Predicate Diagrams for the Verification of Real-Time Systems

Eunyoung Kang, Stephan Merz

Noname manuscript No.
(will be inserted by the editor)

Predicate Diagrams for the Verification of Real-Time Systems

Eun-Young Kang and Stephan Merz

INRIA Lorraine & LORIA
Nancy, France
{Eun-Young.Kang,Stephan.Merz}@loria.fr

The date of receipt and acceptance will be inserted by the editor

Abstract This article discusses a new format of predicate diagrams for the verification of real-time systems. We consider systems that are defined as extended timed graphs, a format that combines timed automata and constructs for modelling data, possibly over infinite domains. Predicate diagrams are succinct and intuitive representations of Boolean abstractions. They also represent an interface between deductive tools used to establish the correctness of an abstraction, and model checking tools that can verify behavioral properties of finite-state models. The contribution of this article is to extend the format of predicate diagrams to timed systems. We establish a set of verification conditions that are sufficient to prove that a given predicate diagram is a correct abstraction of an extended timed graph; these verification conditions can often be discharged with SMT solvers such as CVC-lite. Additionally, we describe how this approach extends naturally to the verification of parameterized systems. The formalism is supported by a toolkit, and we demonstrate its use at the hand of Fischer's real-time mutual-exclusion protocol.

1 Introduction

Model checking has become a routine technique for the verification of hardware systems and communication protocols, which can essentially be modeled as finite-state systems. Seminal work by Alur and Dill, Henzinger, and others [4, 16] has shown that model checking techniques can also be developed for real-time systems, and implementations of such tools have made significant progress and can handle non-trivial systems [7, 28]. Model checking is attractive because it is fully automatic, but also because it provides counter-examples when the property of interest (desirable behaviour) does not hold of the system.

However, real-time model checking is applicable only under certain restrictions; most notably, it requires the system to be represented as a timed automaton whose discrete state space (disregarding the real-valued clocks) is finite. This restriction is in general not satisfied for software systems, and ad-hoc approximations are therefore used in model checking. On the other hand, deductive techniques can in principle be used to verify infinite-state systems, based on sets of axioms and inference rules. Although these can be supported by theorem provers and interactive proof assistants, their use requires considerable expertise and tedious user interaction [5].

Algorithmic and deductive verification techniques are therefore complementary, and combinations of the two approaches should give rise to powerful verification environments. For example, a theorem prover can be used to verify that a finite-state model is a correct abstraction of a given system, and properties of that finite-state abstraction can then be established using model checking. In order to make this idea more concrete, we need to identify a suitable format that serves as an interface between deductive and algorithmic techniques and that gives rise to feasible verification conditions.

Predicate abstraction [14, 20] has emerged as a fruitful basis for software verification. It underlies tools such as SLAM [9] and BLAST [17], which moreover contain algorithms for abstraction refinement when the model checker reports a counter-example for the abstracted model that cannot be reproduced over the original model.

In previous work [11], we have proposed a format of presenting predicate abstractions, called predicate diagrams, with an emphasis on proving liveness properties of discrete systems. In this article we propose a variant PDT of predicate diagrams, intended for the verification of real-time systems. We also show how to relate PDTs to real-time systems described as extended timed-automata graphs (XTGs), a formalism that combines timed automata and constraints for modeling infinite data domains [22, 6]. Basically, a PDT shows a finite-state abstraction of an XTG, and the correctness of the abstraction can be established by proving a number of verification conditions expressed in first-order logic. On the other hand, model checking is used to establish correctness properties (expressed in temporal logic) over the PDT.

Section 2 presents a general relation between a real system and its abstract model. It also describes an informal overview of how we apply abstraction and refinement techniques to construct a correct abstract model. Section 3 presents XTGs as models of real-time systems. Section 4 introduces PDTs, defines the notion of conformance to relate XTGs and PDTs, and establishes a set of sufficient proof obligations to verify conformance. To illustrate the approach, we present a verification of Fischer's mutual-exclusion protocol between two processes in Section 5. Section 6 gives an overview of how we extended this approach to parametric verification. It also describes a case study in applying our idea to Fischer's mutual-exclusion protocol among n processes. Section 7 discusses future work and concludes the article.

2 Abstraction and Refinement Techniques

Predicate abstraction has been found to be a powerful tool for software verification, and we transfer this idea to the domain of real-time systems. The basic assumption underlying predicate abstraction is that for the verification of a given property, the state space of a real system can be partitioned into finitely many equivalence classes. For example, the precise amount of time elapsed in a transition does not really matter as long as the clock values are within certain bounds and similarly, the precise values of the data can be abstracted with the help of predicates that indicate characteristic properties.

In general, the relationship between real and abstract models that underlies abstract interpretation is described by a Galois connection. The abstract domain is a Boolean lattice whose atoms are the set of predicates true or false of a set of states in a concrete model. The model obtained by abstraction w.r.t. this lattice is an over-approximation: it includes all runs of the concrete system, but may also exhibit some behaviors that have no counterpart in the concrete system.

We now introduce our methodology by adding an abstraction and refinement framework. The idea is to reduce the number of states of a model by abstracting away behaviors that are not essential to the verification. The generic techniques are known as incremental abstraction refinement [2] and counterexample-guided abstraction refinement [12].

These approaches have inspired our IRA (Iterative-Abstract-Refinement algorithm) in [19], for constructing PDTs/abstractions of real-time systems: At first, an abstract model can be constructed from a real model manually based on a set of predicates, which is the subset of a real-model's *configurations* (Note that the set of configurations is all information on locations and their constrains in a real model).

The abstract-model is the Boolean abstraction and its atoms are the set of predicates which make a set of states in a real-model true or false. If the abstract model gurantees that it captures all runs of a real model, the system is successfully abstracted (called a *complete* model). If the abstract model does not, we refine the abstract model until it becomes *complete* by checking a set of verification conditions which gurantee that a given abstract model captures all runs of the real model.

3 Extended Timed Automata Graphs

We model real-time systems as XTGs (extended timed automata graphs) [6, 25] a notation that combines the familiar framework of timed automata [3], synchronous value passing between parallel processes, and a language for modeling data. The semantics of XTGs is defined in terms of *timed structures*, also known as timed transition systems.

Definition 1 *A* timed structure *is a tuple* $\langle S, S_0, T \rangle$ *where*

- *S is a set of states,*
- *$S_0 \subseteq S$ is the subset of initial states, and*
- *$T \subseteq S \times (\mathbb{R}^{\geq 0} \cup \{\mu\}) \times S$ is a transition relation.*

A run *of a timed structure is an infinite sequence*

$$\pi = s_0 \xrightarrow{\lambda_0} s_1 \xrightarrow{\lambda_1} s_2 \ldots$$

where $s_0 \in S_0$ is an initial state and $\langle s_i, \lambda_i, s_{i+1} \rangle \in T$ is a transition for all $i \in \mathbb{N}$.

