Predicate Diagrams for the Verification of Real-Time Systems

Eunyoung Kang, Stephan Merz

Predicate Diagrams for the Verification of Real-Time Systems

Eun-Young Kang [1] Stephan Merz

MOSEL project, LORIA
Nancy, France
{Eun-Young.Kang,Stephan.Merz}@loria.fr

Abstract

We propose a 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 modeling 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 paper is to extend the format of predicate diagrams to timed systems. We also 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. The formalism is supported by a toolkit, and we demonstrate its use at the hand of Fischer's real-time mutual-exclusion protocol.

Key words: Real-time systems, verification, abstraction, XTG, predicate diagrams, theorem proving, model checking

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 [2,12] 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 significant systems [4,22]. Model checking is attractive because it is fully automatic, but also because it provides counter-examples when the property of interest does not hold of the system.

[1] Supported by NWO, Faculty of EEMCS, The Technical University of Delft, The Netherlands.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

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 suitable sets of axioms and inference rules. Although they can be supported by theorem provers and interactive proof assistants, their use requires considerable expertise and tedious user interaction.

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 these idea more concrete, we need to identify a suitable format that serves as an interface between deductive and algorithmic techniques and gives rise to feasible verification conditions.

Predicate abstraction [11,21] has emerged as a fruitful basis for software verification. It underlies tools such as SLAM [6] and BLAST [13], 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 [8,9], we have proposed a format of presenting predicate abstractions, called predicate diagrams, with an emphasis on proving liveness properties of discrete systems. In this paper 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 representation developed at TU Delft [16,3]. 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.

This paper is structured as follows. Section 2 presents XTGs as models of real-time systems. Section 3 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 in section 4. Section 5 discusses future work and concludes the paper.

2 Extended Timed Automata Graphs

We model real-time systems as XTGs (extended timed automata graphs) [3], a notation that combines the familiar framework of timed automata [1], syn-

chronous value passing between parallel processes, and a language for modeling data. The semantics of XTGs is defined in terms of *timed (Kripke) structures*, also known as timed transition systems.

Definition 2.1 *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 a (finite or 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*.*

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 2.2 *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 2.3 *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* $\rho : V \to Val$ *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

$$[\![\_]\!]_\_ : Expr \to (V \to Val) \to Val$$

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 [17]. A definition of the core syntax and semantics of XTGs was given by Spelberg [19]. In the present paper we restrict ourselves to shared variables and for simplicity do not consider value passing.

Definition 2.4 *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.*
- $U \subseteq E$ *identifies the subset of urgent edges.*

An XTG is a finite set of XTG processes.

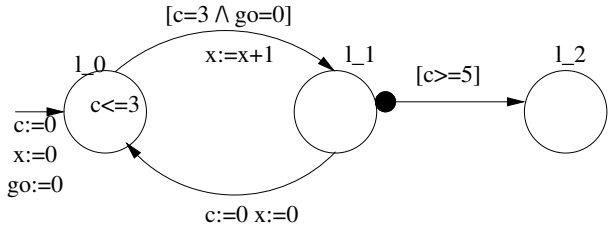
Figure 1 shows a sample XTG consisting of a single process, both in its textual (Fig. 1.a) and graphical (Fig. 1.b) representations. The XTG process consists of three locations l_0, l_1, and l_2. The edge from l_1 to l_2 is urgent, as indicated by the keyword **asap** in Fig. 1.a and by the black dot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

