Diagram-Based Verification of Real-Time Systems using Timed Predicate Diagrams

Cecilia E. Nugraheni

Computer Science Dept., Parahyangan Catholic University, Bandung, Indonesia.

Summary
Computers are frequently used in critical applications where predictable response times are essential for correctness. Such systems are called real-time systems and are a class of reactive systems. Many verification methods for verifying reactive systems were proposed, including diagram-based verification. One of diagrams used for reactive systems verification is predicate diagrams proposed by Cansell et.al. It has been shown that this diagram can be used for the verification of *discrete* reactive systems.
In this paper, a class of diagrams called timed predicate diagrams is introduced. These diagrams are a variant of predicate diagrams, which can be used to verify real-time systems. This method has been applied on an example problem which is Fischer's protocol.

Key words:
Reactive systems, real-time systems, verification, TLA, predicate diagrams.

1. Introduction

Reactive systems are computer programs, implemented in hardware or software or a combination thereof, which are expected to maintain an ongoing interaction with their environment. Reactive systems are commonly classified as *discrete*, *real-time* and *hybrid* systems [1]. A discrete system only represents the qualitative aspect of time, that is the order of events. A realtime system captures the metric aspects of time [2]. In hybrid systems we allow the inclusion of variables that evolve continuously over time between discrete events.

Verification of reactive systems consists of establishing whether a reactive system satisfies its specification, that is, whether all possible behaviors of the system are included in the property specified, such as safety and liveness properties. Verification techniques of reactive systems traditionally are classified into two groups, which are deductive method and algorithmic method. The combination of both techniques, such as deductive model checking, has also been suggested.

The using of diagrams in verifying reactive systems has been proposed, because they can reflect the intuitive understanding of the systems and its specification. Diagram can also be seen as an abstraction of the system, where properties of the diagram are guaranteed to hold for the systems as well.

In this paper *timed predicate diagrams* is introduced. These diagrams are a class of predicate diagrams [3], which are intended as the basis for the verification of real-time systems. This method integrates deductive verification and algorithmic techniques. The correspondence between the original specification and the diagram is established by non-temporal proof obligations, whereas model checking can be used to verify properties over finite-state abstractions. Following [3], the Temporal Logic of Actions (TLA) [4] from Lamport is used to formalize this approach.

This paper is structured as follows. Section 2 describes briefly the specification used for real-time systems in TLA. The definition of timed predicate diagrams will be given in Section 3. Section 4 describes how to verify real-time systems using timed predicate diagrams. An illustration of this method, which is Fischer's protocol, is given in Section 5. Section 6 concludes this paper.

2. Specification

The alphabet of TLA consists of the alphabet of propositional logic and additional symbols $\Box$, [,], and ‘ (primed). There are three kinds of formulas: state, action and temporal formulas. A *state* formula is an expression about the values of system's variables at a particular time (t), e.g. x=1. An *action* formula is identified by the occurrence of *primed*. For example, $x'=x+1$ is an action where x' represents the value of x at t+1. Whereas a *temporal* formula is identified by the occurrence of $\Box$. Formula $\Box x=1$ asserts that the value of x is always (for every t) equals to 1. A formula of the form $[A]_v$ is a short form of $Next \vee v' = v$. Two other short forms that are frequently used are $\Diamond A \equiv \neg\Box\neg A$ and $\langle A\rangle_v \equiv A \wedge v' \neq v$.

The semantic of a TLA formula is relative to a behavior which an infinite sequence of states $\sigma = s_0 s_1 \ldots$. Given a

formula F, we write $\sigma \models F$ if F holds on σ. We also use the notation $s|[F]|$ to represents the value of F at state s.

