new state $\rho[x_1 := \rho(x_1) + \delta, \ldots, x_n := \rho(x_n) + \delta]$, which updates each clock variable x_i with a new value $\rho(x_i) + \delta$ and agrees with ρ 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 5.1.1 *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. *$R \subseteq S \times \Phi \times 2^C \times Act \times S$ 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 Φ;*
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 $\langle s, \varphi, C, \alpha, s' \rangle$ may be executed

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

when the trigger condition φ 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 $\mathsf{T} = \langle S, R, L, S_I \rangle$ can be interpreted as a *concrete transition system.*

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

1. $\Sigma = \{\langle s, \rho \rangle \in S \times \mathcal{D}^{\mathcal{X}} \mid \text{for each } a \in \mathcal{P}, \rho(a) = \omega(a)\}$.

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: $\langle s, \rho \rangle \xrightarrow{\alpha}_c \langle s', \rho' \rangle$ iff there is $\langle s, \varphi, C, \alpha, s' \rangle \in R$ with: $\rho \models L(s)$ and $\rho \models \varphi$ and $\rho' = \rho[C := 0]$

4. $\sigma_I = \{\langle s_I, \rho \rangle \mid s_I \in S_I, \rho \models L(s_I) \text{ and } \rho(x) = 0 \text{ for each } x \in C\}$

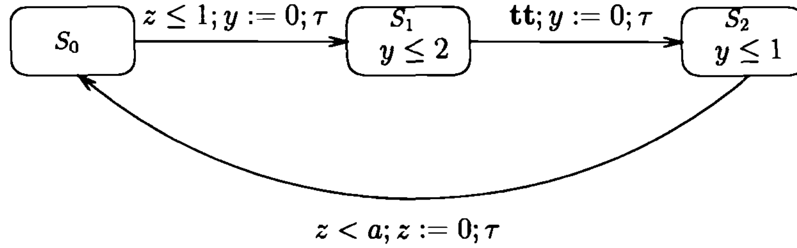


Figure 6: A parametric timed automaton with two clocks

Example 5.1.2 *Consider the parametric timed automaton of Figure 6 with two clocks. The clock y gets set to 0 each time the system switches from location S_0 and S_1. The invariant $y \leq 2$ and $y \leq 1$ ensures that the switch from S_1 happens within time 2 and from S_2 within time 1. Parameter a specifies the upper time limit over clock z when the transition from S_2 happens.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2 The Real-Time Modal Mu-Calculus

The real-time modal mu-calculus [96] can be used to specify system properties. We define formulas with MESs , which consist of blocks of equations of the form $X = \phi$, where $X \in \mathbb{X}$ is a formula variable and ϕ is a formula defined by the following grammar.

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

In the above, $\alpha \in \mathcal{A}ct$ is an action while $x \in C$ is clock. Operators $\langle \alpha \rangle \phi$ and $[\alpha]\phi$ are called modal operators; these, together with $\vee$ and $\wedge$, are standard from the propositional modal mu-calculus [72]. φ_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.

The semantics of the real-time 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 σ satisfies ϕ. This relation is given as follows.

$\sigma \models_{C,\theta} \varphi_s$	iff $V(\sigma) \models \varphi_s$
$\sigma \models_{C,\theta} X$	iff $\sigma \in \theta(X)$
$\sigma \models_{C,\theta} \phi_1 \vee \phi_2$	iff $\sigma \models_{C,\theta} \phi_1$ or $\sigma \models_{C,\theta} \phi_2$
$\sigma \models_{C,\theta} \phi_1 \wedge \phi_2$	iff $\sigma \models_{C,\theta} \phi_1$ and $\sigma \models_{C,\theta} \phi_2$
$\sigma \models_{C,\theta} \exists \phi$	iff $\exists \delta$ s.t. $\sigma \xrightarrow{\delta}_c \sigma' \wedge \sigma' \models_{C,\theta} \phi$
$\sigma \models_{C,\theta} \forall \phi$	iff $\forall \delta$ s.t. $\sigma \xrightarrow{\delta}_c \sigma' \implies \sigma' \models_{C,\theta} \phi$
$\sigma \models_{C,\theta} \langle \alpha \rangle \phi$	iff $\exists \sigma'$ s.t. $\sigma \xrightarrow{\alpha}_c \sigma' \wedge \sigma' \models_{C,\theta} \phi$
$\sigma \models_{C,\theta} [\alpha]\phi$	iff $\forall \sigma'$ s.t. $\sigma \xrightarrow{\alpha}_c \sigma'$, $\sigma' \models_{C,\theta} \phi$
$\sigma \models_{C,\theta} x.\phi$	iff $\sigma \models_{C,\theta} \phi[x := 0]$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We define $[\![\phi]\!]_{C,\theta} = \{\sigma \mid \sigma \models_{C,\theta} \phi\}$. We may now apply the general fixpoint-equation system theory to define the semantics of MESs.

Example 5.2.1 *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" [96].*

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

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

The real-time modal mu-calculus is expressive enough to encode many timed temporal logics, including TCTL [64] . On the other hand, it can be easily encoded by the first-order modal mu-calculus defined in Section 4.2. Note that operator $x.\phi$ can be encoded as $\phi[x' = 0]$ and $\forall\phi$ can be rewritten as $\forall\delta > 0.\phi[\bar{x} := \bar{x} + \delta]$.

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 ϕ be a mu-calculus formula, and s a control location in PTA T, and let θ be a mapping of mu-calculus formula variables to sets of states in $C_{\omega,T}$. Then

$$[\![\phi]\!]_\theta(s) = \{\rho \mid \langle s, \rho\rangle \in [\![\phi]\!]_{C_{\omega,T},\theta} \text{ for all } \omega\}.$$

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 ω. Similarly, if M is a formula-closed MES, and ϕ is a mu-calculus formula with $\mathit{fpv}(\phi) \subseteq \mathsf{lhs}(M)$, we write

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$[\![\phi]\!]_{T,M}(s)$ for $\{\rho \mid \langle s, \rho\rangle \in [\![\phi]\!]_{C_\omega,T,M}$ for all $\omega\}$. In this case, we also say that PTA T satisfies a mu-calculus formula ϕ with respect to equation system M under initial (state-predicate-specified) condition φ (written $T \models^{\varphi}_{M} \phi$) if for all $s_I \in S_I$, $\{\rho \mid \rho \models \varphi\} \subseteq [\![\phi]\!]_{T,M}(s_I)$.

5.3 From Real-Time Model Checking to PESs

The universal parametric model-checking problem may be phrased as follows: given a PTA T, formula-closed MES M and $X \in \mathsf{lhs}(M)$, and a constraint φ over parameter and clock variables, does $T \models^{\varphi}_{M} X$?

This section shows how to translate this question into an equivalent one involving PESs. The translation is an customization of the general PES-generation procedure. The key problem to be addressed is the symbolic representation of the set $[\![X]\!]_{T,M}(s_I)$ for every $s_I \in S_I$ in the parametric real-time settings. Again, this is achieved by constructing a PES equation for each location in T and equation in M. Formally, we define the function F that, given a PTA T and formula-closed mu-calculus equation system M, yields a predicate-closed PES $F(T, M)$. And F is applied on a block-by-block basis; that is, $F(T, \langle B_1, \ldots, B_n\rangle) = \langle F(T, B_1), \ldots, F(T, B_n)\rangle$. $F(T, B) = F(T, \langle p, \overline{E}\rangle)$ 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.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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

Theorem 5.3.1 *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 $X \in lhs(M)$, $[\![X]\!]_{T,M}(s) = [\![Y_{s,X}]\!]_{F(T,M)}$.*

Proof: As an instance of the generic translation to the real-time domain, Proof proceeds in the same way as Theory 4.4.3. ■

It follows that $T \models_M^{\varphi} X$ iff the statement $\mathcal{D}^{\mathcal{X}} = [\![\varphi \rightarrow \bigwedge_{s_I \in S_I} Y_{s_I,X}]\!]_{F(T,M)}$ is true.

Regions Alur *et al.* [3] defined an equivalence relation on the state space of an automaton that equates two clock states if they agree on the integral parts of all clocks values and on the ordering of the fractional parts of all clock values. Let $\delta \in \mathcal{D}$, then $\delta = \lfloor\delta\rfloor + frac(\delta)$, where $\lfloor(\delta)$ is the integral part and $frac(\delta)$ is the fractional part of δ. For each clock $x \in C$, let c_x be the largest integer such that x is compared with in the PES. Given a parameter valuation, the region equivalence relation $\cong$ is defined over the set of all clock states. For two clock state ρ_1 and ρ_2, $\rho_1 \cong \rho_2$ iff all the following conditions hold,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. For all $x \in C$, either $\lfloor \rho_1(x) \rfloor$ and $\lfloor \rho_2(x) \rfloor$ are the same or both $\rho_1(x)$ and $\rho_2(x)$ exceeds c_x;

2. For all x, y with $\rho(x) \leq c_x$ and $\rho(y) \leq c_y$, $frac(\rho_1(x)) \leq frac(\rho_1(y))$ iff $frac(\rho_2(x)) \leq frac(\rho_2(y))$;

3. For all x with $\rho_1(x) \leq c_x$, $frac(\rho_1(x)) = 0$ iff $frac(\rho_2(x)) = 0$.

Given a parameter valuation, a *clock region* is an equivalence class of system states induced by such an equivalence relation. Note that the number of clock regions is limited by an upper bound, $n! \times 2^n \times \prod_{x \in C}(2c_x + 2)$, where n is the number of clocks.

Example 5.3.2 *Figure 7 illustrates the region equivalence for two clocks x and y with $c_x = 3$ and $c_y = 2$. There are 12 corner points, e.g. $(1, 1)$; 30 open line segments, e.g. $1 < x < 2 \wedge y = 1$; 18 open regions, e.g. $1 < x < y < 2$.*

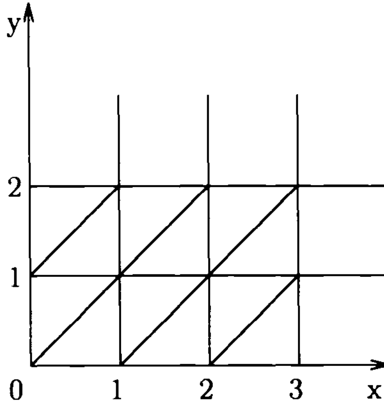


Figure 7: Clock regions

Clock Zones Given a parameter valuation $\omega \in \mathcal{V}^{\mathcal{P}}$, a *clock zone* is a set of clock states described by finite conjunction of state predicates. If the PES has n clocks, then a clock zone is a convex set in the n-dimensional Euclidean

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

space. Clock zones improve the region construction by considering only the convex union of clock regions.

5.4 On-the-Fly Real-Time Model Checking

This section introduces a goal-directed proof system, which is customized from the general proof system, for the solving of universal parametric real-time model-checking problem based on PESs. The proof system is intended to establish when a set of predicate-closed formulas $\Phi \stackrel{\text{def}}{=} \{\phi_1, \ldots, \phi_n\}$ implies a formula ψ 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 $[\![\bigwedge \Phi \rightarrow \psi]\!]_P = \mathcal{D}^{\mathcal{X}}$ (i.e. the implication is a tautology). The rules are given in Fig. 9 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 ψ, ψ_i need not be, and Φ, ϕ is short-hand for $\Phi \cup \{\phi\}$. Also note that s, s' are placeholders, whose meaning will be clear later.

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

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

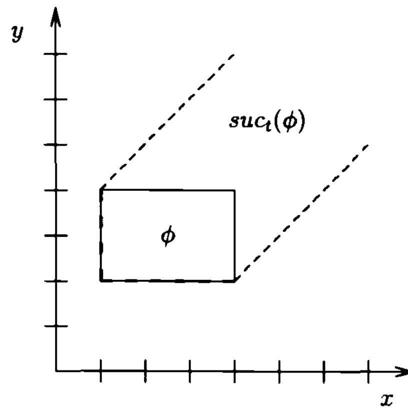
Note that $\phi \vdash_P \psi[\bar{x} := \bar{e}]$ is valid if and only if $\mathsf{post}(\phi, \bar{x} := \bar{e})$ implies ψ. The *weakest precondition* $\mathsf{pre}(\phi, A)$ is defined as

$$\mathsf{pre}(\phi, \bar{x} := \bar{x} + \delta) \stackrel{\text{def}}{=} \forall \bar{v}.(\bar{v} = \bar{x} + \delta) \rightarrow \phi[\bar{x} := \bar{v}]$$

We also have two derived operators,

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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



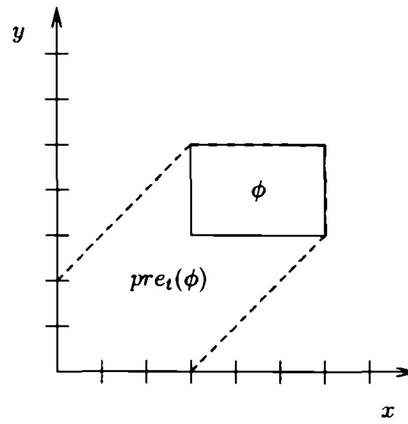


Figure 8: The graphic representation of $pre_t(\phi)$ and $suc_t(\phi)$

Rules $\vee_1$–CALL are familiar from our general Gentzen-like proof system. Instead of a rule T and a cut rule [22] 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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\vee_1 \; \frac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi \vdash_P \psi_1} \qquad \vee_2 \; \frac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi \vdash_P \psi_2}$$

$$\vee_3 \; \frac{\Phi \vdash_P \phi \vee \psi}{\Phi, \mathsf{not}(\phi) \vdash_P \psi} \qquad \vee_4 \; \frac{\Phi \vdash_P \psi \vee \phi}{\Phi, \mathsf{not}(\phi) \vdash_P \psi}$$

$$\wedge \; \frac{\Phi \vdash_P \psi_1 \wedge \psi_2}{\Phi \vdash_P \psi_1 \quad ; \quad \Phi \vdash_P \psi_2} \qquad \text{CALL} \; \frac{\Phi \vdash_P X}{\Phi \vdash_P \psi} \; (X \stackrel{\text{def}}{=} \psi \in P)$$

$$\vee \; \frac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi, s \vdash_P \psi_1 \quad ; \quad \Phi, \neg s \vdash_P \psi_2}$$

$$\forall_1 \; \frac{\Phi \vdash_P \forall \delta . \psi[\bar{x} := \bar{x} + \delta]}{suc_t(\Phi) \vdash_P \psi}$$

$$\forall_2 \; \frac{\Phi, s \vdash_P \forall \delta . \psi[\bar{x} := \bar{x} + \delta]}{suc_t(\Phi), s' \vdash_P \psi \; ; \; suc_t(\Phi \wedge s) \vdash_P suc_t(\Phi) \wedge s'}$$

$$\exists_1 \; \frac{\Phi \vdash_P pre_t(\psi)}{suc_t(\Phi), s \vdash_P \psi \; ; \; \Phi \vdash_P pre_t(s)}$$

$$\exists_2 \; \frac{\Phi, s \vdash_P pre_t(\psi)}{suc_t(\Phi), s' \vdash_P \psi \; ; \; s \vdash_P pre_t(s')}$$

$$[]_1 \; \frac{\Phi \vdash_P \psi[A]}{\mathsf{post}(\Phi, A) \vdash_P \psi}$$

$$[]_2 \; \frac{\Phi, s \vdash_P \psi[A]}{\mathsf{post}(\Phi, A), s' \vdash_P \psi \; ; \; s \vdash_P \mathsf{pre}(s', A)}$$

$$\text{LEAF} \quad \Phi, s \vdash_P \varphi \begin{cases} \text{if } \Phi \rightarrow \varphi \text{ a tautology,} & s \stackrel{\text{def}}{=} \mathsf{true} \\ \text{if } \Phi \rightarrow \varphi \text{ a contradiction,} & s \stackrel{\text{def}}{=} \mathsf{false} \\ \text{otherwise} & s \stackrel{\text{def}}{=} \varphi \end{cases}$$

Figure 9: A local approach for parametric real-time model checking.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(see Rules $\exists_2, \forall_2, []_2$). This strategy is inspired by the *splitting* technique used in [96].