Timed structures distinguish two kinds of transitions: time-passing transitions are labeled by a non-negative real number that represents the amount of time that has elapsed during this transition. Discrete transitions model state changes and have a special label μ.

Our definition of XTGs is parameterized by an underlying language for modeling data. In this paper, we do not need to fix a precise signature, but assume the following generic syntactic framework:

Definition 2 *A* data language *provides the following syntactic domains:*

- *V : a finite set of variables,*
- *$V_c \subseteq V$: a subset of clock variables,*
- *Expr: value expressions (over the set V of variables), and*
- *Bexpr $\subseteq$ Expr: the subset of Boolean expressions.*

Similarly, we do not fix a precise semantics, but simply require the existence of a suitable semantic domain and evaluation function.

Definition 3 *We assume a universe Val of values that includes the set $\mathbb{R}^{\geq 0}$ of non-negative real numbers and the Boolean values tt and ff. A valuation is a mapping $\rho : V \rightarrow Val$ from variables to values such that $\rho(c) \in \mathbb{R}^{\geq 0}$ for all $c \in V_c$. For a valuation ρ and $\delta \in \mathbb{R}^{\geq 0}$ we write $\rho[+\delta]$ to denote the environment that increases each clock in V_c by δ:*

$$\rho[+\delta](v) = \begin{cases} \rho(v) + \delta & \text{if } v \in V_c \\ \rho(v) & \text{otherwise} \end{cases}$$

We assume given an evaluation function

$$[\![\_]\!]_{\_} : Expr \rightarrow (V \rightarrow Val) \rightarrow Val$$

that associates a value $[\![e]\!]_\rho$ with any expression $e \in Expr$ and valuation ρ. We require that $[\![e]\!]_\rho \in \{tt, ff\}$ for all $e \in Bexpr$.

An XTG consists of a fixed, finite number of processes. The control part of any process is described as a finite state machine. The full state space is given by a set of variables (which can be local to the process or shared between processes), communication channels, and clocks. As in timed automata, clocks are continuous variables that all increase at a fixed, uniform rate. Clock values can be tested in transition guards, and clocks can be reset during transitions. Moreover, locations of a process are associated with invariants. These are particularly useful to ensure upper bounds on clocks, limiting the amount of time that a location can remain active.

Finally, transitions of an XTG process can be marked as urgent, implying that they should be taken as soon as they are enabled. Processes of an XTG are executing asynchronously in parallel. They communicate by means of shared variables or by synchronous value passing in the spirit of value-passing CCS [24].

In the present paper we restrict ourselves to shared variables and for simplicity do not consider value passing. Thus Definition 4 presents a subset of full XTG.

Definition 4 *An XTG process is a tuple* $\langle Init, L, l_0, I, E, U\rangle$ *where*

- $Init \in Bexpr$ *indicates the initial condition for (the data part of) the process,*
- L *is a finite set of locations,*
- $l_0 \in L$ *is the initial location,*
- $I : L \to Bexpr$ *assigns an invariant to each location,*
- $E \subseteq L \times Bexpr \times 2^{V \times Expr} \times L$ *is a set of edges, represented as tuples* $\langle l, g, u, l'\rangle$ *where*
  - $l \in L$ *is the source location,*
  - $g \in Bexpr$ *is a boolean expression, the guard,*
  - $u \subseteq V \times Expr$ *is an update, i.e. a set of assignments, and*
  - $l' \in L$ *is the destination location.*

  Note that an assignment is defined as a set of pairs $\langle v, e\rangle$ *where* v *is a variable and* e *is an expression whose value is to be assigned to the variable. Each variable should appear at most once in the update set.*
- $U \subseteq E$ *identifies the subset of urgent edges.*

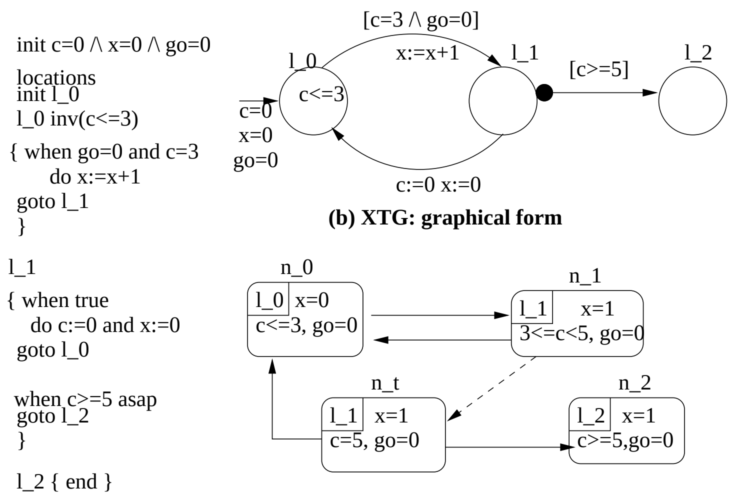
An XTG is a finite set of XTG processes.

Fig. 1 shows a sample XTG consisting of a single process, both in its textual (Fig. 1.a) and graphical (Fig. 1.b) representations. The XTG process consists of three locations l_0, l_1, and l_2. The edge from l_1 to l_2 is urgent, as indicated by the keyword **asap** in Fig. 1.a and by the black dot at the source of the transition in Fig. 1.b.

With any XTG we associate a timed structure whose states are given by the active locations of the XTG and the valuations of the underlying variables.

Definition 5 *Let* $\mathcal{X}$ *be an XTG with processes* $P_1, \ldots, P_n$*. The timed structure* $\mathcal{T} = \langle S, S_0, T\rangle$ *generated by* $\mathcal{X}$ *is the smallest structure such that*

- S_0 *consists of all tuples* $\langle l_{1,0}, \ldots, l_{n,0}, \rho\rangle$ *where* $l_{i,0}$ *is the initial location of process* P_i *and* $[\![Init_i]\!]_\rho = tt$ *for the initial conditions* $Init_i$ *of all processes* P_i*.*



```
init c=0 /\ x=0 /\ go=0

locations
init l_0
l_0 inv(c<=3)
{ when go=0 and c=3
      do x:=x+1
  goto l_1
}

l_1
{ when true
    do c:=0 and x:=0
  goto l_0

  when c>=5 asap
  goto l_2
}

l_2 { end }
```

(a) XTG: text form **(b) XTG: graphical form** **(c) A PDT for this XTG**

Fig. 1 Example XTG and PDT.