The specifications of real-time systems are described by TLA formulas of the form:

$$RTSpec \equiv Init \wedge \Box [Next]_v \wedge RTNow(v) \wedge RT_1 \wedge \ldots \wedge RT_k \quad (1)$$

where

- *Init* is a state predicate that characterizes the system's initial state,
- *Next* is an action formula representing the next-state relation,
- v is the tuple of state discrete variables of interest,
- $RTNow(v)$ is a formula asserting that initially $now = 0$ and the tuple of system's variables, v, does not change when *now* advances and
- for every $i \in 1..k$, RT_i is a formula of the form

$$RTBound(A_i, v, t_i, d_i, e_i) \quad (2)$$

where

- A_i is a subaction of *Next*.
- t_i is a variable called the *timer* for A_i, such that value of t should be reset to 0 by an $\langle A_i \rangle_v$ step or a step that disables $\langle A_i \rangle_v$ and a step that advances *now* should increment t by *now*' - *now* iff $\langle A_i \rangle_v$ is enabled.
- d_i and e_i are constants called the *lower bound* and *upper bound* of A_i respectively, such that $\langle A_i \rangle_v$ can be taken if it has been continuously enabled for at least d seconds since the last $\langle A_i \rangle_v$ step - or since the beginning of the behavior and $\langle A_i \rangle_v$ can be continuously enabled for at most e seconds before an $\langle A_i \rangle_v$ step occurs.

In Figure 1, a small example of a real-time system is given ($\aleph$ denotes the natural numbers). This system consists of two process, *Up* and *Down* and a shared variable x.

$$\begin{aligned} Init &\equiv x \in \aleph \\ Up &\equiv x' = x + 1 \\ Down &\equiv x > 0 \wedge x' = 0 \\ Loop &\equiv Init \wedge \Box[Up \vee Down]_x \wedge RTNow(x) \\ &\quad \wedge RTBound(Down, x, t, 0, 3) \end{aligned}$$

Figure 1. Module *Loop*.

Initially x is set to some natural number. Process *Up* keeps incrementing x; whereas process *Down* is responsible to set x to 0 whenever x is greater than 0. We put a time constraint on the process *Down*, such that x must be reset to 0 in not more than 3 seconds after x becomes greater than 0.

3. Timed Predicate Diagrams

Before we give the definition of timed predicate diagram, we look briefly the definition of predicate diagrams [3]. It is assumed that the underlying assertion language, by assumption, contains a finite set **O** of binary relation symbols $\prec$ that are interpreted by well-founded orderings. For $\prec \in \mathbf{O}$, its reflexive closure is denoted by $\preceq$. We write $\mathbf{O}^{=}$ to denote the set of relation symbols $\prec$ and $\preceq$ for $\prec$ in **O**.

A predicate diagram is a finite graph whose nodes are labeled with sets of (possibly negated) predicates, and whose edges are labeled with actions as well as optional annotations that assert certain expression to decrease with respect to an ordering in $\mathbf{O}^{=}$. A node of a predicate diagram represents the set of system states that satisfy the formulas contained in the node. An edge (n,m) is labeled with action A if A can cause a transition from a state represented by n to a state represented by m. An action A may have an associated fairness condition.

Definition 1. Assume given two finite sets **P** and **A** of state predicates and action names. A predicate diagram $G=(N,I,\delta,o,\zeta)$ over **P** and **A** consists of

- a finite set of $N \subseteq 2^{\mathbf{P}^*}$ of nodes (where $\mathbf{P}^*$ denotes the set containing all predicates in **P** and their negation),
- a finite set $I \subseteq N$ of initial nodes,
- a family $\delta = (\delta_A)_{A \in \mathbf{A}}$ of relations $\delta_A \subseteq N \times N$; we also denote by δ the union of the relation δ_A for $A \in \mathbf{A}$,
- an edge labeling o that associates a finite set $\{(t_1, \prec_1), \ldots, (t_k, \prec_k)\}$, of terms t_i paired with a relation $\prec_i \in \mathbf{O}^{=}$ with every edge $(n,m) \in \delta$, and
- a mapping $\zeta: \mathbf{A} \rightarrow \{NF, WF, SF\}$ that associates a fairness condition with every action in **A**; the possible values represent no fairness, weak fairness, and strong fairness.

Timed predicate diagrams, or TPDs for short, are a variant of predicate diagrams. The idea of these diagrams is to use the components of predicate diagrams related to the discrete properties and to replace the components related to the fairness conditions with some components related to real-time conditions.