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 subsequent. For example, in Figure 10, s might be $x \leq 4$.

Rule $\exists_1$ eliminates the existential quantifier by introducing a placeholder. The right subsequent is used to tell the validity of the splitting s derived from the left subtree. For example, in Figure 11, s might be the shaded region.

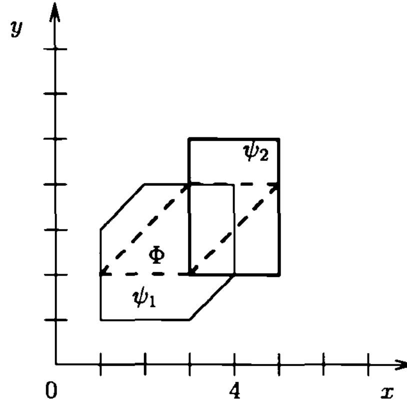


Figure 10: Example for rule $\vee$

Rules $\forall_2$, $\exists_2$ and $[]_2$ all have a right subgoal which is used to compute the weakest splitting constraints s from s'.

The rules also share an implicit side condition: they may only be applied to *non-leaf* sequents. These are defined the same as Section 3.5 for the general Gentzen-like proof system.

Lemma 5.4.1 *We have the following implication hold,*

$$pre_t(s) \wedge \Phi \Rightarrow pre_t(s \wedge suc_t(\Phi))$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

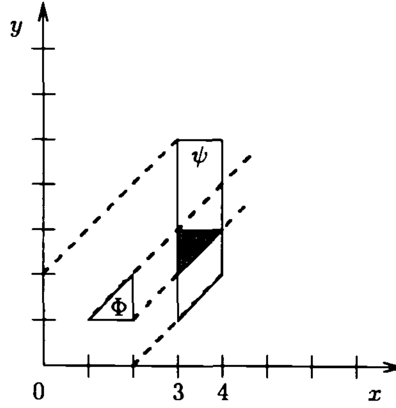


Figure 11: Example for rule $\exists_1$

Proof: The proof proceeds as follows.

$$
\begin{aligned}
pre_t(s) \wedge \Phi =& \exists\delta.\mathsf{pre}(s, x := x + \delta) \wedge \Phi \\
=& (\mathsf{pre}(s, x := x + \delta_1) \vee \cdots \vee \mathsf{pre}(s, x := x + \delta_i) \vee \cdots) \wedge \Phi \\
=& (\mathsf{pre}(s, x := x + \delta_1) \wedge \Phi) \vee \cdots \vee (\mathsf{pre}(s, x := x + \delta_i) \wedge \Phi) \vee \cdots \\
=& (\mathsf{pre}(s, x := x + \delta_1) \wedge \mathsf{pre}(\mathsf{post}(\Phi, x := x + \delta_1), x := x + \delta_1)) \vee \cdots \\
=& \mathsf{pre}(s \wedge \mathsf{post}(\Phi, x := x + \delta_1), x := x + \delta_1) \vee \cdots \\
\Rightarrow& \mathsf{pre}(s \wedge \exists\delta.\mathsf{post}(\Phi, x := x + \delta), x := x + \delta_1) \vee \cdots \\
=& \mathsf{pre}(s \wedge suc_t(\Phi), x := x + \delta_1) \vee \cdots \\
=& pre_t(s \wedge suc_t(\Phi))
\end{aligned}
$$

■

Figure 12 illustrates the Lemma with 3 cases. In each case, the shadowed region at left-hand side graph represents $pre_t(s) \wedge \Phi$, while the shadow at the right-hand side is $pre_t(s \wedge suc_t(\Phi))$. In (a), both sets are empty; in (b), Φ and s are joined, while in (c), Φ and s are separated.

Theorem 5.4.2 (Soundness) *If $\Phi \vdash_P \psi$ has a valid proof, then $[\bigwedge \Phi \rightarrow \psi]_P = \mathfrak{D}^{\mathfrak{X}}$.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

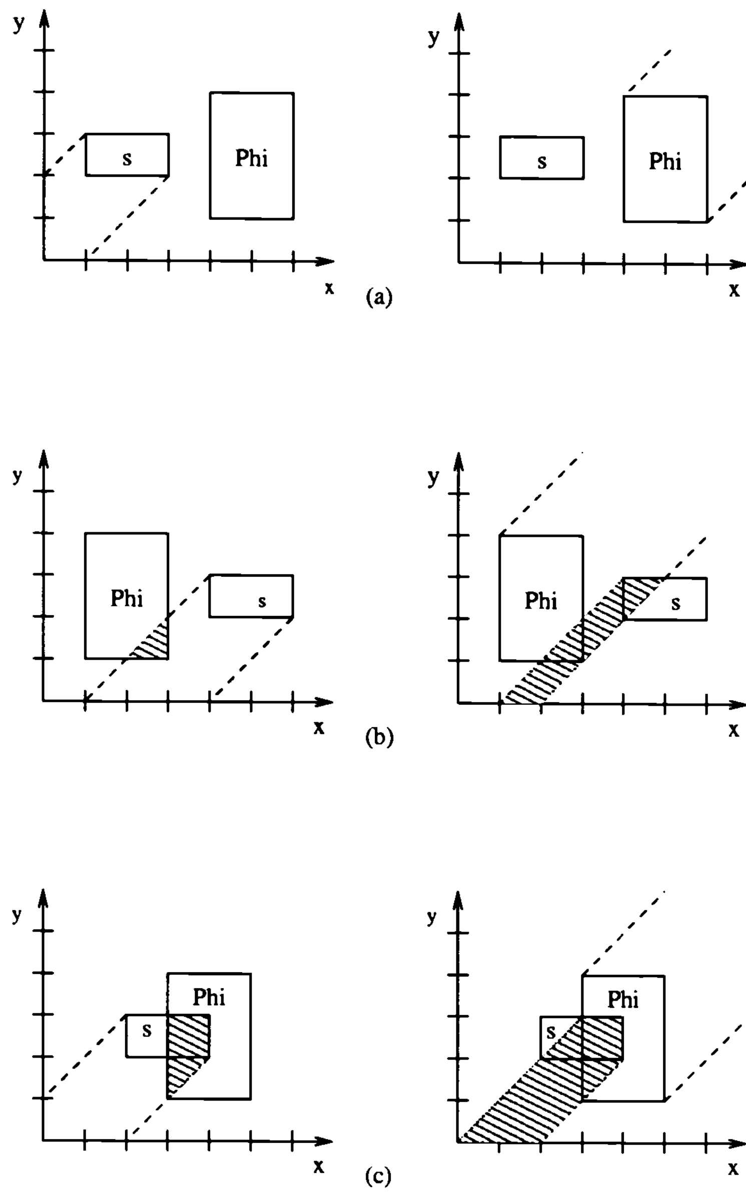


Figure 12: Examples for lemma 5.4.1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Proof: By induction over the derivation of $\Phi \vdash_P \psi$. The inductive step involves proving soundness of each rule in the proof system. And for each rule of the form

$$\frac{\sigma}{\sigma_1, \ldots, \sigma_n}$$

we have to show that if the soundness hold for each subgoal $\sigma_1, \ldots, \sigma_n$, then the goal σ is also sound.

The soundness of most of the rules is straightforward;

- The soundness of rule $\vee_1 - \wedge$, rule CALL and rule $[]_1$ is obvious.
- Rule $\vee$ is sound because $\neg\Phi \vee \neg s \vee \psi$ and $\neg\Phi \vee s \vee \psi$ imply $\neg\Phi \vee \psi$.
- Rule $\forall_1$ is a bit more complicated. It depends on a derived sound and complete rule

  $$\frac{\phi_1 \vee \phi_2 \vdash_P \psi}{\phi_1 \vdash_P \psi \ ; \ \phi_2 \vdash_P \psi}$$

  The left-hand side of the sequent $\exists\delta.\mathbf{post}(\phi, \bar{x} := \bar{x}+\delta) \vdash_P \psi$ is an infinite disjunction. So the sequent is equivalent to a list of subsequents

  $$\mathbf{post}(\phi, \bar{x} := \bar{x} + \delta_1) \vdash_P \psi \ ; \ \ldots \ ; \ \mathbf{post}(\phi, \bar{x} := \bar{x} + \delta_n) \vdash_P \psi$$

  Since the completeness of rule $[]_1$, this list is equivalent to

  $$\phi \vdash_P \psi[\bar{x} := \bar{x} + \delta_1] \ ; \ \ldots ; \ \phi \vdash_P \psi[\bar{x} := \bar{x} + \delta_n]$$

  From the completeness of rule $\wedge$, we have

  $$\phi \vdash_P \psi[\bar{x} := \bar{x} + \delta_1] \wedge \ldots \wedge \psi[\bar{x} := \bar{x} + \delta_n]$$

  which is equivalent to

  $$\phi \vdash_P \forall\delta.\psi[\bar{x} := \bar{x} + \delta]$$

- The soundness of Rule $\forall_2$ follows rule $\forall_1$. Since from $suc_t(\Phi \wedge s) \vdash_P suc_t(\Phi) \wedge s'$ and $suc_t(\Phi) \wedge s' \vdash_P \psi$, we can conclude that $suc_t(\Phi \wedge s) \vdash_P \psi$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Rule $\exists_1$ is sound. The proof depends on the Lemma 5.4.1, $pre_t(s') \wedge \Phi \Rightarrow pre_t(s' \wedge suc_t(\Phi))$. And from $s \wedge suc_t(\Phi) \vdash_P \psi$ we can infer that

  $$pre_t(s \wedge suc_t(\Phi)) \vdash_P pre_t(\psi).$$

  Then we have

  $$pre_t(s) \wedge \Phi \Rightarrow pre_t(\psi).$$

  Combining with $\Phi \vdash_P pre_t(s)$, the target sequent holds

  $$\Phi \vdash_P pre_t(\psi)$$

- Rule $\exists_2$ is sound. The proof depends on the Lemma 5.4.1, $pre_t(s') \wedge \Phi \Rightarrow pre_t(s' \wedge suc_t(\Phi))$. And from $s' \wedge suc_t(\Phi) \vdash_P \psi$ we can infer that

  $$pre_t(s' \wedge suc_t(\Phi)) \vdash_P pre_t(\psi).$$

  Then we have

  $$pre_t(s') \wedge \Phi \Rightarrow pre_t(\psi).$$

  Combining with $s \vdash_P pre_t(s')$, the target sequent holds

  $$\Phi, s \vdash_P pre_t(\psi)$$

- The soundness of rule $[]_2$ can be reasoned as follows. From $s \vdash_p \mathsf{pre}(s', A)$ we have $\mathsf{post}(s, A) \vdash_P s'$, which together with $\mathsf{post}(\Phi, A), s' \vdash_P \psi$ let us get $\mathsf{post}(\Phi, A) \wedge \mathsf{post}(s, A) \vdash_P \psi$. We then have the following sequent hold,

  $$\mathsf{pre}(\mathsf{post}(\Phi, A) \wedge \mathsf{post}(s, A), A) \vdash_P \psi[A].$$

  Since pre operation can distribute over conjunction, we get,

  $$\mathsf{pre}(\mathsf{post}(\Phi, A), A) \wedge \mathsf{pre}(\mathsf{post}(s, A), A) \vdash_P \psi[A].$$

  Finally we have,

  $$\Phi \wedge s \vdash_P \psi[A].$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Rule **LEAF** is used to generate the initial splitting constrains. When $\Phi \vdash_P \varphi$ is a tautology, the weakest splitting s is **tt**; when $\Phi \vdash_P \varphi$ is a contradiction, the weakest s is **ff**; otherwise we can simply set s as φ, note that φ is a predicate-closed formula. Then s is propagated back to enable the computation of previous splitting constraints.

■

In general, the proof rules will not be complete for arbitrary parametric model-checking problem. 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, the completeness does hold. The proof follows from the finiteness of the number of different regions in the systems [3] and an argument for fixpoint approximation similar to [64, 69].