– *For any state* $s = \langle l_1, \ldots, l_n, \rho \rangle \in S$*, any* $i \in \{1, \ldots, n\}$*, and any edge* $\langle l_i, g, u, l'_i \rangle \in E_i$ *of process* P_i *such that* $[\![g]\!]_\rho = tt$*,* $\mathcal{T}$ *contains a transition* $\langle s, \mu, s' \rangle \in T$ *where* $s' = \langle l'_1, \ldots, l'_n, \rho' \rangle$ *and* $l'_j = l_j$ *for* $j \neq i$*, and where*

$$\rho'(v) = \begin{cases} [\![e]\!]_\rho & \text{if } \langle v, e \rangle \in u \\ \rho(v) & \text{otherwise} \end{cases}$$

provided that $[\![I(l'_j)]\!]_{\rho'} = tt$ *for all* $j \in \{1, \ldots, n\}$*.*

– *For a state* $s = \langle l_1, \ldots, l_n, \rho \rangle \in S$ *and* $\delta \in \mathbb{R}^{\geq 0}$*,* $\mathcal{T}$ *contains a transition* $\langle s, \delta, s' \rangle \in T$ *where* $s' = \langle l_1, \ldots, l_n, \rho[+\delta] \rangle$ *provided that for all* $0 \leq \varepsilon \leq \delta$*, the location invariants evaluate to true, i.e.* $[\![I(l_i)]\!]_{\rho[+\varepsilon]} = tt$*, and that for all* $0 \leq \varepsilon < \delta$*, the guards of any urgent edge* $\langle l_i, g, u, l'_i \rangle$ *leaving an active location* l_i *of state* s *evaluate to false, i.e.* $[\![g]\!]_{\rho[+\varepsilon]} = ff$*.*

Discrete transitions correspond to edges of one of the XTG processes. They require the guard of the edge to evaluate to true in the source state. The destination state is obtained by activating the target location of the edge and by applying the updates associated with the edge. Time-passing transitions uniformly update all clock variables; time is not allowed to elapse beyond any value that activates some urgent edge of an XTG process. In either case, the invariants of all active locations have to be maintained.

4 Predicate Diagrams for Timed Systems

Due to their rich data model, standard real-time model checking techniques do not apply to XTGs. We now introduce the PDT notation that we use to represent predicate abstractions of XTGs. The verification problem then reduces to (a) establishing the correctness of the abstraction and (b) verifying the desired property over the abstract model. Because our abstractions give rise to finite-state models, the second subproblem is amenable to model checking. Subproblem (a) can be addressed using theorem proving, and we identify a set of sufficient, non-temporal verification conditions in Section 4.2.

4.1 The PDT Notation

The formal definition of PDTs is given with respect to a set L that represents locations (or, more precisely, location tuples) of the underlying XTG, as well as with respect to a set $\mathcal{P}$ of predicates (i.e., Boolean expressions) of interest. We write $\overline{\mathcal{P}}$ to denote the set containing the predicates in $\mathcal{P}$ and their negations.

Definition 6 *Assume given finite sets L and $\mathcal{P}$. A* PDT *(over L and $\mathcal{P}$) is given by a tuple $\langle N, N_0, R_\mu, R_\tau \rangle$ as follows:*

- *$N \subseteq L \times 2^{\overline{\mathcal{P}}}$ is a finite set of nodes of the PDT; each node is a pair $\langle l, P \rangle$ for $l \in L$ and $P \subseteq \overline{\mathcal{P}}$,*
- *$N_0 \subseteq N$ is the set of initial nodes,*
- *$R_\mu, R_\tau \subseteq N \times N$ are two relations that represent discrete and time-passing transitions of the PDT. We require that R_τ be reflexive. We usually write $n \rightarrow_\mu n'$ and $n \rightarrow_\tau n'$ for $(n, n') \in R_\mu$ and $(n, n') \in R_\tau$.*

A run *of a PDT is an infinite sequence*

$$\sigma \quad = \quad n_0 \xrightarrow{lab_0} n_1 \xrightarrow{lab_1} n_2 \ldots$$

where $n_0 \in N_0$, $lab_i \in \{\mu, \tau\}$, and $n_i \rightarrow_{lab_i} n_{i+1}$ for all $i \in \mathbb{N}$.

Thus, a PDT is a labelled transition system with two transition relations. A PDT node represents a set of XTG states by indicating the active locations and certain predicates satisfied by these states. The transition relations correspond to discrete transitions and time-passing transitions of the XTG. When drawing a PDT, as in Fig. 1.c, we use solid arrows for edges in R_μ and dashed arrows for edges in R_τ. Every node has a τ-loop associated with it, which we do not show explicitly.

4.2 Conformance: Relating XTGs and PDTs

We now formally define what it means for a PDT to conform to an XTG, i.e. when the PDT is a correct abstraction of the XTG. We also establish a set of verification conditions that guarantee conformance. Our purpose in defining conformance is to ensure that any property verified over the PDT also holds for the XTG. Because we are interested in verifying linear-time properties, and such properties hold of a system if they are satisfied by each system run, we should verify that each run of an XTG can be mapped to a run of the PDT. The following definition makes this intuition precise.

Definition 7 *Given an XTG $\mathcal{X}$, a PDT Δ, and a run $\pi = \langle l_0, \rho_0\rangle \xrightarrow{\lambda_0} \langle l_1, \rho_1\rangle \ldots$ of $\mathcal{X}$, we say that a run $\sigma = n_0 \xrightarrow{lab_0} n_1 \ldots$ of Δ is a* trace *of π iff*

- *π and σ are of equal length (in particular, both infinite),*
- *$n_i = \langle l_i, P_i\rangle$ for some $P_i \subseteq \overline{\mathcal{P}}$ such that $[\![p]\!]_{\rho_i} = tt$ for all $p \in P_i$ and all i, i.e. the states of π and the nodes of σ activate the same locations and all predicates of n_i are satisfied in the corresponding state of π, and*
- *$lab_i = \mu$ if $\lambda_i = \mu$, and $lab_i = \tau$ if $\lambda_i \in \mathbb{R}^{\geq 0}$, i.e. the two runs agree on which transitions are discrete and which are time-passing.*

We say that Δ conforms to *$\mathcal{X}$ if every run of $\mathcal{X}$ has a trace in Δ.*

The definition of conformance requires to inspect all runs of an XTG. For practical purposes, we are interested in establishing a reasonably small set of first-order verification conditions that are sufficient to ensure conformance. The following theorem gives such conditions. Intuitively, we verify that every possible initial state of the XTG is represented by some initial node of PDT. Inductively, given any XTG state s corresponding to some PDT node n and any transition from s to some successor XTG state s', that transition can be mapped to a transition from nodel n in the PDT. In formulating the verification conditions, we introduce two copies V' and V'' of the set of variables V whose elements are decorated with single and double primes (v' and v'' for each $v \in V$). When P is a set of predicates, we sometimes also denote by P the conjunction of the predicates in P, and we write P' or P'' to denote the formula obtained by replacing each variable $v \in V$ by its copy v' or v''.