For the components related to real-time property, we adopt the structure of timed-automata [5]. A TPD is equipped with a finite set of real-valued variables that measure time. These variables are called *timer*s. Every timer is associated with some predicates, which we call *time-constraints*.

Definition 2. A *time-constraint* is a state predicate of the form $c \neq z$ or c_1-$c_2 \neq z$ where c, c_1 and c_2 are timers, $\neq \in \{\leq,<,=,>,\geq\}$ and z is a real constant, including ∞.

For a set of timers C, we denote by Φ_C and $\psi(\Phi_C)$, the set of time-constraints over timers $c \in C$ and the set containing all $c \in C$ that appears in Φ_C, respectively.

Definition 3. Given a set of state predicates **P**, a set of actions **A**, a set of timers C and a set of time-constraints over the timers in C, Φ_C, TPD T over **P**, **A**, C and Φ_C is given by a tuple (N,I,δ, o, r, g, R) where

- N, I, δ and o as defined in Definition 1.
- A mapping $r : N \rightarrow 2^{\Phi C}$ that associates a set of time-constraints in Ψ_C with every node in N.
- A mapping $g : N \times N \rightarrow 2^{\Phi C}$ that associates a set of time-constraints in Ψ_C with every edge in δ.
- A mapping $R : N \times N \rightarrow 2^{C}$ that associates a set of timers in C with every edge in δ.

We say that the action $A \in$ **A** can be taken at node $n \in N$ iff $(n,m) \in \delta_A$ holds for some $m \in N$, and denote by $En(A) \subseteq N$ the set of nodes where A can be taken. We say that the action $A \in$ **A** can be taken along (n,m) iff $(n,m) \in \delta_A$.

A timer $c \in C$ is called an active timer on a node $n \in N$ if there exists some node $m \in N$ such that $g(n,m)$ contains some time-constraint over c. We denote by $act(n)$ the set of active timers on n.

Like PDs, TPDs can be viewed as a labeled directed graph, where the nodes of may be labeled with one more set of predicates which we call *time-invariant*. This invariant and the state predicates over system's discrete variables must be satisfied on every node.

The edges of TPDs may be labeled with actions and time-constraints, which we call *guards*, and a set of timers, which we call *reset timers*. Guards will be used to model the timing conditions that constrain the execution of transitions. Every reset timer will be reset to 0 whenever this transition is taken. Besides the transitions that are explicitly represented by the edges and the stuttering transitions, τ, we introduce a special transition called *tick* for representing the elapsing time. Similar to τ, transition *tick* remains in the source node. For every node n and every active timer on it, we require that the value of every active timer increases whenever *tick* is taken.

Figure 2 shows a TPD over **P**={x=0, $x\geq$0}, **A**={*Up*, *Down*}, C = {t}, and Φ_C ={$t\geq$0, $t \leq$ 3}. Every node consists of two parts: the first part, above the dashed line, contains the predicates over the system's discrete variables, the second part, below the dashed line, contains all time-constraints in time-invariant that hold on that node. For every edge (n,m) we write t:=0 for every timer t in $R(n,m)$.

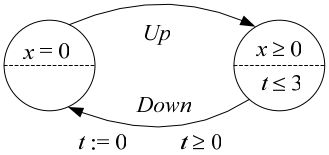

Figure 2. An example of TPD.

TPDs can be viewed as an extension of predicate diagrams. In the other direction, we may say that predicate diagrams are restricted TPDs. Particularly, when we eliminate all the components of TPDs that are related to real-time property, then we have predicate diagrams without fairness conditions. We call such a predicate diagram the untimed version of a TPD.

Definition 4. Let $T=(N,I,\delta,o,r,g,R)$ be a TPD over **P**, **A**, C and Φ_C. The predicate diagram $G=(N,I,\delta,o,\varnothing)$ over **P** and **A** is called the untimed version of T.

We now define *runs* and *traces* through a TPD as the set of behaviors that correspond to runs satisfying the node and edge labels.