5.5 Implementation

We have implemented a prototype, which we call CWB-RT (Concurrency Workbench - Real Time), of the above-mentioned algorithm. For the non-parametric version of the algorithm, we implemented the difference bound matrices (DBMs) [50] to represent state predicates; while for the parametric version, we implemented the *parametric difference bound matrices* (PDBMs) [66] package. C++ was used as the implementation language.

Difference Bound Matrix Given a parameter valuation, clock zone can be efficiently represented using matrix. Suppose the PES contains m clocks $x_1, \ldots, x_m$, then a clock zone can be encoded as a $(m+1) \times (m+1)$ square matrix M whose indices ranging from 0 to m and whose elements belong to $\{\{<, \leq\} \times \mathbb{Z}\} \cup \{\infty\}$. For each i, the entry M_{0i} encodes the lower bound of the clock x_i, while the entry M_{i0} specifies the upper bound of the clock x_i.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The element $M_{i,j}$, for $0 < i,j \leq m$, encodes the constraint $x_i - x_j < c$ if $(<, c)$ is the entry and $x_i - x_j \leq c$ if $(\leq c)$ is the entry. If the entry for $M_{i,j}$ is ∞ then no bound is specified for the difference of $x_i - x_j$. Bounds can be ordered naturally as follows. Let $\preceq \in \{<, \leq\}$ and $<$ be strictly less than $\leq$, $(\preceq, d) \leq (\preceq', d')$ iff $d < d'$ or $d = d'$ and $\preceq \leq \preceq'$. The semantics of a DBM M, written as $[\![M]\!]$, is defined as the set of clock valuations that satisfy the clock zone represented by the matrix. We call M is *satisfiable* if $[\![M]\!]$ is nonempty.

Example 5.5.1 *Considering the clock zone*

$$(1 < x_1) \wedge (x_2 - x_1 < 0) \wedge (x_1 - x_2 < 1) \wedge (x_2 < 2)$$

which is represented by two matrix in Figure 13.

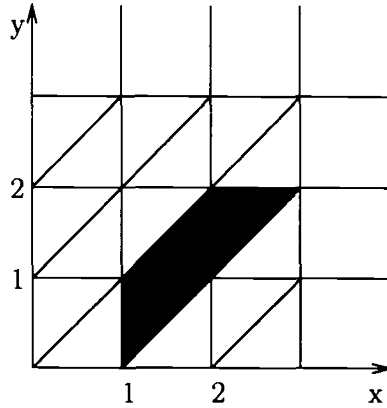
Matrix M' is obtained from matrix M by tightening all the constraints. Such a tightening can be computed by the Floyd-Warshall algorithm [1]. Its time complexity is $\mathcal{O}(m^3)$, where m is the number of clocks. Matrix like M' with tightest possible bounds are called *canonical*. Two canonical matrix M, M' represent the same constraint iff $M_{ij} = M'_{ij}$ for all $0 \leq i,j \leq m$. What's more, if $M_{ij} \leq M'_{ij}$ for all $0 \leq i,j \leq m$, we can conclude that the two zones $M \subseteq M'$.

Since it is expensive to compute the canonical form of a matrix, it is desirable to make frequently used operations over DBM preserve its canonicity.

- The **emptiness**(M) operation is used to detect consistency of a DBM, i.e. to test whether M is satisfiable. The most efficient way to determine emptiness is to detect whether there exists a clock difference whose upper bound has a smaller value than the lower bound in the canonical form.

- The **conjunction**(M, M') of two clock zone M, M' can be computed by $(M \wedge M')_{ij} = min(M_{ij}, M'_{ij})$, for all $0 \leq i,j \leq m$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



M	x_0	x_1	x_2
x_0	$(\leq,0)$	$(<,-1)$	∞
x_1	∞	$(\leq,0)$	$(<,1)$
x_2	$(<,2)$	$(<,0)$	(≤ 0)

M'	x_0	x_1	x_2
x_0	$(\leq,0)$	$(<,-1)$	$(<,0)$
x_1	$(<,3)$	$(\leq,0)$	$(<,1)$
x_2	$(<,2)$	$(<,0)$	(≤ 0)

Figure 13: Representation of a clock zone

- The $suc_t(\phi)$ operation preserves the canonical form. This is because clock difference remains the same as time elapse, lower bounds do not change either, while upper bounds have to be pushed to infinity. Thus for a canonical representation of matrix M, $suc_t(M)$ is computed by setting the upper bound on each individual clock to ∞.

- The $pre_t(\phi)$ can be computed by setting all the lower bounds on individual clocks to $(\leq, 0)$. However, due to the constraints on clock difference, this operation may not preserve the canonicity. An $\mathcal{O}(m^2)$ algorithm [21]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

exists to compute the canonical form from the intermediate matrix.

- The **reset**$(M, x_i := 0)$ operation is computed by setting all M_{i0} and M_{0i} to $(\leq, 0)$ and removing all other bounds on x_i.

- The **preset**$(M, x_i := 0)$ operation computes the weakest precondition of M with respect to the clock reset. It removes all constraints on clock x_i and sets M_{0i} as (≤ 0). However the result may not in its canonical form.

- The **zone difference** $M - M'$ is computed by successively slicing off parts of M that do not lie in M' [4].

- The **disjunction** of two clock zone M, M' is not necessarily a DBM. That is, the disjunction of two constraints may not be convex.

Parametric Difference Bound Matrices PDBMs extends DBMs with linear parameter terms as matrix entries. That is, the entry M_{ij} is now a pair $(\{<, \leq\}, t)$, where t is a linear term defined over constants and parameter variables. Given a parameter valuation ω, a PDBM becomes a DBM, whose semantics is written as $[\![M]\!]_w$. A constrained PDBM is a pair (Φ, M), where Φ is a set of constraints over parameter variables, and M is a PDBM. The semantics of a constrained PDBM $[\![(\Phi, M)]\!] \stackrel{\text{def}}{=} \cup_{\omega \models \Phi} [\![M]\!]_w$ A constrained PDBM (Φ, M) is *satisfiable* if $[\![(\Phi, M)]\!]$ is nonempty.

Here are some basic operations over constrained PDBMs [66].

1. **Adding a guard**. In case of DBM, adding a guard $g : x_i - x_j \preceq d$ to a zone M is a simple operation, i.e. to determine whether $(\preceq, d) < M_{ij}$ and update the bound, written as $M[g]$, and compute the canonical form if so. While in the parametric case, adding a guard to a constrained PDBMs (Φ, M) may result in a set of constrained PDBMs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Let $g : x_i - x_j \preceq t$ be the adding guard and $M_{ij} = (\preceq_{ij}, t_{ij})$, $i \neq j$. Define the boolean operation $\Rightarrow$ as over relation symbols $\leq$ and $<$ as what follows. $(\leq \Rightarrow <) = <$; $(\leq \Rightarrow \leq) = \leq$; $(< \Rightarrow \leq) = \leq$; $(< \Rightarrow <) = \leq$. This operation is used to check whether the bound imposed by the guard is weaker than the corresponding bound in the PDBM.

The resulted set $\mathcal{M}$ of constrained PDBMs are computed as follows,

$$\mathcal{M} = \begin{cases} \{(\Phi, M)\} & \textbf{if } C \models (t_{ij} \Rightarrow t) \\ \{(\Phi, M[g])\} & \textbf{if } C \models \neg(t_{ij} \Rightarrow t) \\ \{(\Phi \cup (t_{ij} \Rightarrow t), M), (\Phi \cup \neg(t_{ij} \Rightarrow t), M[g])\} & \textbf{otherwise} \end{cases}$$

We took the Omega Library [86] as decision procedure for inclusion checking between these linear terms.

2. **Canonicalization** In the parameter case, the canonical form of a constrained PDBM can also be computed with Floyd-Warshall algorithm symbolically.

BDD-like data structure BDD-like data structures help to improve the performance of real-time verification in both space and time complexities with intensive data-sharing in the representation of state space [20, 83, 105, 106].

CRD(Clock-Restriction Diagram) [105] is one of most advanced BDD-like data structures for the verification of timed automata. CRD can be seen as a decision diagram for zone set membership. Each *evaluation variable* in a CRD is of form $x_i - x_j$, and the values of such variable are $\{\{<, \leq\} \times \mathbb{Z}\} \cup \{\infty\}$, just like the entry for DBM.

By fixing an order of the evaluation variables, a CRD can be constructed in a similar way as BDD. In CRDs, a missing evaluation variable, e.g. $x_i - x_j$, is interpreted as $x_i - x_j < \infty$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

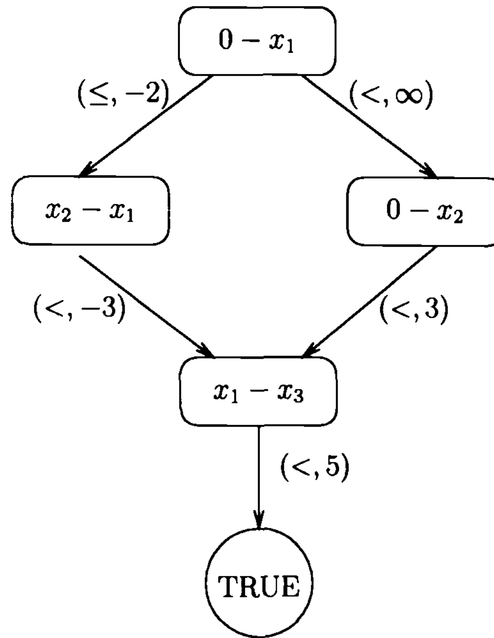


Figure 14: Example for CRD with upper bounds

Example 5.5.2 *Considering the CRD illustrated in Figure 14, which represents the union of two zones,* $\{0 - x_1 \leq -2, x_2 - x_1 < -3, x_1 - x_3 < 5\}$ *and* $\{0 - x_2 < 3, x_1 - x_3 < 5\}$.

HRD(Hybrid-Restriction Diagram) [106] is the extension of CRD for the parametric analysis of linear hybrid systems. In HRDs, evaluation variables are linear terms of the form $\sum_i a_i x_i$, which together with the outgoing arc label $(\preceq, c)$ constitute the linear constraint defined over clock variables and parameter variables.

With a canonical form for CRD/HRD and basic set-oriented manipulations on CRDs/HRDs, verification can be performed efficiently. As BDD-like data structures, the efficiency of CRD/HRD-based fixpoint computation is strongly dependent on the ordering of evaluation variables.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Implementation The algorithm uses a depth-first search technique with caching. The proof rules in Figure 9 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 [11, 25], so we omit further discussion here. The pseudo-code for the algorithm is presented in Table 1 and continued in Table 2

A sequent is defined as a tuple $\langle split, \Sigma, \mathsf{lhs}, \mathsf{rhs}, sub, value\rangle$,where lhs is a zone; *split* indicates whether splitting constraints are necessary, which are stored in a zone set Σ; rhs is a predicate calculus expression, whose kind may be PREDICATE, AND, OR, FORALL, EXISTS, CONSTRAINT, BOOL, ATOMIC, SUBLIST, RESET according to the definition of predicate calculus; *sub* is a mapping function. The procedure PROOF takes *split*, lhs, rhs, *sub* as input parameters and returns the value of the sequent, where splitting constraints are stored as a list of PDBMs in Σ. The algorithm starts from PROOF(*false*, $\emptyset, c, X$, *sub*), where c is the zone that represents $\bar{x} := 0$, X is the interested predicate and *sub* assigns initial value to each control variable.

Sequents are cached to share computations. The value of a cached sequent can be looked up by **tabled** in line (1). Its value my also need to be updated due to dependent information later by **update_tabled_sequents**. Dependent relationship between predicate variables can be defined as dependent trees in the same way as [40].

Zones operation $\cap(\Sigma_1, \Sigma_2)$ takes two zone sets and outputs a zone set Σ guided by $\rho \in \bigvee \Sigma$ iff $\rho \in \bigvee \Sigma_1 \wedge \rho \in \bigvee \Sigma_2$; Similarly operation $\cup(\Sigma_1, \Sigma_2)$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 1: Pseudo-code for local real-time model-checking algorithm

```
bool PROOF(bool split, zones &Σ, zone &lhs, ExprNode &rhs, Subst &sub)
(1)   IF tabled(split, Σ, lhs, rhs, sub), RETURN the_tabled_value;
(2)   bool retval ;
(3)   SWITCH(rhs.kind())
(4)   CASE PREDICATE:
(5)     IF a leaf is determined, RETURN the_leaf_value;
(6)     get the ExprNode e that defines the predicate variable of rhs;
(7)     retval = PROOF(split, Σ, lhs, e, sub);
(8)   CASE AND:
(9)     retval = PROOF(split, Σ1, lhs, rhs.left(), sub)
                 ∧ PROOF(split, Σ2, lhs, rhs.right(), sub);
(10)    IF (split) Σ = Σ1 ∩ Σ2;
(11)  CASE OR:
(12)    IF (split) retval = PROOF(split, Σ1, lhs, rhs.left(), sub)
                   ∨ PROOF(split, Σ2, lhs, rhs.right(), sub);
(13)      Σ = Σ1 ∪ Σ2;
(14)    ELSE
(15)      IF (retval = PROOF(true , Σ1, lhs, rhs.left(), sub))
(16)        generate zones Σ2 = (¬Σ1 ∩ lhs);
(17)        FOR each z ∈ Σ2
(18)            retval = retval ∧ PROOF(false , ∅, z, rhs.right(), sub);
(19)      ELSE retval = PROOF(false , Σ, lhs, rhs.right(), sub);
(20)  CASE FORALL:
(21)    IF (split) retval = PROOF(split, Σ', suc_τ(lhs), rhs.expr(), sub)
                     ∧ forallcond(Σ,Σ',lhs);
(22)    ELSE retval = PROOF(split, Σ, suc_τ(lhs), rhs.expr(), sub);
(23)  CASE EXISTS:
(24)    retval = PROOF(true , Σ', suc_τ(lhs), rhs.expr(), sub)
                 ∧ existcond(split, Σ, Σ', lhs);
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 2: Pseudo-code for local real-time model-checking algorithm (cont)

```
(25)  CASE CONSTRAINT:
(26)    IF (split)
(27)      IF (lhs ≤ rhs.zone())   Σ = UNIVERSE;   retval = true ;
(28)      ELSE IF (lhs ∩ rhs.zone() == ∅ ) Σ = EMPTY; retval = false ;
(29)           ELSE  Σ = {rhs.zone()};   retval = true ;
(30)    ELSE IF (lhs ≤ rhs.zone()) retval = true ;
(31)         ELSE retval = false ;
(32)  CASE BOOL: retval = rhs.bool();
(33)  CASE ATOMIC: retval = (sub(rhs.atomic()) == rhs.intval());
(34)  CASE SUBLIST:
        retval = PROOF(split, Σ, lhs, rhs.expr(), rhs.sub()[sub]);
(35)  CASE RESET:
(36)    IF (split) retval =
          PROOF(split, Σ', reset(lhs, rhs.clockSet), rhs.expr(), sub);
(37)      Σ = preset(Σ', rhs.clockSet);
(38)    ELSE retval =
          PROOF(split, Σ, reset(lhs,rhs.clockSet), rhs.expr(), sub);
(39)  update_tabled_sequents(split, Σ, lhs, rhs, sub, retval );
(40)  RETURN retval ;
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

defines a zone set Σ by $\rho \in \bigvee\Sigma$ iff $\rho \in \bigvee\Sigma_1 \vee \rho \in \bigvee\Sigma_2$; complement of a zone set $\rho \in \bigvee\overline{\Sigma}$ iff $\rho \notin \bigvee\Sigma$. We sometime abuse these notations by providing a zone instead of a zone set as an input. Zone operation $\leq(z_1, z_2)$ iff $\rho \in z_1 \rightarrow \rho \in z_2$.