Theorem 1 *Let $\mathcal{X}$ be an XTG that consists of m processes $P_i = \langle Init_i, L_i, l_{0,i}, I_i, E_i, U_i\rangle$, and that $\Delta = \langle N, N_0, R_\mu, R_\tau\rangle$ is a PDT over $L_1 \times \cdots \times L_m$ and a set $\mathcal{P}$ of predicates. If all of the following conditions hold then Δ conforms to $\mathcal{X}$:*

1. $$\bigwedge_{j=1}^{m} Init_j \wedge I(l_{0,j}) \quad \Rightarrow \quad \bigvee_{\langle l_{0,1},\ldots,l_{0,m},P\rangle \in N_0} P$$
   In words, the conjunction of the initial conditions of $\mathcal{X}$ and the invariants of the initial locations imply that the predicates of one of the initial nodes of Δ marked with the initial locations must be true.
2. *For any node $n = \langle l_1, \ldots, l_m, P\rangle$ of Δ and any edge $\langle l_i, g, u, l'_i\rangle$ of XTG process P_i, let V_u denote the set of variables v that are updated by u (i.e. such that $\langle v, e\rangle \in u$ for some e), and let N' denote the set of all nodes $n' = \langle l'_1, \ldots, l'_m, Q\rangle$ where $l'_j = l_j$ for $j \neq i$ such that $n \rightarrow_\mu n'$.*
   $$P \wedge g \wedge \bigwedge_{j=1}^{m} (I(l_j) \wedge I'(l'_j)) \wedge \bigwedge_{\langle v,e\rangle \in u} v' = e \wedge \bigwedge_{v \in V \setminus V_u} v' = v \quad \Rightarrow \bigvee_{\langle l'_1,\ldots,l'_m,Q\rangle \in N'} Q'$$

In words, the predicate label of node n and the invariants of all active locations before and after the transition of $\mathcal{X}$ should imply the predicate label of some node in N'.

3. *For any node $n = \langle l_1, \ldots, l_m, P\rangle$ of Δ, let N'' denote the set of all nodes $n'' = \langle l_1, \ldots, l_m, Q\rangle$ that agree with n on the location components such that $n \rightarrow_\tau n''$.*

$$\begin{array}{l}
P \wedge \delta \in \mathbb{R}^{\geq 0} \wedge \bigwedge\limits_{c \in V_c} c' = c + \delta \wedge \bigwedge\limits_{v \in V \setminus V_c} v' = v \wedge \bigwedge\limits_{j=1}^{m} I(l_j) \wedge I'(l_j) \\
\wedge\ \forall \varepsilon \leq \delta : \bigwedge\limits_{c \in V_c} c'' = c + \varepsilon \wedge \bigwedge\limits_{v \in V \setminus V_c} v'' = v \ \Rightarrow \bigwedge\limits_{j=1}^{m} I''(l_j) \\
\wedge\ \forall \varepsilon < \delta : \bigwedge\limits_{c \in V_c} c'' = c + \varepsilon \wedge \bigwedge\limits_{v \in V \setminus V_c} v'' = v \ \Rightarrow \bigwedge\limits_{j=1}^{m} \bigwedge\limits_{\langle l_j, g, u, l_j' \rangle \in U_j} \neg g'' \\
\Rightarrow \bigvee\limits_{\langle l_1, \ldots, l_m, Q\rangle \in N''} Q'
\end{array}$$

In words, assuming the predicate label of n and the invariants of all active locations before and after a time passing transition by amount δ that does not activate any urgent transition of $\mathcal{X}$, the PDT must contain some node n'' that is reachable from n by a τ-transition and whose predicate label is guaranteed to hold.

Proof Given a run $\pi = \langle l_0, \rho_0\rangle \xrightarrow{\lambda_0} \langle l_1, \rho_1\rangle \ldots$ of $\mathcal{X}$, we can inductively construct a trace σ of π in PDT Δ as follows: because ρ_0 must satisfy the initial conditions of all processes as well as the invariants of the initial locations, condition (i) ensures that there exists some initial node of Δ that is associated with the tuple of initial locations of $\mathcal{X}$ and whose predicate label is true in ρ_0. Inductively, assume that a node $n = \langle l, P\rangle$ corresponding to the XTG configuration $s_i = \langle l_i, \rho_i\rangle$ has already been identified. If the transition in π from s_i is a discrete transition, it is due to some edge of some process P_j (cf. Def. 5), and therefore the guard of that edge must be true in ρ_i and its updates will be performed during the transition to state $\langle l_{i+1}, \rho_{i+1}\rangle$. Moreover, the location invariants must be true in the states before and after the transition. According to condition (ii) we can therefore find a node n' associated with l_{i+1} such that $n \rightarrow_\mu n'$ and that the predicate label of n' holds in ρ_{i+1}. Similarly, a time-passing transition from configuration s_i can be matched according to condition (iii). □

For example, theorem 1 can be used to show that the PDT in Fig. 1.c conforms to the XTG of Fig. 1.b. For the initial condition, we obtain the proof obligation

$$c = 0 \wedge x = 0 \wedge go = 0 \wedge c \leq 3 \quad \Rightarrow \quad c \leq 3 \wedge x = 0 \wedge go = 0$$

As an example for the verification conditions of type (ii), we consider the XTG transition from l_0 to l_1, which has to be matched with the transitions leaving node n_0 of the PDT:

$$x = 0 \land c \leq 3 \land go = 0 \land c = 3 \land go = 0 \land x' = x + 1 \land go' = go \land c' = c$$
$$\Rightarrow x' = 1 \land 3 \leq c' \land c' < 5 \land go' = 0$$

Finally, we consider the possible time passing transitions leaving location l_1, focussing on the PDT node n_1:

$$\begin{array}{l} x = 1 \land 3 \leq c \land c < 5 \land go = 0 \land \delta \in \mathbb{R}^{\geq 0} \land c' = c + \delta \land x' = x \\ \land go' = go \land \ \forall \varepsilon < \delta : c'' = c + \varepsilon \land x'' = x \land go'' = go \Rightarrow \neg(c'' \geq 5) \\ \Rightarrow (x' = 1 \land 3 \leq c' \land c' < 5 \land go' = 0) \lor (x' = 1 \land c' = 5 \land go' = 0) \end{array}$$

Observe in particular that time cannot advance beyond a clock value of 5 because the transition from l_1 to l_2 is marked as urgent.

4.3 Verification

We now turn to establishing behavioral properties of an XTG from a conformant PDT. We assume that the properties of interest are expressed in linear-time temporal logic LTL, and that they are built from the predicates in $\mathcal{P}$. We can thus simply consider the predicates that appear as labels of the PDT as uninterpreted atomic propositions. We add atomic predicates of the form $\mathbf{at}\,l$ to identify the control locations of XTG processes.