Definition 5 Let $T=(N,I,\delta,o,r,g,R)$ over **P**, **A**, C and Φ_C as defined. A run of T is ω-sequence $\varphi = (s_0,n_0,A_0,\Delta_0)$ (s_1,n_1,A_1,Δ_1) … of quadruples where si is a state, $n_i \in N$ is a node, $A_i \in$ **A** $\cup$ {τ, *tick*} is an action and Δ_i is a real number such that all of the following conditions hold:

- $n_0 \in I$ is an initial node.
- $s_0|[c]| = 0$ holds for every $c \in C$.
- For every $i \in \aleph$ hold the following conditions:
  - $s_i\,|[n_i \wedge r(n_i)]|$.
  - Either $A_i \in \{\tau,tick\}$ and $n_i=n_{i+1}$ or $A_i \in$ **A** and $(n_i,n_{i+1}) \in \delta_{Ai}$.
  - If $A_i \in$ **A** and $(t, \prec) \in o(n_i,n_{i+1})$, then $s_{i+1}|[t]| \prec s_i|[t]|$.
  - If $A_i \in \{\tau,tick\}$ then $s_{i+1}|[t]| \preceq s_i|[t]|$ holds whenever $(t, \prec) \in o(n_i,m)$ for some $m \in N$.
  - If $A_i = \tau$ then $\Delta_i=0$ and $s_{i+1}|[c]| = s_i|[c]|$ holds for every c in C.
  - If $A_i=tick$ then $\Delta_i>0$ and $s_{i+1}\,|[c]| = s_i|[c]|+\Delta_i$ holds for every active timer c on n_i and $s_{i+1}|[c]| = s_i|[c]|$ holds for remaining timers.
  - If $A_i \in$ **A** then $\Delta_i=0$, $s_i|[g(n_i,n_{i+1})]|$ and $s_{i+1}|[c]| = 0$ holds for every c in $R(n_i,n_{i+1})$ and $s_{i+1}|[c]| = s_i|[c]|$ holds for remaining timers .

We write *runs*(T) to denote the set of runs of T. The set *tr*(T) of traces through T consists of all behaviors σ =

$s_0s_1\ldots$ such that there exists a run $\varphi = (s_0,n_0,A_0,\Delta_0)$ (s_1,n_1,A_1,Δ_1) … of T based on the states in σ.

4. Verification

Assume given a real-time specification *RTSpec* and a property F. We recall that in TLA formalism, the proof that *RTSpec* satisfies F can be considered as proving the validity of $RTSpec \Rightarrow F$. Following the approach of the verification of discrete systems using predicate diagrams, we split the proof into two steps: finding a TPD T such that every model of *RTSpec* is a trace through T and then proving that every trace through T is a model of F. The first step is done by considering node and edge labels of predicates on the concrete state space of *RTSpec* and reducing the trace inclusions to a set of first-order verification conditions that concern individual states and transitions. Thus, the first step is done deductively. On the other hand, the second step is done by regarding the node labels related to discrete properties, and probably the auxiliary invariants, as Boolean variables and then encoding the diagrams as a finite labeled transition system. The temporal properties of the system is then can be established by model checking.

4.1 Relating specifications and TPDs

To compare a specification and a TPD, we first have to assign meaning to the action names that appear in the diagram. We assume given a function α that assigns an action formula A every action name. Because no confusion is possible, we will leave this assignment implicit, and again write A instead of $\alpha(A)$ when referring to the formula assigned to the name A.

Recalling the general format of real-time specification in Equation 1, we put the time constraints explicitly on timed-bounded actions. In the context of TPDs, the situation is different, since we put time constraints on timers and not on actions. To overcome this, we define *bounded-actions* of TPDs. Basically bounded-actions are the same as timed bounded actions, which are actions on which we put time-constraints.