We use function **forallcond**($\Sigma, \Sigma', \mathsf{lhs}$) to determine the weakest Σ given by the condition $\mathsf{suc}_\tau(\mathsf{lhs} \cap \Sigma) = \mathsf{suc}_\tau(\mathsf{lhs}) \cap \Sigma'$; To do this, we can first test whether $\mathsf{suc}_\tau(\mathsf{lhs}) \cap \Sigma'$ has a ray to positive infinity and then define Σ in the way as $z \in \Sigma$ iff $z \in \Sigma' \wedge (z \cap \mathsf{lhs}) \neq \emptyset$; While **existcond**(*split*, $\Sigma, \Sigma', \mathsf{lhs}$) defines $\Sigma = \mathsf{pre}(\Sigma')$ when *split* is true. If *split* is false, it returns false when there is any $\rho \in \mathsf{lhs}$ and $\rho \notin \bigvee \mathsf{pre}(\Sigma')$)

Operation **reset**(z, *clockSet*) computes the strongest post-condition with respect to the reset clocks; while **preset**(z, *clockSet*) gives the weakest pre-condition.

All other operations over the class **ExprNode**, like *left*(), *right*(), *expr*(), *sub*(), *atomic*(), *intval*(), *bool*(), *zone*(), *clockSet*, works in a straight way. In line (34), *sub*()[*sub*] is the updating operation over mapping functions. UNIVERSE represents the universe constraint, EMPTY the empty constraint.

To identify leaves, a stack is useful to cache sequents whose **rhs** is PREDICATE. Note that leaves can be identified even with unknown splittings.

5.6 Experimental Results for Real-Time

To assess the performance of the non-parametric algorithm, we ran CWB-RT on several examples taken from the literature and compared the results with those from the most recent available versions of Kronos [109] (2.5i.2), UPPAAL [19] (3.4.7, with both breadth-first (-b) and depth-first (-d) search options) and RED [105] (5.3, with both forward and backward analysis). The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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)* [4, 105]. 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 Δ_B seconds before assigning its identifier to p. It may enter the critical section within Δ_C seconds provided p still equal to its identifier. It reinitializes p to 0 upon leaving the critical section. When $\Delta_B > \Delta_C$ two processes may enter the critical section at the same time. The constants we use are $\Delta_B = 10, \Delta_C = 19$. 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* [47, 105]. A network is consisted of n identical stations and a ring, where the stations can communicate by synchronous messages with high priority and asynchronous messages with low priority. For each station, two clocks are used. The biggest timing constant used is $50 * n + 20$, where n is the number of stations. 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)* [15, 105]. Each process runs with a distinct priority in a period equal to the number

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of processes. Scheduling policy must follow priority among processes. 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)* [105]. Each process has a unique identifier greater than 0 and a control variable p which records its parent and is initialized to 0. A process with $p = 0$ may broadcast its request to be adopted by a parent. Another process with $p = 0$ may respond. Then the process with smaller identifier will become the parent of the other one. The biggest timing constant used is 2. 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)* [105]. 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* [105, 109]. 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 in Table 3 for the model checkers in question. These figures indicate that CWB-RT performs much better than the other tools in this case.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 3: Non-parametric real-time performance data when correct systems fail buggy (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 non-parametric	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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 4: Non-parametric real-time performance data for buggy system specifications and correct (a) 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 non-parametric	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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 5: Non-parametric real-time performance data for correct systems and (a) 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 non-parametric	Kronos 2.5i.2	UPPAAL 3.4.7 (-b)	UPPAAL 3.4.7 (-d)	RED 5.3 (forward)	RED 5.3 (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

We then studied situations in which correct formulas were used but buggy system specifications given. The data we obtained is given in Table 4, 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 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 in Table 5 are for the correct versions of the case studies against (a) properties. 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 [109] was implemented with DBMs, as CWB-RT is, UPPAAL [19] use CDDs and RED5.3 CRDs [105]. We conjecture that CWB-RT would see considerable performance improvement if we used CDDs / CRDs

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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.

5.7 Experimental Results for Parametric Real-Time

To assess the performance of the parametric 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 TReX-1.4 [13], HyTech-1.04f [63] and RED5.3 [106] 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 [13] 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 [63] and RED [106] 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 [106] for the experimental results. Due to the absence of publicly available implementations, other constraint-synthesis tools that are capable of parametric analysis, namely LPMC [97] and the extension of UPPAAL for linear parametric model checking [66], are not considered here; see [44] for the performance reports.

The experimental platform used was an Intel Pentium IV 2.8GHz with

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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)* [4, 106]. We verify that at any time, no more than one process is in its critical section, when (a). $\Delta_B >= \Delta_C$; (b). $\Delta_B < \Delta_C$; (c). $\Delta_B > 0, \Delta_C > 0$.

2. *Nuclear reactor controller (REACTOR)* [6, 106]. 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). $T >= 16 + (m - 1) * 21$; (b). $T < 16 + (m - 1) * 21$; (c). $T >= (m - 1) * 21$, 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 [106]. 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 δ 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). $\delta <= 20$; (c). $\delta > 10$.

4. *CSMA/CD benchmark* [106, 109]. This is the ethernet bus arbitration protocol with the idea of collision-and-retry. A typical worst case round trip propagation is δ time units, and it need σ time units to detect a collision. One safety property requires that at any moment, at most one

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

process is in the transmission mode for no less than δ time units. We check (a). $\delta <= \sigma$; (b). $\delta > 52, \sigma <= 26$; (c) $\delta > 52, \sigma > 0$.

We tested the fast error-detection capability of the parametric CWB-RT with buggy condition (a) and (c) over parameters, we collected comparative performance data for the model checkers in question. These figures in Table 6 and Table 7 indicate that CWB-RT significantly outperforms the other tools in this case. Again, we conjecture that CWB-RT's superior performance in this 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.

The performance figures collected in Table 8 are from all (b) properties of these case studies. Specifically, it can be seen that CWB-RT generally outperforms TReX-1.4 and HyTech-1.04f. RED-5.3 generally outperforms CWB-RT on these examples, although it should be noted that while TReX [109] was implemented with PDBMs, as CWB-RT is, HyTech [19] use polyhedra and RED5.3 HRDs [106]. Since data structures have been one of the key challenges for efficient real-time model checking [105, 106], we conjecture that CWB-RT 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-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.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 6: Parametric real-time 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-RT	TReX1.4		HyTech1.04f		RED5.3	
	parametric	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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 7: Parametric real-time performance data with (c) 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-RT	TReX1.4		HyTech1.04f		RED5.3	
	parametric	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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 8: Parametric real-time performance data with (b) 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-RT	TReX1.4		HyTech1.04f		RED5.3	
	parametric	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

Chapter 6

Model Checking Presburger Systems with PESs

This section introduces a proof-based symbolic model-checking technique for Presburger systems. This technique will also serve as the basis of our query-checking approach.

6.1 Presburger Systems

We begin by introducing some terminology and notation. Throughout let $(x, y, \ldots \in)\mathcal{X}$ be a set of data variables, $(a, b, \ldots \in)\mathbb{Z}$ be the set of integers, and $(t \in)\Pi$ be the set of linear terms of form of $\Sigma_{i=1}^{n} a_i x_i$. Also fix a set of (uninterpreted) actions $(\alpha, \beta, \ldots \in)\mathcal{A}ct$.

Presburger systems may be thought of as state machines that, in the course of their execution, may modify integer-valued data variables. The tests and modifications to these variables that these state machines may engage in must take the form of so-called Presburger formulas, which represent a restricted subset of logical formulas over integer arithmetic. In this section, we review the definition of Presburger formulas and introduce Presburger systems formally.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Presburger formulas Presburger formulas are generated by the following BNF grammar, where $x \in \mathcal{X}$ and $t_1, t_2 \in \Pi$.

$$\phi ::= t_1 \leq t_2 \mid \phi \wedge \phi \mid \neg\phi \mid \exists x.\phi$$

We use Φ to represent the set of Presburger formulas in what follows.

Semantically, Presburger formulas are interpreted with respect to *data states* $\rho \in \mathbb{Z}^{\mathcal{X}}$ mapping data variables to integers. We write $\rho \models \phi$ when ρ makes ϕ true; the definition is standard and is omitted. Formula ϕ is called *satisfiable* if there exists ρ such that $\rho \models \phi$.

The satisfiability of Presburger formulas is decidable, although the worst-case time bound is double-exponential in the length of the formula. Efficient procedures [71, 86] do exist to solve the satisfiability problems that arise most often in practice, which typically posses a small number of constraints and do not contain multiple levels of alternating quantifiers [32].

Finally, a Presburger formula ϕ defines a set $[\![\phi]\!]$ of data states in the obvious manner: $[\![\phi]\!] = \{\rho \mid \rho \models \phi\}$.

Presburger systems Presburger systems may be seen as symbolic state machines, with a finite sets of control locations and Presburger formulas and state transformations used to show how the data variables are modified as control locations are updated.

Definition 6.1.1 *A* Presburger system *(PS) is a tuple* $\langle S, R, S_I, \mathit{Init}\mathcal{C}\rangle$*, where* S *is a finite set of control locations;* $R \subseteq S \times \Phi \times \mathbb{A} \times \mathcal{A}ct \times S$ *is a finite set of transitions;* $S_I \subseteq S$ *are the initial locations; and* $\mathit{Init}\mathcal{C} \in \Phi$ *is the initial condition.*

Intuitively, S_I contains the possible starting locations and $\mathrm{Init}\mathcal{C}$ the initial conditions on data variables. Based on the current control location and state of the data variables, transitions whose $\phi \in \Phi$ components are true may fire,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

with data variables being updated in a manner consistent with the transition's state-transformation formulas. These notions are made precise by interpreting PSs semantically using *concrete transition systems.*

Given a PS $G = \langle S, R, S_I, \text{Init}\mathcal{C}\rangle$, CTS $C_G = \langle \Sigma, V, \rightarrow_c, \Sigma_I\rangle$ is given as follows.

1. $\Sigma \subseteq S \times \mathbb{Z}^{\mathfrak{X}}$

2. $V(\langle s, \rho\rangle) = \rho$

3. $\langle s, \rho\rangle \xrightarrow{\alpha} \langle s', \rho'\rangle$, iff there is $\langle s, \phi, A, \alpha, s'\rangle \in R$, with $\rho \models \phi$ and $(\rho, \rho') \models A$

4. $\Sigma_I = \{\langle s_I, \rho\rangle \mid s_I \in S_I, \rho \models \text{Init}\mathcal{C}\}$

As an example, we define the PS $G = \langle S, R, S_I, \text{Init}\mathcal{C}\rangle$ for the system in Figure 1 as follows, where τ is a special internal action.

- $S = \{s_0, s_1\}$

- $$R = \bigcup \begin{array}{l} \{s_0, x < 5; x' = x + 1; \tau; s_0\} \\ \{s_0, y < 8; y' = y + 1; \tau; s_0\} \\ \{s_0, y \leq 5; ; \tau; s_1\} \\ \{s_1, x > 2; x' = x - 1; \tau; s_1\} \\ \{s_1, y > 3; y' = y - 1; \tau; s_1\} \\ \{s_1, x \geq 2; ; \tau; s_0\} \end{array}$$

- $S_I = \{s_0\}$

- $\text{Init}\mathcal{C} = (x = 2 \wedge y = 3)$

6.2 The Presburger Modal Mu-Calculus

Temporal-logic query checking requires a "base logic" in which to write system requirements. In query-checking work, variants of the temporal logic CTL [36]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

are typically used for this purpose. In this work, we consider a different, lower-level, but more expressive logic that allows the definition of recursive formulas. We call this logic the Presburger Modal Mu-Calculus.

Our formulas are defined using MESs, which consist of blocks of equations of the form $X = \psi$, where $X \in \mathbb{X}$ is a *formula variable* and ψ is a formula defined by the following grammar.

$$\psi \ ::= \ \phi_s \mid \psi \vee \psi \mid \psi \wedge \psi \mid \langle \alpha \rangle \psi \mid [\alpha]\psi \mid X$$

In the above, ϕ_s is a Presburger formula, and α is an action. Operators $\langle \alpha \rangle$ and $[\alpha]$ are called modal operators; these, together with $\vee$ and $\wedge$, are standard from the propositional modal mu-calculus [72].