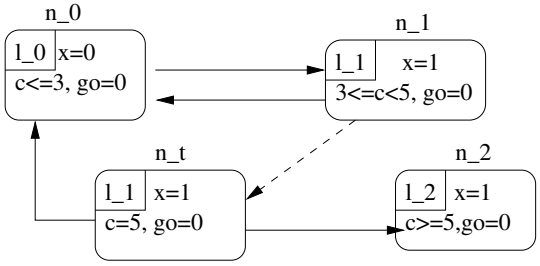
Definition 2.5 *Assume given an XTG* $\mathcal{X}$ *with processes* $P_1, \ldots, P_n$*. The timed structure* $\mathcal{T} = \langle S, S_0, T \rangle$ *generated by* $\mathcal{X}$ *is the smallest structure such that*

```
system example
state integer x:=0,go:=0
process P p1;
graph P
state clock c:=0
init l_0
locations
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.

- *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.*
- *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.

3 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 3.2.

3.1 The PDT Notation

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 an XTG 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.

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 3.1 *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 a (finite or 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.

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.

3.2 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 3.2 *Given an XTG $\mathcal{X}$, a PDT Δ, and a run $\pi = \langle \vec{l}_0, \rho_0 \rangle \xrightarrow{\lambda_0} \langle \vec{l}_1, \rho_1 \rangle \ldots$ 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, either both finite or both infinite),*
- *$n_i = \langle \vec{l}_i, P \rangle$ for some $P \subseteq \overline{\mathcal{P}}$ such that $[\![p]\!]_{\rho_i} = tt$ for all $p \in P$ 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 some initial PDT node corresponds to any state satisfying the initial condition of the XTG. Inductively, given any XTG state s corresponding to some PDT node and any transition from s to some successor XTG state s', that transition can be mapped to a transition of 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 doube 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 3.3 *Assume that $\mathcal{X}$ is 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 $\mathcal{X}$:*

(i) $\bigwedge_{j=1}^{m} Init_j \wedge I(l_{0,j}) \;\Rightarrow \bigvee_{\langle l_{0,1},\ldots,l_{0,m},P\rangle \in N_0} P$

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.

(ii) *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 $n \rightarrow_\mu n'$.*

$$P \wedge g \wedge \bigwedge_{j=1}^{m} I(l_j) \wedge I'(l_j') \wedge \bigwedge_{\langle v,e\rangle \in u} v' = e \wedge \bigwedge_{v \in V \setminus V_u} v' = v \;\Rightarrow \bigvee_{\langle l_1',\ldots,l_m',Q\rangle \in N'} Q'$$

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'.

(iii) *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 $n \rightarrow_\tau n''$.*

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

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 (sketch). Given a run $\pi = \langle \vec{l_0}, \rho_0\rangle \xrightarrow{\lambda_0} \langle \vec{l_1}, \rho_1\rangle \ldots$ 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 $n = \langle \vec{l}, P\rangle$ corresponding to the

XTG configuration $s_i = \langle \vec{l}_i, \rho_i \rangle$ 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. 2.5), and therefore the guard of that edge must be true in ρ_i and its updates will be performed during the transition to state $\langle \vec{l}_{i+1}, \rho_{i+1} \rangle$. 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 $\vec{l}_{i+1}$ 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 3.3 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