Definition 6. Let $G = (T,I,\delta,o,r,g,R)$ be a TPD over **P**, **A**, C and Φ_C. An action $A \in \mathbf{A}$ is called a *bounded-action* if there exists some timer $c \in C$ and two integer numbers d and e such that the following conditions hold:

- for every $n \in N$, a predicate of the form $c \le e$ is in $r(n)$ whenever $n \in En(A)$,
- for every $n,m \in N$, a predicate of the form $c \ge d$ is in $g(n,m)$ whenever $(n,m) \in \delta_A$ and
- for every $n,m \in N$, c is in $R(n,m)$ whenever $(n,m) \in \delta_A$ or $m \in En(A)$.

For a bounded action A, we call c, d and e its corresponding timer, lower-bound and upper-bound, and denote by $clk(A)$, $L(A)$ and $U(A)$, respectively.

Lemma 1. For some-bounded action A and for every node $n \in N$, $clk(A)$ is an active timer on n if $n \in En(A)$.

A TPD T *conforms* to a specification *RTSpec*, written *conf*(*RTSpec*, T), if every behavior that satisfies *RTSpec* is a trace through T. The following theorem essentially introduces a set of first-order ("local") verification conditions that are sufficient to establish conformance of a diagram to a real-time system specification in standard form.

Theorem 1. Let $RTSpec \equiv Init \wedge \Box[Next]_v \wedge RTNow(v) \wedge RT$ be a real time system specification and $T = (N,I,\delta,o,r,g,R)$ be a TPD over **P**,**A**,C and Φ_C as defined. We say that T conforms to *RTSpec* if the following conditions hold:

1. $\models Init \Rightarrow \bigvee_{n \in I} n$.
2. $\models n \wedge [Next]_v \Rightarrow n' \vee \bigvee_{(A,m):(n,m)\in\delta_A} \langle A\rangle_v \wedge m'$.
3. For $n,m \in \aleph$ and all $(t,\prec) \in o(n,m)$:
   a. $\models n \wedge m' \wedge \bigvee_{A:(n,m)\in\delta_A} \langle A \rangle_v \rightarrow t' \prec t$.
   b. $\models n \wedge [Next]_v \wedge n' \Rightarrow t' \preceq t$.
4. For every bounded-action A:
   a. $\models RTSpec \Rightarrow RTBound(A,v,clk(A),L(A),U(A))$,
   b. $\models n \Rightarrow$ ENABLED$\langle A\rangle_v$ holds for every node $n \in En(A)$,
   c. $\models n \Rightarrow \neg$ENABLED $\langle A\rangle_v$ holds for every node $n \notin En(A)$,
   d. $clk(A) \notin act(n)$ holds for every $n \in N$ such that $n \notin En(A)$,
   e. $clk(A) \notin \psi(g(n,m))$ holds for every $n,m \in N$ such that $(n,m) \notin \delta_A$ and
   f. $clk(A) \notin R(n,m)$ holds for every $n,m \in N$ such that $(n,m) \notin \delta_A$ and $m \in En(A)$.
5. $\models \bigwedge_{c\in C} TInit(c) \Rightarrow r(n)$ holds for every $n \in I$.
6. For every bounded action $A \in \mathbf{A}$ and for every $n,m \in N$:
   a. if $(n,m) \in \delta_A$ or $m \notin En(A)$ then
      $\models r(n) \wedge clk(A) \ge L(A) \wedge clk(A)' = 0 \wedge$
      $\bigwedge_{A_1:clk(A_1)\in act(m)} clk(A_1)' \le U(A_1) \Rightarrow r(m)'$.
   b. otherwise,

$$|= r(n) \wedge \bigwedge_{A_1:clk(A_1)\in act(n)} (clk(A_1)' \geq clk(A_1) \wedge clk(A_1)' \leq U(A_1) \Rightarrow r(n)').$$

The first three conditions in Theorem 1 are inherited from the conformance theorem of predicate diagrams. Those conditions are related to the discrete properties of the system. Conditions 4a-4f are related to the bounded-actions in the diagram and the two last conditions are related to the time-invariants in the diagrams. Condition 5 ensures that the time-invariant of every initial node is implied by the initial condition of every timer. Condition 6 guarantees that the time-invariants always agree with the changes of the concrete values of the timers whenever a transition is taken. In particular, the condition 6a requires that for every time-bounded action A, its corresponding timer should be reset to 0 whenever A is taken or it is not enabled at the next state and the value of every active timer on the next state is less than or equal to its corresponding upper-bound; and for the other cases condition 6b requires that the value of each action timer on the next state should be greater than or equal to its current value and less than or equal to its upper bound.