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

$$\begin{array}{ll} \sigma \models_{C,\theta} \phi_s & \text{iff } \mathcal{V}(\sigma) \models \phi_s \\ \sigma \models_{C,\theta} X & \text{iff } \sigma \in \theta(X) \\ \sigma \models_{C,\theta} \langle \alpha \rangle \psi & \text{iff there is } \sigma' \text{ s.t. } \sigma \xrightarrow{\alpha}_c \sigma' \text{ and } \sigma' \models_{C,\theta} \psi \\ \sigma \models_{C,\theta} [\alpha]\psi & \text{iff for all } \sigma' \text{ s.t. } \sigma \xrightarrow{\alpha}_c \sigma',\ \sigma' \models_{C,\theta} \psi \end{array}$$

We define $[\![\psi]\!]_{C,\theta} = \{\sigma \mid \sigma \models_{C,\theta} \psi\}$. We may now apply the general fixpoint theory, to define the semantics of MESs.

As a modal mu-calculus, the Presburger MESs are expressive enough to encode many temporal logics, including CTL. We sometimes use these CTL operators as shorthands to specify query formulas whenever convenient. For example, the invariant property $\mathsf{AG}\phi$ can be defined as $X \stackrel{\nu}{=} \phi \wedge [\tau]X$, the reachability formula $\mathsf{EF}\phi : X \stackrel{\mu}{=} \phi \vee \langle \tau \rangle X$, where τ is a special action that can

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

label any transition (because there is no action label in CTL); the bounded liveness property $\mathsf{A}\phi\mathsf{W}\varphi$: $X \stackrel{\nu}{=} \varphi \vee (\phi \wedge [\tau]X)$; and $\mathsf{EX}\phi$: $\langle\tau\rangle\phi$ etc.

6.3 From Presburger Model Checking to PESs

The Presburger system model-checking problem asks: given PS G, formula-closed MES M and $X \in \mathsf{lhs}(M)$, does $G \models_M X$? This section shows how to translate this question into an equivalent one involving PESs.

The translation from an PS G and an MES M to a PES is achieved by constructing a PES equation for each control location in G and equation in M. Formally, we define a function F that, given an PS 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, $F(G, \langle B_1, \ldots, B_n\rangle) = \langle F(G, B_1), \ldots, F(G, B_n)\rangle$. And $F(G, B) = F(G, \langle p, \overline{E}\rangle)$ in turn yields a predicate equation block of form $\langle p, \overline{E'}\rangle$, where for each equation $X = \psi$ in $\overline{E}$ and control location s in G, there is an equation of form $Y_{s,X} = F(s, \psi)$ in $\overline{E'}$. $F(s, \psi)$ is defined as follows.

$$
\begin{aligned}
F(s, \phi_s) &= \phi_s \\
F(s, \psi_1 \vee \psi_2) &= F(s, \psi_1) \vee F(s, \psi_2) \\
F(s, \psi_1 \wedge \psi_2) &= F(s, \psi_1) \wedge F(s, \psi_2) \\
F(s, X) &= Y_{s,X} \\
F(s, \langle\alpha\rangle\psi) &= \bigvee\{\phi \wedge (F(s', \psi)[A]) \mid \langle s, \phi, A, \alpha, s'\rangle \in R\} \\
F(s, [\alpha]\psi) &= \bigwedge\{\phi \rightarrow (F(s', \psi)[A]) \mid \langle s, \phi, A, \alpha, s'\rangle \in R\}
\end{aligned}
$$

Theorem 6.3.1 *Let $G = \langle S, R, S_I, Init\mathcal{C}\rangle$ be an PS, and let M be a closed modal equation system. Then for any $s \in S$ and any $X \in lhs(M)$, we have that $[\![X]\!]_{G,M}(s) = [\![Y_{s,X}]\!]_{F(G,M)}$.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Proof: Proof follows Theory 4.4.3 as a specialization. ∎

6.4 Local Model Checking

We now customize the "goal-directed" proof system to the Presburger systems in Figure 15. The proof system establishes when a set of predicate-closed formulas $\Phi \stackrel{\text{def}}{=} \{\phi_1, \ldots, \phi_n\}$ implies a formula ψ potentially containing predicate variables from a PES P. The proof rules operates on *sequents* of the form: $\Phi \vdash_P \psi$, which we shall interpret as the formula $\bigwedge \Phi \rightarrow \psi$. The rules follow the following syntactic conventions: ϕ, ϕ_i, φ are predicate closed, while ψ, ψ_i need not be; and Φ, ϕ is short-hand for $\Phi \cup \{\phi\}$. Conclusions are also written above subgoals, which are separated by a ";".

$$\vee_1 \frac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi \vdash_P \psi_1} \qquad \vee_2 \frac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi \vdash_P \psi_2}$$

$$\vee_3 \frac{\Phi \vdash_P \phi \vee \psi}{\Phi, \mathsf{not}(\phi) \vdash_P \psi} \qquad \vee_4 \frac{\Phi \vdash_P \psi \vee \phi}{\Phi, \mathsf{not}(\phi) \vdash_P \psi}$$

$$\wedge \frac{\Phi \vdash_P \psi_1 \wedge \psi_2}{\Phi \vdash_P \psi_1 \quad ; \quad \Phi \vdash_P \psi_2} \qquad \vee \frac{\Phi \vdash_P \psi_1 \vee \psi_2}{\Phi, \phi \vdash_P \psi_1 \quad ; \quad \Phi, \neg\phi \vdash_P \psi_2}$$

$$[] \frac{\Phi \vdash_P \psi[A]}{\mathsf{post}(\Phi, A) \vdash_P \psi} \qquad C \frac{\Phi \vdash_P X}{\Phi \vdash_P \psi} \; (X = \psi \;\in P\,)$$

Figure 15: A local approach for Presburger systems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Rules $\vee_1 - \wedge$ are familiar from the standard predicate calculus; not function "drives" negations inside; Intuitively, rule $\vee$ is used for splitting conditions, The remaining rules are for the substitution operator and predicate variables. $\mathsf{post}(\Phi, A) \stackrel{\text{def}}{=} \{\rho' \mid \rho \models \Phi \text{ and } (\rho, \rho') \in A\}$ defines the strongest postcondition of Presburger formulas Φ wrt. the state transformation A. These postconditions may also be represented as Presburger formulas.

The rules also share the implicit side condition: they may only be applied to *non-leaf* sequents. These are defined in the samw way as Definition 3.5.1.

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 6.4.1 *The proof rules in Figure 15 are sound: if $\Phi \vdash_P \psi$ has a valid proof then $[\![\Phi \vdash_P \psi]\!]_P = \mathbb{Z}^{\mathcal{X}}$, where P is the PES containing the definitions of the predicate variables.*

Proof: Proof follows Theory 3.5.2. ■

In general, the proof rules will not be complete, although Presburger arithmetic is decidable. This is because proofs of certain sequents may require infinite-depth trees (i.e. fixpoint computation does not converge in finite time [45]). To overcome this, conservative approximation techniques and acceleration heuristics (e.g. [32]) may be needed to achieve convergence of an approximate fixpoint computation. On the other hand, if all the data variables are bounded (i.e. take values from a range), then the proof system is complete. In this case, the set of Presburger formulas are finite. The argument of the completeness is similar to the propositional model checking [39].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.5 Implementation and Performance Evaluation

Since the above-mentioned local model-checking technique also serves as the basis for our query checking in Chapter 7, we leave our description of the implementation and experimental results to Section 7.4, where model checking is considered as query checking without any placeholder. Typically, Table 10 and Table 11 shows that our model checker runs faster than the Action Language Verifier [17], the state-of-the-art model checker for Presburger systems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 7

Temporal-Logic Query Checking for Presburger Systems

In this chapter, we describe our query checking approach for the Presburger modal mu-calculus. Queries in our setting will consist of formulas in the mu-calculus augmented with *placeholders* of form $?_X$, where X is a formula variable used only for identification purposes. Placeholders may also have negation applied to them in queries: so $\neg ?_X$ may also appear within a query. A query may also have multiple placeholders $?_X, ?_Y$, etc., distinguished by the formula variables labeling the placeholders. For technical convenience, throughout of the paper we assume that placeholders are different from formula variables, and thus that queries are "formula closed". We call an occurrence of the placeholder $?_X$ is *positive* in a query ψ if it appears under no negation in the query ψ, and *negative* if it appears negated. A placeholder $?_X$ is *pure* in a query ψ if all of its occurrences have the same polarity (positive or negative), and *mixed* otherwise.

A *query problem* consists of a query formula ψ and a PS G; a *solution* to such a problem is an assignment of formulas to placeholders in ψ such that G satisfies the resulting mu-calculus ψ' obtained by replacing the placeholders in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ψ by their formulas. In general, a query problem can have many solutions; we are particularly interested in *strongest* / *weakest* solutions, when they exist.

Sometimes we are interested in a subset of data variables in a solution to a placeholder. This is done with a *projection* operator ": {}" after the placeholder. For example, suppose $\mathfrak{X} = \{x, y, z\}$, we only care about variable x and y, then $?_X : \{x, y\}$ projects the solution from $\mathbb{Z}^{\{x,y,z\}}$ to $\mathbb{Z}^{\{x,y\}}$.

7.1 A Simple Example

We use a simple example to show how our query checking works.

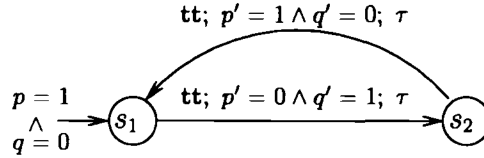


Figure 16: A simple transition graph

Considering the query formula $\mathsf{AG}?_X$, that is $Y \stackrel{\nu}{=} (?_X \wedge [\tau]Y)$, against the transition system in Figure 16, computing the "solution" for $?_X$ amounts to answering the question "What's the strongest invariant in the system?". Note that data variables p and q are unbounded. We show that this problem can be solved with our Gentzen-like proof system. Taking the placeholder $?_X$ as a Presburger formula, we translate the query formula and the system model into the following PES.

$$\begin{cases} Y_1 \stackrel{\nu}{=} & ?_X \wedge Y_2[p' = 0 \wedge q' = 1] \\ Y_2 \stackrel{\nu}{=} & ?_X \wedge Y_1[p' = 1 \wedge q' = 0] \end{cases}$$

Then the query checking problem is reduced to finding the strongest formula that can replace $?_X$ such that $\Phi \rightarrow Y_1$ is a tautology, where $\Phi \stackrel{\text{def}}{=} (p = 1 \wedge q = 0)$ is the initial condition of the transition system, and Y_1 is generated

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

from the initial control location s_1 and the formula variable Y. Applying the proof rules, we get the following tableau.

<table>
<tr><td colspan="3">$\Phi \vdash Y_1$</td></tr>
<tr><td colspan="3">$\Phi \vdash ?_X \wedge Y_2[p' = 0 \wedge q' = 1]$</td></tr>
<tr><td>$\Phi \vdash ?_X$</td><td colspan="2">$\Phi \vdash Y_2[p' = 0 \wedge q' = 1]$</td></tr>
<tr><td></td><td colspan="2">$\Phi \vdash ?_X[p' = 0 \wedge q' = 1] \wedge Y_1[p' = 1 \wedge q' = 0]$</td></tr>
<tr><td></td><td>$\Phi \vdash ?_X[p' = 0 \wedge q' = 1]$</td><td>$\Phi \vdash Y_1[p' = 1 \wedge q' = 0]$</td></tr>
<tr><td></td><td>$p = 0 \wedge q = 1 \vdash ?_X$</td><td>A ν-leaf is reached</td></tr>
</table>

For this tableau (proof structure) to be successful, all its leaves must be successful. Now:

- The ν-leaf $\Phi \vdash Y_1[p' = 1 \wedge q' = 0]$ is successful.
- For sequent $\Phi \vdash ?_X$ to be a successful leaf, we need $p = 1 \wedge q = 0 \rightarrow ?_X$ to be a tautology. The strongest formula to replace $?_X$ such that this sequent is valid would be Φ itself, that is $(p = 1 \wedge q = 0)$.
- Similarly, for leaf $p = 0 \wedge q = 1 \vdash ?_X$ to be successful, the strongest formula is $(p = 0 \wedge q = 1)$.

Therefore, to make the proof successful, we have the strongest

$$?_X \stackrel{\text{def}}{=} (p = 1 \wedge q = 0) \vee (p = 0 \wedge q = 1)$$

7.2 Existential Query Checking

The above example suggests an efficient symbolic approach for solving query problems: determine the solutions to placeholders at the leaves of a *potentially successful tableau* (PST) that arise when applying our model-checking procedure to a query. By PST, we mean a tableau whose leaves may contain

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

occurrences of placeholders and which could be identified as successful by appropriate assignments to these placeholders. For queries, it may be shown that leaves containing placeholders in a PST may have one of two forms: $\phi \vdash ?_X$ or $\phi \vdash \neg ?_X$, where ϕ is predicate closed and *placeholder closed*, (i.e. without any occurrence of any placeholder). We call such leaf sequents *potentially successful leaves*. Note that the occurrence of $?_X$ in $\phi \vdash ?_X$ is positive, and the occurrence of $?_X$ in $\phi \vdash \neg ?_X$ is negative.

Let G be a PS and ψ a query formula with (possibly multiple) placeholders$?_X(, \ldots, ?_Y)$. To solve ψ against G, i.e. give a solution to all the placeholders we compute the PES (augmented with queries) from the model-checking problem for G and ψ, treating placeholders like Presburger formula in the translation; we call the resulting extended PES a *query predicate*, since it contains placeholders. We then search for a potentially successful tableau T by applying proof rules to the query predicate.

Existential query checking provides solutions to placeholders according to a single (potentially) successful tableau. Once we have identified a potentially successful tableau T, we compute solutions for the placeholders as follows.

1. The solution to the *positive* occurrences of placeholder $?_X$ with respect to T, written as $[\![?^+_X]\!]_T$, is given by

$$[\![?^+_X]\!]_T \stackrel{\text{def}}{=} \bigvee\{\ \phi \ \mid\ \phi \vdash ?_X \text{ is a sequent in } T\}$$

2. The solution to the *negative* occurrences of placeholder $?_X$ with respect to T, written as $[\![?^-_X]\!]_T$, is given by

$$[\![?^-_X]\!]_T \stackrel{\text{def}}{=} \bigwedge\{\ \mathsf{not}(\phi)\ \mid\ \phi \vdash \neg ?_X \text{ is a sequent in } T\}$$

We now have the following.