Any PDT $\varDelta$ is a finite-state transition system and can be encoded in the modeling language of conventional finite-state model checkers, following the approach described in [11]. For our experiments, we use the Spin model checker via the DIXIT tool for manipulating and analyzing predicate diagrams [13]. The definition of conformance implies that any LTL property φ built from predicates in $\mathcal{P}$ that holds over some PDT $\varDelta$ also holds of the XTG $\mathcal{X}$ provided that $\varDelta$ conforms to $\mathcal{X}$. Indeed, let π be any run of $\mathcal{X}$. Because $\varDelta$ conforms to $\mathcal{X}$, we can find a trace σ of π in $\varDelta$. Since φ is assumed to hold of $\varDelta$, it follows that σ satisfies φ, and given that only predicates in $\mathcal{P}$ appear in φ, a straightforward induction on LTL formulas shows that φ must also hold of π.

On the other hand, counter-examples produced by the model checker need not correspond to actual system runs because some detail may have been lost in the abstraction. For example, the PDT of Fig. 1.c appears to contain a discrete transition from a state represented by the node n_t to a state corresponding to node n_2 where $c = 7$, which is impossible for the XTG of Fig. 1.b.

In this paper, we do not use such counterexample-guided abstraction refinement (CEGAR) part. Because during the conformance checking, we use the property interest (desirable behaviors) as predicates in the sense that the constructed abstract model (resulted PDT) captures the run of desirable behavior of XTG. Thus, we can know that applying model checking on the result PDT doesn't give false-negative since the PDT is constructed by making the propertu we like to veryfy true.

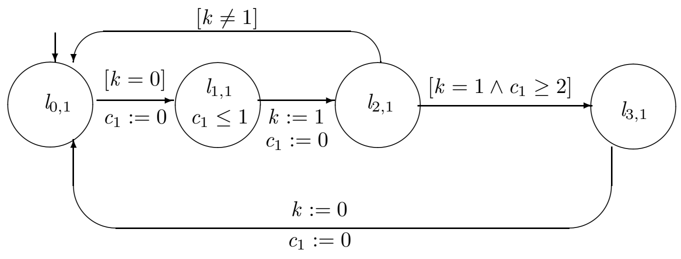


Fig. 2 An XTG for Fischer's protocol (process 1).

However, we believe that mayor improvements could be made here by applying/adding CEGAR on the refinement step resulting in a much higher degree of refinement technique.

5 An example: Fischer's protocol

We illustrate the use of PDTs using Fischer's well-known real-time protocol for ensuring mutual exclusion between two processes [8, 21]. Figure 2 shows the structure of process 1 (the other process is symmetrical): k is a shared variable accessed by both processes, whereas c_1 is a local clock of the process.

Intuitively, the protocol behaves as follows: in the first phase each process tries to register its process identification in the shared variable k. In the second phase each process tests whether its identity is still registered in k after a predefined lapse of time and then enters the critical section. The purpose of the protocol is to ensure that there is never more than one process in the critical section, expressed by the LTL formula $\Box\neg(\mathbf{at}\, l_{3,1} \wedge \mathbf{at}\, l_{3,2})$.

Figure 3 gives a PDT for Fischer's protocol, which can be shown to conform to the XTG by discharging the conditions of Theorem 1. As an example, we consider the possible transitions of process 1 from the node marked (*) in the PDT of Fig. 3 with corresponding control locations $l_{2,1}$ and $l_{3,2}$. There are two possible transitions: the first leads from $l_{2,1}$ to $l_{0,1}$ and is represented by the edge to the right neighbor of the marked node in Fig. 3. Indeed, the corresponding proof obligation

$$k = 2 \wedge k \neq 1 \wedge k' = k \wedge c_1' = c_1 \wedge c_2' = c_2 \;\Rightarrow\; k' = 2$$

is obviously correct. The other possible transition of process 1 in the XTG corresponds to a move to the critical section (location $l_{3,1}$). Because no matching node is reachable in the predicate diagram, the proof obligation becomes

$$k = 2 \wedge k = 1 \wedge c_1 \geq 2 \wedge k' = k \wedge c_1' = c_1 \wedge c_2' = c_2 \;\Rightarrow\; \mathbf{false}$$

which holds because the left-hand side is contradictory. Effectively, we demonstrate that process 1 cannot enter when process 2 is already inside its critical section. The remaining proof obligations are similar. (Observe that the PDT