Theorem 1 can be used to show that the TPD of Figure 2 conforms to the specification *Loop* in Figure 1. For example, we have:

- $Init \Rightarrow x = 0 \vee x > 0$.
- $x = 0 \wedge [Next]_v \Rightarrow x'=0 \vee \langle Up \rangle_v \wedge x' > 0$.
- $x > 0 \wedge [Next]_v \Rightarrow x'>0 \vee \langle Down \rangle_v \wedge x' = 0$.
- $Loop \Rightarrow RTBound(Down,x,t,0,3)$.
- $t \leq 3 \wedge t \geq 0 \wedge t' = 0 \wedge true \Rightarrow t' \leq \infty$.
- $t \leq 3 \wedge t \geq t' \wedge t' \leq 3 \Rightarrow t' \leq 3$.

4.2 Model checking TPDs

For the proof that all traces through a TPD satisfy some property F, the TPD is viewed as a finite transition system that is amenable to model checking.

Two approaches for model checking TPDs are proposed. First, if the quantitative aspect of times doesn't come into account, it is enough to model check their untimed versions which are predicate diagrams.

However, whenever we have to consider the quantitative aspect of times for proving the properties we want to verify, we will use some existing real-time system model-checker for verifying TPDs. For example, we can use Kronos, which is a software tool built with the aim of assisting designers of real-time systems to verify whether their designs meet the specified requirements [6]. To do that, we first translate our diagrams into the input of Kronos, which are timed automata with some additional information.

Definition 7. A timed automaton Γ is given by a tuple (Q, Q_0, **X**, **I**, P, Λ) where

- Q is a set of locations,
- $Q_0 \subseteq Q$ is a set of initial locations,
- **X** is a finite set of clocks,
- $\mathbf{I} : Q \rightarrow \Theta(\mathbf{X})$ is a mapping from locations to clock constraints, called the location invariant,
- P is a function associates with each location a set of atomic propositions.
- $\Lambda \subseteq Q \times \Sigma \times \Theta(\mathbf{X}) \times 2^{\mathbf{X}} \times Q$ is a set of transition, and

The 5-tuple (q, a, θ, λ, q') corresponds to a transition from location q to location q' labeled with a, a constraint θ and a set of atomic propositions $P(q)$ that specify when transition is enabled and a set of clocks $\lambda \subseteq \mathbf{X}$ that are reset when the transition is executed.

Given a TPD T over **P**, **A**, C, and Φ_C, the translation from T to some timed-automata can be done by using this following construction.

Construction 1. Let $T=(N,I,\delta,o,r,g,R)$ be a TPD over **P**, **A**, C, and Φ_C. Let $\mathbf{A}^n$ be a set containing action names and κ: $\{1..|N|\} \rightarrow \aleph$ be an injective function which associates every node in N with some natural number. One can construct the corresponding timed automaton tuple (Q, Q_0, **X**, **I**, P, Λ) as follows:

- $Q = \{q_1,\ldots,q_{|N|}\}$.
- $Q_0 = \{q_i : \kappa(i) \in I\}$.
- $\mathbf{X} = C$.
- For every $i \in 1..|N|$, $P(q_i)=\kappa(i) \wedge r(\kappa(i)) \wedge \mathbf{I}(q_i) = r(\kappa(i))$.
- For every $(n,m) \in \delta$ and for every $A \in \mathbf{A}$, there exists some tuple $(q_i,A,\lambda,\theta,q_j)$ in Λ such that $\kappa(i) = n$, $\kappa(j) = m$, $\lambda = r(n,m)$ and $\theta = g(n,m)$.
- For every $n \in N$, there exists some tuple of the form $(q_i,a,\lambda,\theta,q_i)$ in Λ such that $\kappa(i) = n$, $a=\{tick,\tau\}$, $\lambda= R(n,n)$ and $\theta=g(n,n)$.