Theorem 7.2.1 *Let T be a potentially successful tableau for a PES constructed from query formula ψ and PS G, and let $?_X$ be a placeholder. Then*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

replacing all occurrences of $?_X$ *in* T *by any formula* ϕ *such that* $[\![?^+_X]\!]_T$ *implies* ϕ *and* ϕ *implies* $[\![?^-_X]\!]_T$ *results in a tableau that can be extended into a successful tableau.*

Proof: The theorem holds apparently. ∎

In other words, this theorem asserts that $[\![?^+_X]\!]_T$ and $[\![?^-_X]\!]_T$ "bound" the solutions to the query problem that can be inferred from the PST T.

When $?_X$ is pure and positive (i.e. $?_X$ has no negative occurrences in ϕ), then it may be shown that $[\![?^-_X]\!]_T = \bigwedge \emptyset = true$; and similarly for the case when $?_X$ is pure and negative we have $[\![?^+_X]\!]_T = \bigvee \emptyset = false$. These observations lead to the following corollary of the above theorem.

Corollary 7.2.2 *Let* T *be a potentially successful tableau for a PES constructed from query formula* ψ *and PS* G.

1. *If* $?_X$ *is a placeholder in* ψ, *then* $[\![?^+_X]\!]_T$ *is the strongest formula* ϕ *such that* T *may be extended into a successful tableau when each positive occurrence of* $?_X$ *is replaced by* ϕ.

2. *If* $?_X$ *is a placeholder in* ψ, *then* $[\![?^-_X]\!]_T$ *is the weakest formula* ϕ *such that* T *may be extended into a successful tableau when each negative occurrence of* $?_X$ *is replaced by* ϕ.

For instance, the positive solution to the placeholder $?_X$ in section 7.1 is $[\![?^+_X]\!] \stackrel{\text{def}}{=} (p = 1 \wedge q = 0) \vee (p = 0 \wedge q = 1)$, while the negative solution is $[\![?^-_X]\!] \stackrel{\text{def}}{=} true$. Therefore, any formula that is implied by $(p = 1 \wedge q = 0) \vee (p = 0 \wedge q = 1)$ and implies *true*, e.g. $p = 0 \vee p = 1$, $q \leq 5$, or $p \geq 0$ etc., can be used to replace $?_X$ in the query formula $\nu Y = (?_X \wedge [\tau]Y)$, makes the model-checking problem successful.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

All the applications of the temporal-logic query checking established for propositional systems [61], such as *reachability analysis*, discovering invariants, *guard discovery*, *guided simulation* and *test case generation* etc. can be formulated immediately as existential query-checking for Presburger systems. The full paper will consider this point more fully.

7.3 Universal Query Checking

In general, a query problem may give rise to many PSTs, each yielding a bound on the solution for each placeholder. *Universal query checking* provides multiple solutions by the means of all (potentially) successful tableaux.

Previous works [61] shows that solving a propositional tempo-ral-logic query with a single placeholder takes 2^{2^n} times slower than checking an equivalent model-checking property, where n is the number of atomic propositions in the system. Application of these algorithms to industrial systems turns out to be impractical, since a moderate system might contain dozens of boolean variables, not mention of even a single unbounded integer variable in the system.

In our setting, universal query checking has to find all potentially successful tableaux, while existential query checking only needs to locate one (potentially) successful tableau; Consequently, existential query checking has the same time complexity as local model checking, while universal query checking might take longer (since tableaux must be enumerated). Therefore, the existential query checking has significant advantages in practice.

Note that for queries involving in an invariant or bounded liveness property, in which a single greatest fixpoint requires computing, universal query checking may have the same time complexity as the existential one, because there are only a very small number of potentially successful tableaux in most cases. For

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

example, the query $\mathsf{AG?}_X$ in Section 7.1 only leads to one PST.

7.4 Implementation

To evaluate the performance of our query-checking technique, we have built a prototype called CWB-QC (Concurrency Workbench–Query Checking). The algorithm uses a depth-first search technique with caching. The proof rules in Figure 15 are used to generate sequents needed to be proved next in order for the goal sequent to be true. The cache management are similar to the real-time cases.

Symbolic Representations Symbolic representations enable model-based analysis for large state spaces. They are one of the key elements to improve the performance. There are two basic approaches to symbolic representation of linear arithmetic constraints in verification.

1. **Polyhedra representation** This approach encodes linear arithmetic formulas in a disjunctive normal form where each disjunct corresponds to a convex polyhedra. Each disjunct corresponds to a conjunction of linear constraints [49, 63]. The Omega Library [86], which we have used as a decision procedure for the parametric real-time model checking, is specially tuned to solve integer problems in polyhedra representation. It also implements an extension of the Fourier-Motzkin linear programming algorithm [46].

2. **Automata-based representation** Automata-based representation of Presburger formulas dates back to at least Büchi [70]. Recent developments have proposed more efficient encodings [17, 27, 28, 35]. An arithmetic constraint ϕ over n integer variables can be encoded by a n-track deterministic finite-state automaton $\mathcal{A}_\phi$. The language recognized

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

by $\mathcal{A}_\phi$ corresponds to the set of all solutions to ϕ. Since each integer can be represented by the binary format in 2's complement, a solution to ϕ can represented as a vector of a n binary strings, with each binary string, or a tack, representing an integer for the corresponding variable. The $i-th$ column of the solution vector is the $i-th$ least significant bits of all variables. In what follows, we give a brief introduction [35]. We rewrite the formula $\sum_{1\leq i\leq n} a_i x_i \sim c$ as $a^T x \sim c$, where $\sim \in \{<, \leq, =, \geq, >\}$, $a^T \stackrel{\text{def}}{=} \{a_1, \ldots, a_n\}$, and

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

The automaton $\mathcal{A}_\phi \stackrel{\text{def}}{=} \langle S, s_0, S_{accept}, \{0,1\}^n, \delta \rangle$, where $S = \mathbb{Z} \cup s_0$ is the set of states, the initial state $s_0 \notin \mathbb{Z}$; S_{accept} is the set of accepting states, defined as $S_{accept} \stackrel{\text{def}}{=} \{l \in Z \mid l \sim c\}$, united with $\{s_0\}$ if $(-a^T b \sim c)$ holds; $\{0,1\}^n$ is the input alphabet. δ is the transition function $S \times \{0,1\}^n \rightarrow S$, defined as follows,

$$\delta(s_0, b) = -a^T \cdot b$$

$$\delta(l, b) = 2l + a^T \cdot b$$

where $l \in \mathbb{Z}$.

Let $\|a^T\|_- \stackrel{\text{def}}{=} \Sigma_{a_i \leq 0} |a_i|$ and $\|a^T\|_+ \stackrel{\text{def}}{=} \Sigma_{a_i > 0} a_i$, theoretically, once the automaton reaches a state outside of

$$[\|a^T\|_-\,,\ \|a^T\|_+]$$

it is guaranteed to stay outside of this range and on the same side of it. So all the states outside of the range can be collapsed into two states $-\infty$ and $+\infty$. As an example, Figure 7.4 shows an automaton for $x - y \leq 0$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

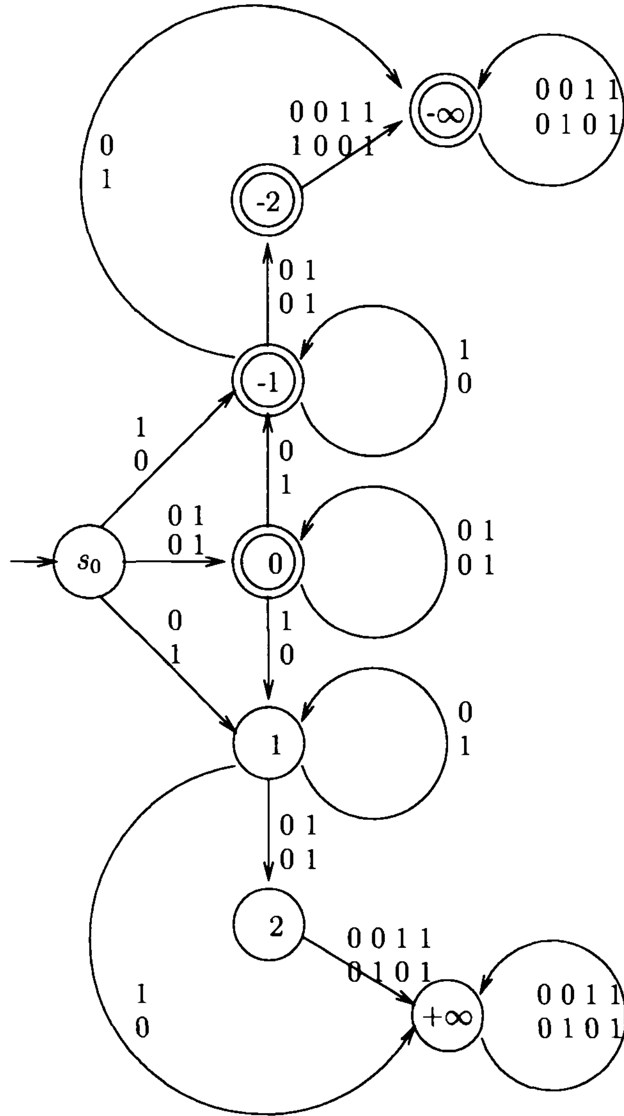


Figure 17: Automaton for $x - y \leq 0$

Both symbolic representations provides intersection, union, complement, existential quantifier elimination, and inclusion, emptiness, equivalence testing. Therefore they can be used in verification.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The implication-checking procedure we employ is the MONA tool [71], which provides an automata-based decision procedure for WS1S (weak monadic second order logic with one successor) and takes BDDs to represent the internal transitions . These features also make the MONA tool widely used by other verification tools [17].

Note that all the experimental data in this chapter was collected on an Intel Pentium III 700MHZ CPU and 512MB memory laptop, running Linux 2.4.

7.5 Case Study : A Simple Thermostat

As an example, we analyze the requirements of a simple thermostat; this example is adapted from the previous specification in [14]. The target system is responsible for keeping the room's temperature in a moderate range between *low* and *high* if *Switch* is on. The SCR specification [62] of the system is given in Figure 18.

An SCR requirements specification models the system in an event-driven fashion. The input interface of the system is given as a set of monitored variables and the output interface as a set of controlled variables. For example, the thermostat reads the sensor of the room temperature and the status of the switch; and it controls variables for the power switches of the heat and air conditioner. The state space is partitioned into sets of states called *modes*. The system changes its state due to conditioned events. For example, the event @T(SwitchIsOn) represents condition "switch is turned On from Off at next state" and @T(TooCold) describes the condition "*temp* < *low* becomes true at next state", while condition SwitchIsOn says "switch is On" at the current state.

Note that in the specification given in Figure 18, values of the constants

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Constants: low, high : Integer;
Monitored Variables: temp : Integer;
Switch : {On,Off}
Mode Class: Thermostat : {Off,Inactive,Heat,AC}
Initial Conditions: Switch=Off;
Thermostat=Off;
low < high;

TooCold $\stackrel{\text{def}}{=}$ temp < low
TempOK $\stackrel{\text{def}}{=}$ temp $\geq$ low & temp $\leq$ high
TooHot $\stackrel{\text{def}}{=}$ temp > high

Old Mode	SwitchIsOn	Event	New Mode
Off	@T	TooCold	Heat
	@T	@T(TooCold)	
	@T	TempOK	Inactive
	@T	@T(TempOK)	
	@T	TooHot	AC
	@T	@T(TooHot)	
Inactive	@F	–	Off
	t	@T(TooCold)	Heat
	t	@T(TooHot)	AC
Heat	@F	–	Off
	t	@T(TempOK)	Inactive
AC	@F	–	Off
	t	@T(TempOK)	Inactive

Figure 18: SCR specification of a simple thermostat

low, *high* are unspecified. These constants can take any integer value as long as they satisfy the ordering *low* $\leq$ *high*. Our representation of the system also leaves these constants as unspecified.

Modeling the thermostat as a PS We model the Thermostat as a symbolic transition system $T \stackrel{\text{def}}{=} \langle S, R, S_I, \text{Init}\mathcal{C}\rangle$, where $S = S_I = \{s\}$, $\text{Init}\mathcal{C} \stackrel{\text{def}}{=}$ (Switch = Off & Thermostat = Off), and the set of transition R is defined as

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

what follows.

⟨ s, Thermostat = Off & Switch = Off & temp $\leq$ low, Switch′ = On & Thermostat′= Heat, τ, s ⟩ ;

⟨ s, Thermostat = Off & Switch = Off & temp $\geq$ low, Switch′ = On & temp $<$ low & Thermostat′ = Heat, τ, s ⟩ ;

...

⟨ s, Thermostat = AC & Switch = On & (temp $>$ high | temp $<$ low), low $<$ temp′ $<$ high & Thermostat′ = Inactive, τ, s ⟩ ;

Query formulas Invariants summarize relationships between data variables in the model and are often useful to add confidence to system designers. For example, we can use the following queries to find interesting invariants in each mode.

(a). AG (Thermostat=Off $\rightarrow ?_{X_1}$:{Switch})

(b). AG (Thermostat=Inactive $\rightarrow ?_{X_2}$:{Switch,temp,low,high})

(c). AG (Thermostat=Heat $\rightarrow ?_{X_3}$:{Switch,temp,low})

(d). AG (Thermostat=AC $\rightarrow ?_{X_4}$:{Switch,temp,high})

To check the status of the thermostat under different conditions, one can use the queries

$$\text{AG (Thermostat=Heat)} \rightarrow \text{EX}?_{X_5} : \{\text{Thermostat}\}$$
$$\text{AG (Switch=On \& temp<low)} \rightarrow ?_{X_6} : \{\text{Thermostat}\}$$

The query AG($?_{X_7}$: {Switch} $\rightarrow$ Thermostat=Heat) can return the status of the switch when the Thermostat is heating, and AG$?_{X_8}$:{Thermostat} can return all reachable modes in the system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Performance results One can check each of the above query formulas separately, i.e. run CWB-QC multiple times, or query the conjunction of all the above formulas, i.e. run CWB-QC only once. We first take the former way and perform the existential query checking with CWB-QC. Each run takes about 0.01 seconds and a maximal memory of 3 megabytes. For the latter way, CWB-QC takes a total of 0.02 seconds and a maximal memory of 4 megabytes. Here are the output formulas to query predicates:

$[\![?^+_{X_1}]\!]_T$: Switch = Off

$[\![?^+_{X_2}]\!]_T$: Switch = On & low $\leq$ temp $\leq$ high

$[\![?^+_{X_3}]\!]_T$: Switch = On & temp < low

$[\![?^+_{X_4}]\!]_T$: Switch = On & temp > high

$[\![?^+_{X_5}]\!]_T$: Thermostat=Inactive | Off

$[\![?^+_{X_6}]\!]_T$: Thermostat=Heat

$[\![?^-_{X_7}]\!]_T$: Switch=On

$[\![?^+_{X_8}]\!]_T$: Thermostat=Off | Inactive | Heat | AC

Note that the system we check is unbounded since *low*, *high* remain unspecified. One cannot query such a system with a finite-state query checking algorithm like the one [61], without using some abstraction techniques.