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

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:

$$\begin{aligned} & x = 0 \wedge c \leq 3 \wedge go = 0 \wedge c = 3 \wedge go = 0 \wedge x' = x + 1 \wedge go' = go \wedge c' = c \\ \Rightarrow\ & x' = 1 \wedge 3 \leq c' \wedge c' < 5 \wedge go' = 0 \end{aligned}$$

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

$$\begin{aligned} & x = 1 \wedge 3 \leq c \wedge c < 5 \wedge go = 0 \wedge \delta \in \mathbb{R}^{\geq 0} \wedge c' = c + \delta \wedge x' = x \wedge go' = go \\ & \wedge \forall \varepsilon < \delta : c'' = c + \varepsilon \wedge x'' = x \wedge go'' = go \Rightarrow \neg(c'' \geq 5) \\ \Rightarrow\ & (x' = 1 \wedge 3 \leq c' \wedge c' < 5 \wedge go' = 0) \vee (x' = 1 \wedge c' = 5 \wedge go' = 0) \end{aligned}$$

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.

3.3 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 Δ 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 [8]. For our experiments, we use the DIXIT tool [10].

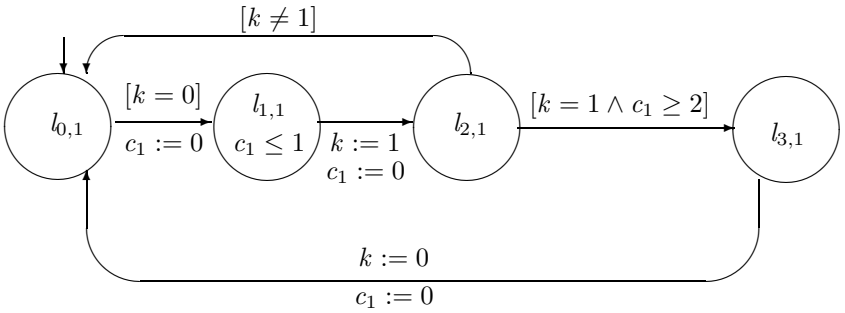


Fig. 2. An XTG for Fischer's protocol (process 1).

We claim that any property φ that model checking establishes over some PDT Δ also holds of the XTG $\mathcal{X}$ provided that Δ conforms to $\mathcal{X}$. Indeed, let π be any run of $\mathcal{X}$. By the definition of conformance, we can find a trace σ of π in Δ. Since φ is assumed to hold of Δ, 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. Nevertheless, these counter-examples can be helpful to refine the abstraction.

4 An example: Fischer's protocol

We illustrate the use of PDTs at the hands of Fischer's well-known real-time protocol for ensuring mutual exclusion between two processes [5,15]. 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 3.3. 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 $l_{3,2}$. For the XTG transition from $l_{2,1}$ to $l_{0,1}$, we find that the right neighbor node in the PDT activates the corresponding locations, and we obtain the proof obligation

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

which obviously holds. The other possible transition of process 1 in the XTG

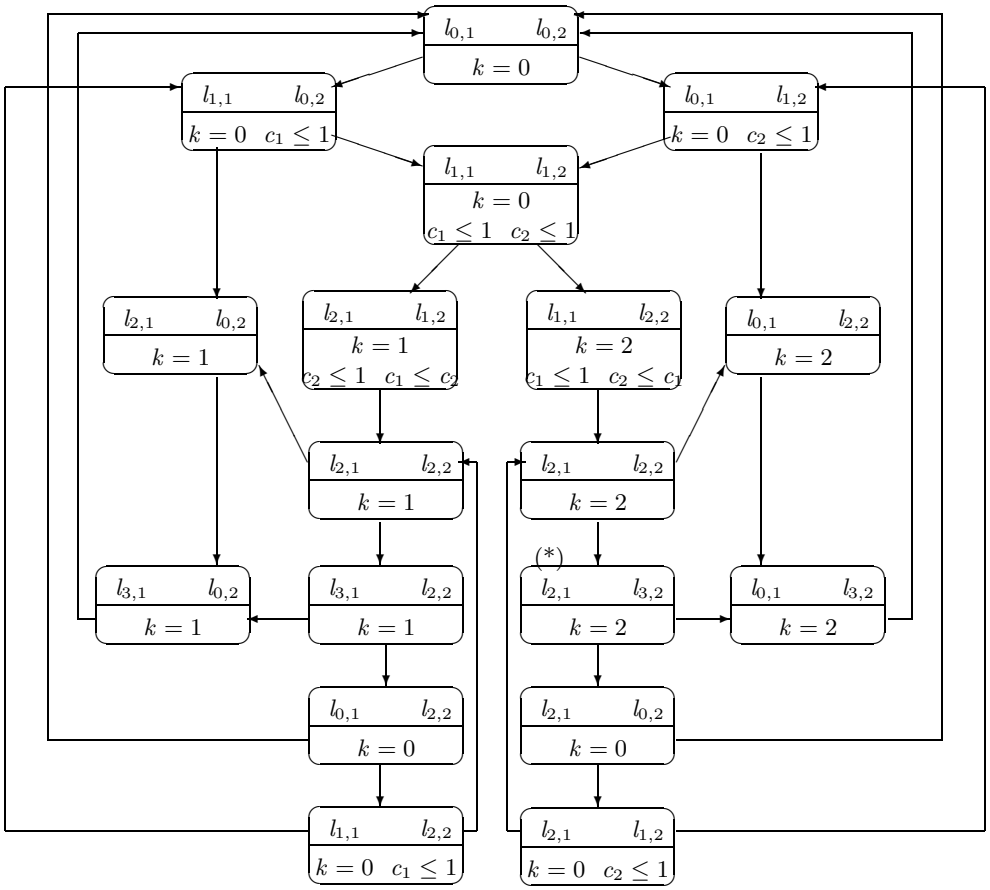


Fig. 3. A PDT for Fischer's protocol (cf. Fig. 2).

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

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

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 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 [10]. 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.

5 Discussion and Future Work

In this paper, 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 [8,9], in particular, by distinguishing discrete and time-passing transitions. We have also established a set of proof obligations for proving conformance between an XTG model of a timed system and a PDT.

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, and it can be argued that PDTs here simply recast standard representations used for symbolic model checking in a format based on predicates. However, these same techniques extend to systems with unbounded or even infinite state spaces (apart from the infinity due to real time) where model checking alone is no longer sufficient. For example, an n-process version of Fischer's protocol could be represented as a relatively straightforward generalization of the two-process PDT shown in Fig. 3.