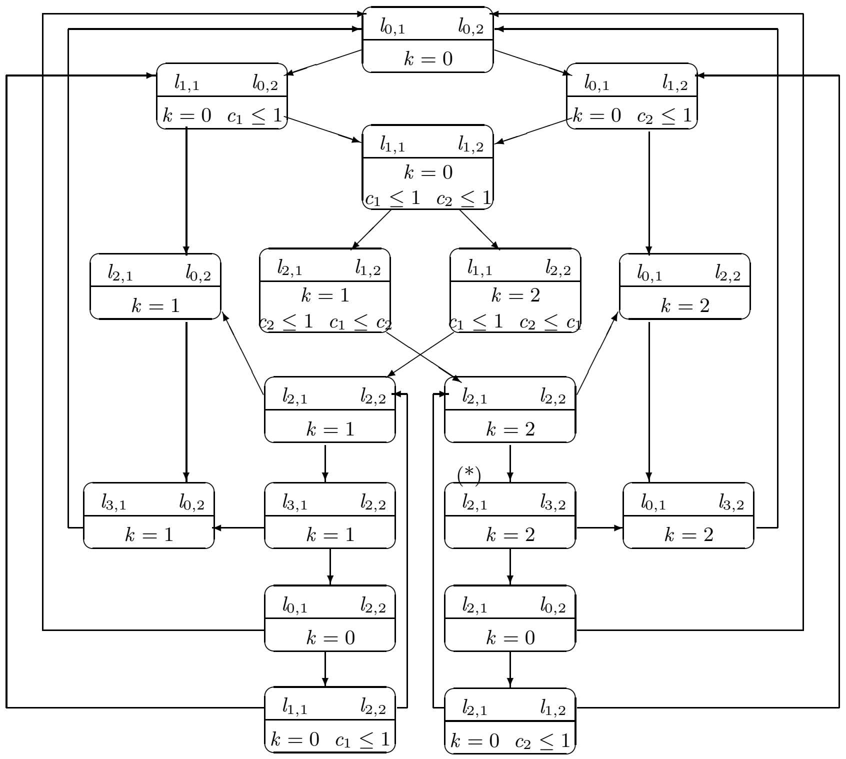

$l_{0,1}$ $l_{0,2}$ $k = 0$
$l_{1,1}$ $l_{0,2}$ $k = 0$ $c_1 \leq 1$
$l_{0,1}$ $l_{1,2}$ $k = 0$ $c_2 \leq 1$
$l_{1,1}$ $l_{1,2}$ $k = 0$ $c_1 \leq 1$ $c_2 \leq 1$
$l_{2,1}$ $l_{0,2}$ $k = 1$
$l_{2,1}$ $l_{1,2}$ $k = 1$ $c_2 \leq 1$ $c_1 \leq c_2$
$l_{1,1}$ $l_{2,2}$ $k = 2$ $c_1 \leq 1$ $c_2 \leq c_1$
$l_{0,1}$ $l_{2,2}$ $k = 2$
$l_{2,1}$ $l_{2,2}$ $k = 1$
$l_{2,1}$ $l_{2,2}$ $k = 2$
(*)
$l_{3,1}$ $l_{0,2}$ $k = 1$
$l_{3,1}$ $l_{2,2}$ $k = 1$
$l_{2,1}$ $l_{3,2}$ $k = 2$
$l_{0,1}$ $l_{3,2}$ $k = 2$
$l_{0,1}$ $l_{2,2}$ $k = 0$
$l_{2,1}$ $l_{0,2}$ $k = 0$
$l_{1,1}$ $l_{2,2}$ $k = 0$ $c_1 \leq 1$
$l_{2,1}$ $l_{1,2}$ $k = 0$ $c_2 \leq 1$

Fig. 3 A PDT for Fischer's protocol (cf. Fig. 2).

of Fig. 3 contains no time-passing edges other than the self-loops, which we do not show explicitly according to our convention.)

Because no node of the PDT corresponds to both processes being in their critical sections, we conclude that Fischer's protocol ensures mutual exclusion. The verification is supported by the DIXIT toolkit [13]. Centered around a graphical editor for drawing a predicate diagram, proof obligations for proving conformance can be generated, LTL properties can be verified by model checking, and counter-examples can be visualized. For our example, DIXIT reports that the diagram satisfies mutual exclusion. While DIXIT generates the proof obligations for establishing conformance, it does not yet contain a theorem proving component. We have successfully used the SMT solver CVC-LITE [1] to discharge the proof conditions for this example.

6 Predicate Diagrams for Parameterized Systems

Verification of parameterized systems is often done by hand, or with the guidance of a theorem prover [15, 23]. Methods based on the semi-automatic construction of a processes invariant are proposed in [27]. However, it is not in general possible to obtain a finite state process invariant. A typical parameterized system consists of an *arbitrary* number of identical processes interacting via synchronous or asynchronous communication. A typical example are mutual exclusion protocols for an arbitrary number of processes competing for a common resource.

Conventional model checking techniques can be used to verify instances of parameterized systems for fixed parameter values, for example for a fixed number of participant processes. In order to prove the correctness for any parameter value, we need to construct a single syntactic object that represents (an abstraction for) the entire family of systems. We discuss the use of PDTs for the verification of parameterized systems in this section, and we use an n-process version of Fischer's mutual-exclusion protocol as a running example.

We consider two classes of properties: properties of the entire system, i.e. concerning all processes, and "per-process" properties that should hold of every single process in the system. The latter properties are sometimes referred to as *universal* properties. Given a parameterized system that consists of N processes, universal properties are expressed as formulas of the form $\forall k \in 1...N : P(k)$. Baukus et al. [10] have considered techniques for the verification of *universal* properties of parameterized systems based on the transformation of an infinite family of systems into a single transition system expressed as a formula in WS1S and applying abstraction techniques on this system.

In Section 6.1, we explain the use of predicate diagrams for the verification of properties concerning the entire system, using the example of Fischer's protocol for $N \geq 1$ processes. In Section 6.2, we use predicate diagrams to prove *universal* properties of Fischer's protocol.

6.1 Verification of Properties related to the Whole System

For the N processes version of Fischer's protocol, we write $p[i].loc$ and $p[i].c$ to denote the control location and the local clock of process i, and denote by cs the set of processes that are in their critical region, i.e.

$$cs \;=\; \{i \in 1..N : p[i].loc = 3\}$$

As for the two-process version, the system contains a shared variable k that holds the process allowed to enter the critical region. The overall approach to verification proceeds in two steps as before by establishing conformance of a predicate diagram with respect to an XTG and by model checking the finite-state PDT.

Fig. 4 gives a PDT for Fischer's protocol for n-processes, which can be shown to conform to the XTG by discharging the conditions of Theorem 1. For example, we consider the possible transitions from the node N_1 marked $(*)$ in the PDT of Fig. 4. Of particular interest are the transitions of process k: after a certain process i takes a transition from location 1 to location 2, the process i can be the process k with reset local clock ($c_i = 0$). Some other

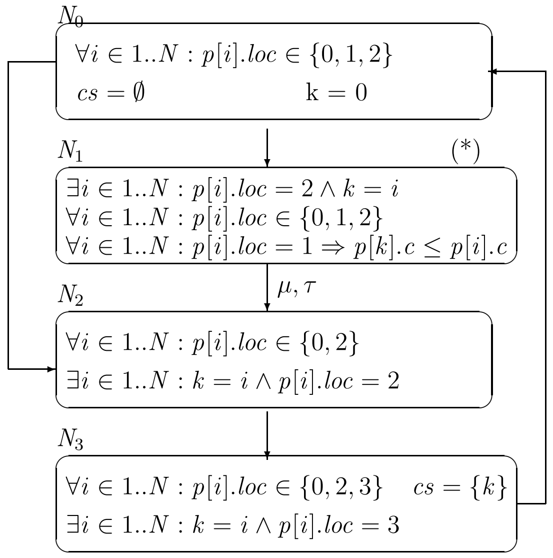

N_0
$\forall i \in 1..N : p[i].loc \in \{0,1,2\}$
$cs = \emptyset$ k = 0
N_1
(*)
$\exists i \in 1..N : p[i].loc = 2 \wedge k = i$
$\forall i \in 1..N : p[i].loc \in \{0,1,2\}$
$\forall i \in 1..N : p[i].loc = 1 \Rightarrow p[k].c \leq p[i].c$
μ, τ
N_2
$\forall i \in 1..N : p[i].loc \in \{0,2\}$
$\exists i \in 1..N : k = i \wedge p[i].loc = 2$
N_3
$\forall i \in 1..N : p[i].loc \in \{0,2,3\}$ $cs = \{k\}$
$\exists i \in 1..N : k = i \wedge p[i].loc = 3$

Fig. 4 A PDT for Fischer's protocol for N processes

processes, which are in location 1 with clock constraint ($c \leq 1$), also wait for entering to location 2 in case their local clocks should be greater than equal to the process k's local clock since whenever the process i takes a transition from location 1 to location 2, its clock becomes reset and the other processes are still in location 1 as increasing their clocks. The corresponding predicate

$$\forall i \in 1..N : p[i].loc = 1 \Rightarrow p[k].c \leq p[i].c$$

added into the node N_1

This process k has transition to control location 2 is covered by the transition from node N_1 to node N_2 of the PDT, because the corresponding proof obligation

$$\begin{aligned} & N_1 \wedge\ k' = k \wedge\ p' = [p \text{ EXCEPT } p[i]'.loc \neq 1] \\ \Rightarrow\ & N_1' \vee N_2' \end{aligned}$$

obviously holds.