7.6 Performance Comparisons

Due to the absence of publicly available implementations of the query-checking tools, we compare the performance of our query checker with our model checker, and with the Action Language Verifier (ALV) [17] (version 0.3) for Presburger systems.

Comparison with model checking The performance data are collected in Table 9. Besides the thermostat, we also check the cruise control system [14] for several invariant properties. Note that the performance of our model checker

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and existential query checker are virtually identical.

Table 9: Query checking performance comparison with model checking. The letters in the names of the systems refer to the indexes of properties; s : CPU time in seconds; k : maximum kilobytes of memory used by the verifier.

Example	CWB-QC query check	CWB-QC model check
thermostat-a	0.01s/3236k	0.01s/2812k
thermostat-b	0.01s/3316k	0.01s/2812k
thermostat-c	0.01s/3348k	0.01s/2808k
thermostat-d	0.01s/3304k	0.01s/2812k
cruise-a	0.02s/2932k	0.02s/2700k
cruise-b	0.02s/2528k	0.01s/2528k
cruise-c	0.01s/4088k	0.01s/3240k
cruise-d	0.02s/3940k	0.01s/3264k
cruise-e	0.01s/3508k	0.01s/3248k
cruise-f	0.02s/3084k	0.02s/2836k

Comparison with ALV ALV is a symbolic model checker for Presburger systems that which uses the Composite Symbolic Library (CSL) [108] as its symbolic manipulation engine. CSL combines different symbolic representations, which include the automata representation from the MONA package adopted by CWB-QC. To be fair for the performance comparison, we run ALV with the option "-F -I B" for forward and "-A -I B" for backward analysis respectively, and only automata representations are used.

Besides the above thermostat (all invariants are checked together), we have also verified the mutual exclusion properties for both bakery and ticket protocols, and an invariant property for the sleeping barber problem. The specifications for these systems are the same as [17]. The query formula we check for these examples is $\mathsf{AG}?_X$ and with projection over some data variables. Table 10 contains the performance data.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

These figures in Table 9 and Table 10 show that the query checking runs as fast as the model checking of CWB-QC, and the latter is more efficient than ALV. This fact together with the running time from the previous subsection indicates that our proof-based symbolic query checking technique can provide very efficient service to the design of Presburger systems in practice.

Fast Error-Detection of CWB-QC The (potentially) successful tableau constructed by CWB-QC provides a *witness* showing why the solution satisfies the query. On the other hand, a *counter-example* is reported when a formula is violated by the model. As an on-the-fly model checker, CWB-QC can detect errors quickly. We show this by checking buggy formulas for the above case studies.

The buggy formula we checked for the Thermostat is

$$\mathsf{AG}(\text{Thermostat=AC} \rightarrow \text{temp} < \text{high}).$$

The property we verified for both ticket and bakery protocol is whether it is allowed for a second process in the *try* mode while one is already in the *critical section*. The barber algorithm is checked with the negation of an invariant constraint. The performance data is reported in Table 11. CWB-QC generally outperforms ALV on these case studies. We conjecture that the superior performance in this case is due to the forward proof-based analysis of our technique.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 10: Query checking performance comparison with ALV-0.3.
The numbers in the names of the systems refer to the numbers of processes in the models. s : CPU time in seconds; k : maximum kilobytes of memory used by the verifier; N/A : "UNABLE TO VERIFY" reported by the model checker (in this case we still report the time and memory consumption); O/M: computation does not terminate within one hour.

Example	CWB-QC query check	CWB-QC model check	ALV-0.3 forward	ALV-0.3 backward
thermostat	0.02s/3804k	0.02s/2468k	0.11s/16288k	0.10/16192k
ticket-2	0.02s/3076k	0.01s/2468k	0.08s/15288k	N/A(0.14s/15520k)
ticket-3	0.16s/3864k	0.11s/3368k	0.30s/15896k	N/A(0.64s/16364k)
ticket-4	1.22s/5224k	1.04s/4760k	2.98s/19300k	N/A(5.26s/26492k)
ticket-5	14.21s/12056k	14.00s/11468k	20.83s/33552k	N/A(30.24s/61584k)
bakery-2	0.01s/2832k	0.01s/2656k	0.11s/15576k	0.04s/15228k
bakery-3	0.51s/5496k	0.49s/3476k	14.40s/28368k	N/A(0.79s/17604k)
barber-10	0.11s/4972k	0.10s/4688k	0.15s/16636k	0.16s/16952k
barber-12	0.12s/5500k	0.12s/4972k	0.17s/16924k	0.18s/17136k
barber-14	0.15s/5896k	0.15s/5376k	0.19s/17156k	0.19s/17560k
barber-16	0.18s/6496k	0.17s/5376k	0.21s/17360k	0.21s/17748k

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 11: Query checking performance comparison with ALV-0.3 for buggy properties.
The numbers in the names of the systems refer to the numbers of processes in the models. s : CPU time in seconds; k : maximum kilobytes of memory used by the verifier; O/M: computation does not terminate within one hour.

Example	CWB-QC model check	ALV-0.3 forward	ALV-0.3 backward
thermostat	0.00s/2784k	0.03s/15516k	0.02/16220k
ticket-2	0.01s/2848k	0.13s/15340k	0.06s/15360k
ticket-3	0.02s/2968k	2.64s/17644k	0.19s/16036k
ticket-4	0.03s/3432k	48.90s/45732k	1.45s/20796k
ticket-5	0.06s/4880k	820.84s/316988k	9.47s/37956k
bakery-2	0.00s/2468k	0.16s/15624k	0.07s/15116k
bakery-3	0.01s/2740k	13.79s/28504k	0.51s/17464k
barber-10	0.01s/2472k	O/M	0.17s/17024k
barber-12	0.01s/2704k	O/M	0.18s/17396k
barber-14	0.01s/3184k	O/M	0.21s/17410k
barber-16	0.01s/3272k	O/M	0.24s/17576k

Chapter 8

Conclusion and Future Work

In this dissertation, we have developed a generic model-checking framework for data-based systems. Existing model checking problems can be encoded via predicate equation systems. We have investigated how global model checking and local model checking techniques could be developed based on PESs. Especially, a Gentzen-like proof system is proposed for the local model checking via PESs. Two important applications of the local model checking technique have been studied for the domains of real-time systems and Presburger systems.

Real-time model checking We have presented an on-the-fly algorithm for solving the traditional real-time and universal parametric real-time model-checking problems based on PESs. Experimental results demonstrate that our proof-theoretic method is significantly superior to existing approaches for systems contain 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 in which model checkers are used "early and often" to detect design errors in an ongoing manner.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Temporal Logic Query Checking We have also proposed a framework for solving temporal-logic query checking for Presburger systems based on the local model-checking technique. Existential query checking returns only one solution to the query predicate by locating one potentially successful tableau, while universal query checking returns multiple solutions from all such potentially successful tableaux. Our query checking works with multiple placeholders and placeholders with both positive and negative occurrences. Performance comparisons show that our query-checking technique is very efficient and our model-checking runs as fast as the existing state-of-the-art model checker ALV for Presburger systems.

The efficient query checking and model checking together with the fast-error-detection capability make CWB-QC interesting for the understanding of system designs.

Directions for Future Research Apparently, there are direct extensions of the parametric model checking to Presburger systems and the temporal logic query checking to real-time systems. These extensions are the benefits of the generic model-checking framework. Techniques developed in one application domain can be shared by another. With these tools in hand, the next step would be some industry-level case studies. We are planning to apply them to the projects from aerospace and automotive industries.

Our parametric algorithm terminates for parameter constraints taking the form of finite sets on allowed values. We would investigate whether it also terminates with a more general constraint over parameters (e.g. an infinite set of parameter settings) in the future. To study the constraint synthesis problem with our forward / backward approach would also be very interesting.

PESs provides a generic model checking framework. Seeking new techniques to solve the PESs, providing optimizations to existing algorithms would be a long-run task.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In addition, advanced data structures play key roles in efficient symbolic model checking for data-based systems. To invent and experiment new data-structures would always be exciting. Particularly, we are planning to use BDD-like data structures for our next version of the real-time model checking.

To find new applications based on PESs would also be promising. For example, the relationship between vacuity checking [18, 73] and query checking has been studied by [93]. The idea is to use query checking to solve a weaker (parameterized) version of vacuity. Typically, if $M \models \psi$ and there is a stronger/weaker (depending on the polarity of the subformula in question) formula ϕ which could be used to substitute the subformula φ in ψ, and $M \models \psi[\varphi \leftarrow \phi]$, we can say that φ is relatively vacuous with respect to ϕ in this model checking problem. And the existence of a stronger/weaker formula can be detected by query checking. Therefore, our query-checking technique can help to detect vacuity for the design of data-based systems.

Data mining is the practice of automatically searching large stores of data for patterns. In recent years, attempts have been made to bridge model checking and data mining. For example, the XML path language can be encoded into CTL formalism [56]. Since temporal logic can also be thought as pattern language, it would be fruitful to systematically investigate how techniques developed in the area of model-checking could be used for data mining.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Bibliography

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addision-Wesley, 1974.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of LICS'90*. IEEE Computer Society Press, 1990.

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

[4] 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 *Proceedings of the 13th IEEE Real-Time Systems Symposium*, pages 157–166, 1992.

[5] R. Alur, K. Etessami, S. L. Torre, and D. Peled. Parametric temporal logic for "model measuring". In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP*, volume 1644 of *Lecture Notes in Computer Science*, pages 159–168. Springer, 1999.

[6] R. Alur, T. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. on Software Engineering*, 22(3):181–201, 1996.

[7] R. Alur and T. A. Henzinger. A really temporal logic. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, 1989.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[8] R. Alur and T. A. Henzinger, editors. *Computer Aided Verification (CAV '96)*, volume 1102 of *Lecture Notes in Computer Science*, New Brunswick, New Jersey, July 1996. Springer-Verlag.

[9] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *ACM Symposium on Theory of Computing*, pages 592–601, 1993.

[10] R. Alur and D. Peled, editors. *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, volume 3114 of *Lecture Notes in Computer Science*. Springer, 2004.

[11] H. R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1), 1994.

[12] H. R. Andersen. On model checking infinite-state systems. In A. Nerode and Y. Matiyasevich, editors, *LFCS*, volume 813 of *Lecture Notes in Computer Science*, pages 8–17. Springer, 1994.

[13] A. Annichini, A. Bouajjani, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In G. Berry, H. Comon, and A. Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 368–372. Springer, 2001.

[14] J. M. Atlee and M. A. Buckley. A logic-model semantics for scr software requirements. In *ISSTA*, pages 280–292, 1996.

[15] F. Balarin. Approximate reachability analysis of timed automata. In *IEEE Real-Time Systems Symposium*, pages 52–61. IEEE Computer Society, 1996.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[16] G. Bandini, R. F. L. Spelberg, R. C. H. de Rooij, and H. Toetenel. Application of parametric model checking - the root contention protocol. In *34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, 2001.

[17] C. Bartzis and T. Bultan. Efficient symbolic representations for arithmetic constraints in verification. *International Journal of Foundations of Computer Science (IJFCS)*, 14(4):605–624, Aug. 2003.

[18] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in actl formulaas. In Grumberg [60], pages 279–290.

[19] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi. Uppaal implementation secrets. In W. Damm and E.-R. Olderog, editors, *FTRTFT*, volume 2469 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2002.

[20] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In N. Halbwachs and D. Peled, editors, *CAV*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353. Springer, 1999.

[21] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

[22] S. Berezin. *Model Checking and Theorem Proving: a Unified Framework*. PhD thesis, Carnegie Mellon University, 2002.

[23] G. S. Bhat and R. Cleaveland. Efficient local model checking for fragments of the modal μ-calculus. In T. Margaria and B. Steffen, editors, *Proceedings of the Second International Workshop on*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96), Vol. 1055 of *Lecture Notes in Computer Science*, pages 107–126. Springer-Verlag, Mar. 1996.

[24] G. S. Bhat and R. Cleaveland. Efficient model checking via the equational μ-calculus. In E. M. Clarke, editor, *11th Annual Symposium on Logic in Computer Science (LICS '96)*, pages 304–312, New Brunswick, NJ, July 1996. Computer Society Press.

[25] 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.

[26] N. Bjørner, A. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H. B. Sipma, and T. E. Uribe. STeP: Deductive-algorithmic verification of reactive and real-time systems. In Alur and Henzinger [8], pages 415–418.

[27] B. Boigelot. *Symbolic Mehtods for Exploring Infinite State Spaces.* PhD thesis, Université de Liège, Belgium, 1999.

[28] A. Boudet and H. Comon. Diophantine equations, presburger arithmetic and finite automata. In H. Kirchner, editor, *CAAP*, volume 1059 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 1996.

[29] G. Bruns and P. Godefroid. Temporal logic query-checking. In *Proc. of 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 409–417, Boston, MA, USA, June 2001. IEEE Computer Society.