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. [12] and by Tripakis [20]. This would in particular allow us to take advantage of model checking tools for real-time systems such as Uppaal [7] or PMC [18].

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 3.3 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.

We also intend to study techniques of abstract interpretation for the construction of PDTs, given an XTG and a set of predicates of interest. Pre-

liminary work on combining tools for abstract interpretation and state space exploration has been reported in [14], but more experience will be necessary in order to identify adequate 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] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.

[2] R. Alur and D. Dill. The theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] M. Ammerlaan, R. Lutje Spelberg, and W.J. 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.

[4] Tobias Amnell and many others. UPPAAL: Now, next, and future. In F. Cassez et al., editor, *Modeling and Verification of Parallel Processes*, volume 2067 of *Lecture Notes in Computer Science*, pages 99–124. Springer-Verlag, Berlin, 2001.

[5] 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.

[6] 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.

[7] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer-Verlag, 1995.

[8] Dominique Cansell, Dominique Mery, and Stephan Merz. Predicates diagrams for the verification of reactive systems. In *Proceedings the 2nd International Conference on Integrated Formal Methods*, volume 1945 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[9] Dominique Cansell, Dominique Mery, and Stephan Merz. Diagram refinements for the design of reactive systems. *Journal of Universal Computer Science,7(2):159-174*, 2001.

[10] Loic Fejoz, Dominique Méry, and Stephan 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)*, pages 39–48. LFCS, Univ. of Edinburgh, 2005. see also http://www.loria.fr/equipes/mosel/dixit.

[11] 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.

[12] T.A. 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.

[13] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth McMillan. Abstractions from proofs. In *31st Annual Symp. Princ. of Prog. Lang. (POPL 2004)*. ACM Press, 2004.

[14] Eun-Young Kang. Parametric analysis of real-time embedded systems with abstract approximation interpretation. In *26th International Conference on Software Engineering*, 2004.

[15] K.J. Kristoffersen, F. Laroussinie, K.G. Larsen, P. Pettersson, and W. Yi. A compositional proof of a real-time mutual exclusion protocol. Technical report, BRICS, Aalborg University, Denmark, 1996.

[16] R.F. Lutje Spelberg, W.J. 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.

[17] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[18] R.F. Lutje Spelberg and W.J. Toetenel. Parametric real-time model checking using splitting trees. *Nodic Journal of Computing 8(2001), 88-120*, 2001.

[19] Ronald Lutje Spelberg. *Model Checking Real-Time Systems based on partition refinement*. PhD thesis, Delft University, 2004.

[20] S.Tripakis. *The Formal Analysis of Timed Systems in practice*. PhD thesis, University of Joseph Fourrier de Grenoble, 1998.

[21] 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.

[22] S. Yovine. Kronos: A verification tool for real-time systems. *Springer International Journal of Software Tools for Technology Transfer*, 1997.