The remaining proof obligations are similar, and CVC Lite can be used to discharge them. (Again, the PDT of Fig. 4 contains no time-passing edges other than the self-loops, which we do not show explicitly according to our convention.)

The PDT of Fig. 4 can be used to prove the mutual exclusion property, which states that no two processes can be simultaneously inside their critical sections, which can be expressed as the LTL formula

$$\Box(\forall i, j \in 1..N : i \in cs \wedge j \in cs \Rightarrow i = j).$$

In this case, it is not quite enough to consider the predicates as Boolean variables and use standard model checkers because we need elementary set theory to infer mutual exclusion from the node labels. Model checking would still work if extra predicates were added to the labels of the PDT nodes.

Alternatively, we can again use CVC Lite to infer the property of interest from each of the node labels. For instance above LTL formula can be expressed as follow.

$$\Box(cs = \emptyset \vee cs = \{k\})$$

And CVC Lite is queried about the satisfiability of this alternative formula.

6.2 Verification of Universal Properties

Many properties of parameterized systems are naturally expressed as formulas of the form $\forall i \in 1..N : P(i)$ where $P(i)$ is an LTL property that describes a property of a single process. Such properties can be verified by distinguishing a single process $i \in 1..N$ from the rest of the processes. Formally, this corresponds to introducing a Skolem constant i and proving the property $i \in 1..N \Rightarrow P(i)$.

The idea in constructing a PDT for a *universal* property is thus to follow the evolution of the distinguished process i and to represent the transitions of the other processes in a separate part of the predicate diagram. Fig. 5 illustrates this idea for the N processes version of Fischer's protocol. In order to simplify the node labels, we write $\forall Q(j)$ as a short-hand for

$$\forall j \in 1..N \setminus \{i\} : Q(j)$$

and similarly for $\exists Q(j)$.

The PDT of Fig. 5 is a correct abstraction of the N processes version of Fischer's protocol. Again, we use Theorem 1 for proving this conformance.

For the conformance checking, we consider the possible transitions of process j from the node marked $(*)$ the PDT of Fig. 5 with corresponding control locations: the transition to location 3 is captured by the downward edge, and indeed the proof obligation

$$\begin{aligned}
&p[i].loc \in \{0,2\} \wedge \forall p[j].loc \in \{0,2\} \\
&\wedge\ \exists k = j \wedge p[k].loc = 2 \wedge cs = \emptyset \\
&\wedge\ k' = k \wedge cs' = k \wedge p' = [p \text{ EXCEPT } p[j]'.loc = 3] \\
\Rightarrow\ &p[i]'.loc \in \{0,2\} \wedge \forall p[j]'.loc \in \{0,2,3\} \\
&\wedge\ \exists k' = j \wedge cs' = k
\end{aligned}$$

is easily seen to hold (and proven by CVC Lite) and the remaining proof obligations for possible trasitions corresponding XTG are similar. The mutual exclusion property of Fischer's protocol for N processes can be verified using the PDT in Fig. 5 in a similar way as for the PDT of Sec. 6.1.

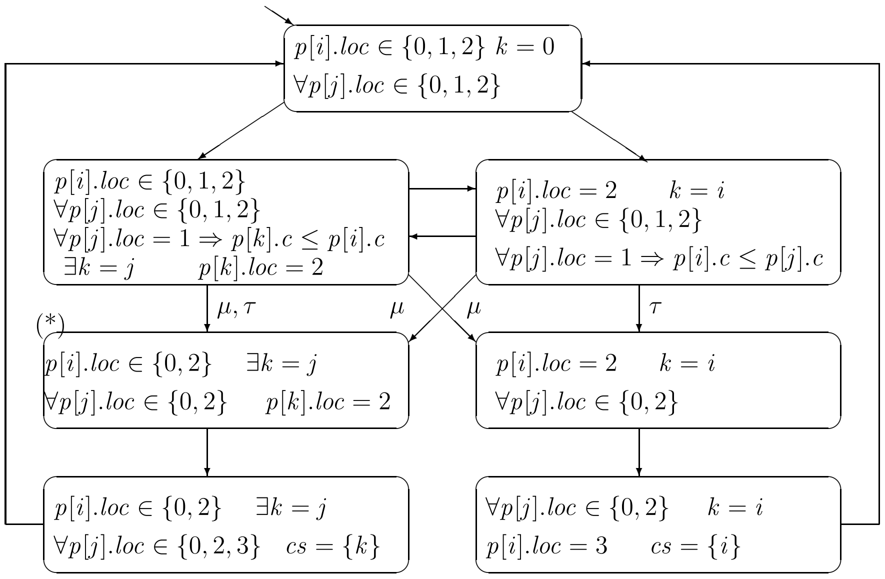

$p[i].loc \in \{0,1,2\}$ $k = 0$
$\forall p[j].loc \in \{0,1,2\}$
$p[i].loc \in \{0,1,2\}$
$\forall p[j].loc \in \{0,1,2\}$
$\forall p[j].loc = 1 \Rightarrow p[k].c \leq p[i].c$
$\exists k = j$ $p[k].loc = 2$
$p[i].loc = 2$ $k = i$
$\forall p[j].loc \in \{0,1,2\}$
$\forall p[j].loc = 1 \Rightarrow p[i].c \leq p[j].c$
μ, τ
μ
μ
τ
(*)
$p[i].loc \in \{0,2\}$ $\exists k = j$
$\forall p[j].loc \in \{0,2\}$ $p[k].loc = 2$
$p[i].loc = 2$ $k = i$
$\forall p[j].loc \in \{0,2\}$
$p[i].loc \in \{0,2\}$ $\exists k = j$
$\forall p[j].loc \in \{0,2,3\}$ $cs = \{k\}$
$\forall p[j].loc \in \{0,2\}$ $k = i$
$p[i].loc = 3$ $cs = \{i\}$

Fig. 5 A PDT for Fischer's protocol for N processes.

7 Discussion and Future Work

In this article, we have proposed the format of predicate diagrams for timed systems (PDT) as a notation to represent Boolean abstractions of real-time systems. This format is a variant of predicate diagrams for discrete systems [11]; in particular, time-passing transitions are distinguished from discrete ones. We have also established a set of proof obligations for proving conformance between an XTG model of a timed system and a PDT. These proof obligations are first-order formulas and can often be established using automatic provers for first-order logic, such as SMT solvers.

In this sense, PDTs constitute an interface between verification techniques based on deduction and model checking. Basically, the idea is that only a finite set of equivalence classes of system configurations need be distinguished for the proof of a given LTL property. Predicates are interpreted during the conformance proof, whereas they are considered as atomic propositions during model checking. The format of predicate diagrams is supported by the DIXIT toolkit, and we have demonstrated its use via Fischer's mutual-exclusion protocol for two processes.