[30] V. Bruyère, E. Dall'Olio, and J.-F. Raskin. Durations, parametric model-checking in timed automata with presburger arithmetic. In H. Alt and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

M. Habib, editors, *STACS*, volume 2607 of *Lecture Notes in Computer Science*, pages 687–698. Springer, 2003.

[31] V. Bruyère and J.-F. Raskin. Real-time model-checking: Parameters everywhere. In P. K. Pandya and J. Radhakrishnan, editors, *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2003.

[32] T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using presburger arithmetic. In Grumberg [60], pages 400–411.

[33] 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.

[34] W. Chan. Temporal-locig queries. In E. A. Emerson and A. P. Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 450–463. Springer, 2000.

[35] J. Chang, S. Berezin, and D. L. Dill. Using interface refinement to integrate formal verification into the design cycle. In Alur and Peled [10], pages 122–134.

[36] 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, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[37] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2), 1986.

[38] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[39] R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27(8):725–747, 1989.

[40] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In G. von Bochmann and D. K. Probst, editors, *CAV*, volume 663 of *Lecture Notes in Computer Science*, pages 410–422. Springer, 1992.

[41] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A semantics-based tool for the verification of concurrent systems. *ACM TOPLAS*, 15(1), 1993.

[42] R. Cleaveland and J. Riely. Testing-based abstractions for value-passing systems. In B. Jonsson and J. Parrow, editors, *CONCUR*, volume 836 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 1994.

[43] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In K. G. Larsen and A. Skou, editors, *CAV*, volume 575 of *Lecture Notes in Computer Science*, pages 48–58. Springer, 1991.

[44] A. Collomb-Annichini and M. Sighireanu. Parameterized reachability analysis of the ieee 1394 root contention protocol using trex. In *Proceedings of Workshop on Real-Time Tools (RT-TOOLS'2001), Aalborg, Denmark*, August 2001.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[45] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In A. J. Hu and M. Y. Vardi, editors, *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 1998.

[46] G. B. Dantzig and B. C. Eaves. Fourier-motzkin eliminationand its dual. *Journal of Combinatorial Theory A*, 14:288–297, 1973.

[47] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool kronos. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1995.

[48] G. Delzanno and A. Podelski. Model checking in clp. In R. Cleaveland, editor, *TACAS*, volume 1579 of *Lecture Notes in Computer Science*, pages 223–239. Springer, 1999.

[49] G. Delzanno and A. Podelski. Constraint-based deductive model checking. *Journal of Software and Tools for Technology Transfer*, 3(3):250–270, 2001.

[50] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

[51] X. Du, C. Ramakrishnan, and S. Smolka. Tabled resolution + constraints: A recipe for model checking real-time systems. In *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS 2000)*, 2000.

[52] E. A. Emerson and J. Y. Halpern. 'Sometime' and 'not never' revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[53] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the First Annual Symposium on Logic in Computer Science*, pages 267–278, 1986.

[54] E. A. Emerson and R. J. Trefler. Parametric quantitative temporal reasoning. In *LICS 99*, pages 336–343, 1999.

[55] C. Flanagan. Automatic software model checking using clp. In P. Degano, editor, *ESOP*, volume 2618 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 2003.

[56] G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *LICS*, pages 189–202. IEEE Computer Society, 2002.

[57] J. Groote and T. Willemse. A checker for modal formulas for processes with data. Technical report, Technische Universiteit Eindhoven, The Neitherlands, 2002.

[58] J. F. Groote and M. Keinänen. Solving disjunctive/conjunctive boolean equation systems with alternating fixed points. In K. Jensen and A. Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 436–450. Springer, 2004.

[59] J. F. Groote and T. A. C. Willemse. Parameterised boolean equation systems (extended abstract). In P. Gardner and N. Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 308–324. Springer, 2004.

[60] O. Grumberg, editor. *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*. Springer, 1997.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[61] A. Gurfinkel, B. Devereux, and M. Chechik. Model exploration with temporal logic query checking. In *SIGSOFT FSE*, pages 139–148, 2002.

[62] C. Heitmeyer, R. Jeffords, and B. Lawbaw. Automated consistency checking of requirements specification. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, 1996.

[63] 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.

[64] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2), 1994.

[65] S. Hornus and P. Schnoebelen. On solving temporal logic queries. In H. Kirchner and C. Ringeissen, editors, *AMAST*, volume 2422 of *Lecture Notes in Computer Science*, pages 163–177. Springer, 2002.

[66] T. Hune, J. Romijn, M. Stoelinga, and F. W. Vaandrager. Linear parametric model checking of timed automata. In Margaria and Yi [80], pages 189–203.

[67] J. Jaffar and J. L. Lassez. Constraint logic programming. In *Proceedings of the 29th Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[68] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[69] J.Bradfield and C.Stirling. Local model checking for infinite state spaces. *Theoretical Computer Science*, 96:157–174, 1992.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[70] J.R.Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Cogress on Logic, Methodology and Philosophy of Science*, pages 1–11. Standford University Press, 1960.

[71] N. Klarlund and A. Moller. *MONA Version 1.4 User Manual.* BRICS Notes Series NS-01-1, Department of Computer Science, University of Aarhus, Jan. 2001.

[72] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, Dec. 1983.

[73] O. Kupferman and M. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2):224–233, Feb. 2003.

[74] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL*, pages 97–107, 1985.

[75] H. Lin. Symbolic transition graph with assignment. In U. Montanari and V. Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 1996.

[76] X. Liu, C. R. Ramakrishnan, and S. A. Smolka. Fully local and efficient evaluation of alternating fixed points (extended abstract). In B. Steffen, editor, *TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 5–19. Springer, 1998.

[77] D. E. Long, A. Browne, E. M. Clarke, S. Jha, and W. R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In D. Dill, editor, *Proceedings of the Sixth International Conference on Computer Aided Verification (CAV '94)*, Vol. 818 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[78] A. Mader. *Verification of Modal Properties Using Boolean Equation Systems.* PhD thesis, Müchen, Techn-Univ., 1997.

[79] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, 1995.

[80] T. Margaria and W. Yi, editors. *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, volume 2031 of *Lecture Notes in Computer Science.* Springer, 2001.

[81] R. Mateescu. Local model-checking of an alternation-free value-based modal mu-calculus. In *Proceedings of the 2nd International Workshop on Verification, Model Checking and Abstract Interpretation*, Sept. 1998.

[82] R. Mateescu. A generic on-the-fly solver for alternation-free boolean equation systems. In H. Garavel and J. Hatcliff, editors, *TACAS*, volume 2619 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2003.

[83] J. B. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Difference decision diagrams. In J. Flum and M. Rodríguez-Artalejo, editors, *CSL*, volume 1683 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 1999.

[84] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In Alur and Henzinger [8], pages 411–414.

[85] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In W. Brauer, editor, *ICALP*, volume 194 of *Lecture Notes in Computer Science*, pages 15–32. Springer, 1985.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[86] W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:102–104, 1992.

[87] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proceedings of the International Symposium in Programming*, volume 137 of *Lecture Notes in Computer Science*, Berlin, 1982. Springer-Verlag.

[88] Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In Grumberg [60], pages 143–154.

[89] C. R. Ramakrishnan. A model checker for value-passing mu-calculus using logic programming. In I. V. Ramakrishnan, editor, *PADL*, volume 1990 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2001.

[90] J. Rathke. *Symbolic techniques for value-passing Calculi.* PhD thesis, University of Sussex, 1997.

[91] J. Rathke and M. Hennessy. Local model checking for value-passing processes (extended abstract). In M. Abadi and T. Ito, editors, *TACS*, volume 1281 of *Lecture Notes in Computer Science*, pages 250–266. Springer, 1997.

[92] M. Samer and H. Veith. Validity of ctl queries revisited. In M. Baaz and J. A. Makowsky, editors, *CSL*, volume 2803 of *Lecture Notes in Computer Science*, pages 470–483. Springer, 2003.

[93] M. Samer and H. Veith. Parameterized vacuity. In A. J. Hu and A. K. Martin, editors, *FMCAD*, volume 3312 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2004.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[94] J. Sifakis. A unified approach for studying the properties of transition systems. *TCS*, 18, 1982.

[95] H. B. Sipma, T. E. Uribe, and Z. Manna. Deductive model checking. In Alur and Henzinger [8], pages 208–219.

[96] O. Sokolsky and S. A. Smolka. Local model checking for real-time systems (extended abstract). In P. Wolper, editor, *CAV*, volume 939 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 1995.

[97] R. F. L. Spelberg and W. J. Toetenel. Parametric real-time model checking using splitting trees. *Nordic Journal of Computing*, 8(1):88–120, 2001.

[98] A. Szalas. Logic for computer science. lecture notes. URL http://www.ida.liu.se/~andsz.

[99] A. Szałas. On natural deduction in first-order fixpoint logics. *Fundamenta Informaticae*, 26:81–94, 1996.

[100] L. Tan. *Evidence-Based Verification*. PhD thesis, State University of New York at Stony Brook, 2002.

[101] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pac. J. Math.*, 1955.

[102] B. Vergauwen and J. Lewi. Efficient local correctness checking for single and alternating boolean equation systems. In S. Abiteboul and E. Shamir, editors, *ICALP*, volume 820 of *Lecture Notes in Computer Science*, pages 304–315. Springer, 1994.

[103] F. Wang. Parametric timing analysis for real-time systems. *Information and Computation*, 130:131–150, 1996.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[104] F. Wang. Parametric analysis of computer systems. *Formal Methods in System Design*, 17(1):39–60, 2000.

[105] F. Wang. Efficient verification of timed automata with bdd-like data-structures. In L. D. Zuck, P. C. Attie, A. Cortesi, and S. Mukhopadhyay, editors, *VMCAI*, volume 2575 of *Lecture Notes in Computer Science*, pages 189–205. Springer, 2003.

[106] F. Wang. Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures. In Alur and Peled [10], pages 295–307.

[107] F. Wang and H.-C. Yen. Parametric optimization of open real-time systems. In P. Cousot, editor, *SAS*, volume 2126 of *Lecture Notes in Computer Science*, pages 299–318. Springer, 2001.

[108] T. Yavuz-Kahveci, M. Tuncer, and T. Bultan. A library for composite symbolic representations. In Margaria and Yi [80], pages 52–66.

[109] S. Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1:123–133, 1997.

[110] D. Zhang and R. Cleaveland. Efficient temopral logic query checking for presburger systems. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, California, USA*. IEEE Computer Society, 2005.

[111] D. Zhang and R. Cleaveland. Fast generic model-checking for data-based systems. In F. Wang, editor, *Proceedings of the 25th International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2005)*, volume 3731 of *Lecture Notes in Computer Science*, Taiwan, Oct. 2005. Springer-Verlag.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[112] D. Zhang and R. Cleaveland. Fast parametric real-time model checking. In *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS 2005), December 5-8, 2005 Miami, Florida, USA*. IEEE Computer Society, 2005.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Index

ALV, 9, 99
assignment, 17

BDD, 95
BDD-like data structure, 60
BES, 2, 38
boolean equation systems, 2, 14, 38
bound variables, 12

calibration parameters, 4
case study
- bakery, 101
- barber, 102
- CSMA/CD, 67, 73
- FDDI, 66
- GRC, 73
- LBOUND, 67
- LEADER, 67
- MUX, 66, 73
- PATHOS, 66
- REACTOR, 73
- thermostat, 95
- ticket, 101

clock region, 47
clock zones, 47
CLP, 3
complete lattice, 10, 20
concrete actions, 28
Concurrency Workbench, 8
constraint solver, 3
constraint synthesis, 5
CRD, 60
CSL, 99
CTL, 1, 2, 5, 7, 31, 81, 105
CTL*, 1
CWB-QC, 8, 92
CWB-RT, 56

data expressions, 16
data predicates, 16
data states, 17
DBM, 56
discrete-time model checking, 22

environment, 11, 29, 43, 81

finite-state model checking, 38
first-order boolean equation systems, 2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

first-order modal mu-calculus, 29
fixpoint, 10
fixpoint approximation, 11, 21
fixpoint equation systems, 12
  parity block, 12
  semantics, 12
  syntax, 12
Floyd-Warshall algorithm, 57, 60
Fourier-Motzkin algorithm, 92
free variables, 12, 16, 18

Gentzen-like proof system, 23, 49, 83

Horn clause, 3
HRD, 61
HyTech, 72

internal action, 29

Kronos, 65

linear terms, 41, 78
local model checking, 3
logic programming, 3
LPMC, 72
LTL, 1

MES, 29, 31, 43, 81
modal equation systems, 29
  formula variables, 29, 43, 81
  formula-closed, 29
  semantics, 29, 30
  syntax, 29
modal mu-calculus, 1, 29, 33
MONA, 95

Omega Library, 60, 92

parameter valuation, 41
parameterized boolean equation systems, 39
parametric real-time model checking, 4, 40
parity indicator, 12, 19
partial order, 10
PDBM, 56
PES, 2, 19, 20
placeholder, 7, 86
polyhedra, 92
positive normal form, 31
predicate equation systems, 19
  basic data theory, 16
  predicate block, 20
  predicate calculus, 18
  predicate equation block, 19
  predicate state, 18
  predicate variables, 18
  predicate-closed, 18
Presburger formulas, 79
Presburger modal mu-calculus
  semantics, 81

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

syntax, 81
projection operator, 87
PST, 88, 91

real-time modal mu-calculus
semantics, 43
syntax, 43
RED, 65, 72

SCR, 95
sequent, 23
skolemization, 26
state predicates, 41
state-transformation formula, 17, 30
strongest postcondition, 18, 48
substitution operation, 16, 18
successful leaf, 23
symbolic action, 32

TCTL, 44
temporal-logic query checking, 7, 86
existential query checking, 88
universal query checking, 91
time predecessor, 48
time successor, 48
transition systems
concrete transition systems, 28
CTS, 29, 33, 42, 43, 81
event-action language, 32
Linear Process Equations, 32
parametric timed automata, 41
Presburger systems, 7, 79
PS, 79, 86
STG, 32, 33
STGA, 32
symbolic transition graphs, 28, 32
timed automata, 32
value-passing CCS, 32
translation function, 35, 46, 82
TReX, 72

universal parametric real-time model-checking, 5
UPPAAL, 65

vacuity checking, 105

weakest precondition, 30, 48
WS1S, 95

XML, 105

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.