It is well known that Fischer's two-process protocol can be verified by real-time model checking. However, the PDT format can equally well be used for parametric systems, such as the N process version of Fischer's protocol. For these applications, a family of processes is represented in a single diagram. We are often interested in *universal* properties of parametric systems, and they can be established by distinguishing a single process and following its behavior separate from that of the remainder of the system.

We consider this work as a first step towards the application of Boolean abstractions in the verification of real-time systems. One of the current limitations lies in the fact that we abstract from the precise amount of time that may elapse in a time-passing transition. Thus, we cannot easily verify quantitative properties, such as upper bounds on global response times, although properties that mention individual clocks can be verified. We intend to study two possible solutions to this problem, either by using a timed temporal logic (TLTL) or by introducing auxiliary clocks during verification, as suggested by Henzinger et al. [16] and by Tripakis [26]. This would in particular allow us to take advantage of model checking tools for real-time systems such as Uppaal [7] or PMC.

Besides, we aim at reducing the number of verification conditions that users have to discharge with the help of a theorem prover in order to establish conformance. In fact, we consider the proof obligations of Theorem 1 mainly as a litmus test to establish the conditions that a PDT should satisfy, and we observe that most of them are quite trivial for typical examples. It will be interesting to restrict attention to specific classes of systems that give rise to decidable proof obligations,thus enabling the automatic construction of PDTs.

We also intend to study in more detail techniques of refinement for the construction of PDTs, given an XTG and a set of predicates of interest. Preliminary work on combining tools for abstract interpretation and state space exploration has been reported in [18,19], but more experience will be necessary in order to identify complete abstractions for real-time systems. Although a PDT obtained by abstract interpretation is unlikely to already satisfy the desired correctness properties, it can then be refined, either by user intervention or by algorithmic abstraction refinement guided by counter-examples. This would significantly raise the degree of automation possible in the verification of complex real-time systems.

References

1. Cvc lite homepage. available at. http://www.cs.nyu.edu/acsys/cvcl.
2. J.-R. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.
3. R. Alur and D. Dill. The theory of timed automata. In *Proceedings REX workshop on Real-Time: Theory and Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 45–73. Springer-Verlag, 1991.
4. R. Alur and D. Dill. The theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
5. H. Amjad. Combining model checking and theorem proving. Technical report, UCAM-CL-TR-601, ISSN 146-2986, Cambridge University, pages 15-16, September 2004.
6. M. Ammerlaan, R. L. Spelberg, and W. Toetenel. XTG – an engineering approach to modelling and analysis of real-time systems. In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, pages 88–97. IEEE press, 1998.

7. T. Amnell and many others. UPPAAL: Now, next, and future. In F. C. et al., editor, *Modeling and Verification of Parallel Processes*, LNCS(2067):99-124. Springer-Verlag, Berlin, 2001.
8. E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of timed automata. In *Proceedings of the 1st International Workshop on Hybrid and Real-Time Systems*, volume 1201 of *Lecture Notes in Computer Science*, pages 346–360. Springer-Verlag, 1997.
9. T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In *Principles of Programming Languages (POPL 2002)*, pages 1–3, 2002.
10. K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting WS1S systems to verify parameterized networks. In *Proceedings of the 6th Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 188–204. Springer-Verlag, 2000.
11. D. Cansell, D. Mery, and S. Merz. Diagram refinements for the design of reactive systems. *Journal of Universal Computer Science,7(2):159-174*, 2001.
12. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169, 2000.
13. L. Fejoz, D. Méry, and S. Merz. DIXIT: a graphical toolkit for predicate abstractions. In R. Bharadwaj and S. Mukhopadhyay, editors, *Intl. Workshop Automatic Verification of Infinite-State Systems (AVIS 2005, to appear in ENTCS)*, pages 39–48. LFCS, Univ. of Edinburgh, 2005. see also `http://www.loria.fr/equipes/mosel/dixit`.
14. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Proceedings 9th International Conference on Computer Aided Verification, CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 1997.
15. K. Havelund and N. Shankar. Experiments in Theorem Proving and Model Checking for Protocol Verification. In M.-C. Gaudel and J. Woodcock, editors, *FME'96: Industrial Benefit and Advances in Formal Methods*, pages 662–681. Springer-Verlag, 1996.
16. T. Henzinger and O. Kupferman. From quantity to quality. In *Proceedings of the 1st International Workshop on Hybrid and Real-Time Systems*, volume 1201 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.
17. T. A. Henzinger, R. Jhala, R. Majumdar, and K. McMillan. Abstractions from proofs. In *31st Annual Symp. Princ. of Prog. Lang. (POPL 2004)*, pages 232–244. ACM Press, 2004.
18. E.-Y. Kang. Parametric analysis of real-time embedded systems with abstract approximation interpretation. In *26th International Conference on Software Engineering*, pages 39–41, 2004.
19. E.-Y. Kang. Real-time system verification techniques based on abstraction/deduction and model checking. In *Proceedings the 5th International Conference on Integrated Formal Methods*, Doctoral Symposium, CS-Report 05-29, pages 26–32. Technische Universiteit Eindhoven, 2005.
20. Y. Kesten and A. Pnueli. Modularization and abstraction: The keys to practical formal verification. In *Proceedings of the 23th International Sumposium on Mathematical Foundations of Computer Science*, volume 1450 of *Lecture Notes in Computer Science*, pages 54–71. Springer-Verlag, 1998.

21. K. Kristoffersen, F. Laroussinie, K. Larsen, P. Pettersson, and W. Yi. A compositional proof of a real-time mutual exclusion protocol. Technical report, BRICS, Aalborg University, Denmark, 1996.
22. R. Lutje Spelberg, W. Toetenel, and M. Ammerlaan. Partition refinement in real-time model checking. In *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1486 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, 1998.
23. Z. Manna and A. Pnueli. Verification of parameterized programs. In *Z. Manna and A. Pnueli. Verification of parameterized programs. In E. Borger, editor, Specification and Validation Methods, pages 167–230. Clarendon Press, 1995.*, 1995.
24. R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.
25. R. L. Spelberg. *Model Checking Real-Time Systems based on partition refinement*. PhD thesis, Delft University, 2004.
26. S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulations. In *Proceedings 8th International Conference on Computer Aided Verification, CAV'96*, volume 1102 of *Lecture Notes in Computer Science*, page ... Springer-Verlag, 1996.
27. P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Proceedings of Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science, pages 68–80. Springer-Verlag, 1990.
28. S. Yovine. Kronos: A verification tool for real-time systems. *Springer International Journal of Software Tools for Technology Transfer, 1(1-2):123-133*, Oct,1997.