Theorem 2. Let $T=(N,I,\delta,o,r,g,R)$ be a TPD over **P**, **A**, C, and Φ_C as defined and let $\Gamma=(Q,Q_0,\mathbf{X},\mathbf{I},P,\Lambda)$ be the resulted automaton from Construction 1 over T. For every run through T, $\rho=(s_0,n_0,A_0,\Delta_0)$ $(s_1,n_1,A_1,\Delta_1)\ldots$, there exists a run of Γ: $\theta = (q_0,\nu_0) \xrightarrow{(w_0,\varphi_0)} (q_1,\nu_1) \xrightarrow{(w_1,\varphi_1)} \ldots$ such that $\kappa(i) = n_i$ and $s_i|[C]|=\nu_i(\mathbf{X})$ holds for every $i \in \aleph$.

5. An example: Fischer's protocol

As illustration we take the Fischer's mutual exclusion protocol which is a well-known and well-studied by

researchers in the context of real-time verification. We take the simplified version which only two processes in the protocol.

$$
\begin{aligned}
\mathit{Init} &\equiv x = 0 \wedge pc_1 = 0 \wedge pc_2 = 0 \\
\mathit{Try}_1 &\equiv x = 0 \wedge pc_1 = 0 \wedge pc_1' = 1 \wedge x' = x \wedge pc_2' = pc_2 \\
\mathit{Set}_1 &\equiv pc_1 \wedge x'=1 \wedge pc_1' = 2 \wedge pc_2' = pc_2 \\
\mathit{Enter}_1 &\equiv pc2 \wedge x' = x \wedge pc_2'=pc_2 \wedge \\
&\quad ((x=1 \wedge pc_1'=3) \vee (x\neq 1 \wedge pc_1'=0)) \\
\mathit{Exit}_1 &\equiv pc_1 = 3 \wedge pc_1'=0 \wedge x'=0 \wedge pc_2' = pc_2 \\
\mathit{Try}_2 &\equiv x = 0 \wedge pc_2 = 0 \wedge pc_2' = 2 \wedge pc_1' = pc_1 \\
\mathit{Set}_2 &\equiv pc_2 =1 \wedge x' = 2 \wedge pc_2' = 2 \wedge pc_1' = pc_1 \\
\mathit{Enter}2 &\equiv pc_2 = 2 \wedge x'=x \wedge pc_1'= pc_1 \wedge \\
&\quad ((x = 2 \wedge pc_2'=3) \vee (x\neq 2 \wedge pc_2'=0)) \\
\mathit{Exit}_2 &\equiv pc_2 = 3 \wedge pc_2' = 0 \wedge x' = 0 \wedge pc_1' = pc_1 \\
\mathit{Next} &\equiv \mathit{Try}_1 \vee \mathit{Set}_1 \vee \mathit{Enter}_1 \vee \mathit{Exit}_1 \vee \\
&\quad \mathit{Try}_2 \vee \mathit{Set}_2 \vee \mathit{Enter}_2 \vee \mathit{Exit}_2 \\
v &\equiv \langle x, pc_1, pc_2 \rangle \\
\mathit{Fischer} &\equiv \mathit{Init} \wedge \Box\,[\mathit{Next}]_v \wedge \mathit{RTNow}(v) \wedge \\
&\quad \mathit{RTBound}(\mathit{Set}_1, v, s_1, 0, D) \wedge \\
&\quad \mathit{RTBound}(\mathit{Enter}_1, v, t_1, E, \infty) \wedge \\
&\quad \mathit{RTBound}(\mathit{Set}_1, v, s_2, 0, D) \wedge \\
&\quad \mathit{RTBound}(\mathit{Enter}_2, v, t_2, E, \infty)
\end{aligned}
$$

Figure 3. Fischer's protocol.

The system is composed by a set of 2 timed processes, P_1 and P_2 plus a shared variable x. Each process P_i behaves as follows: after remaining idle for some time, it checks whether the common resource is free (test $x = 0$) and if so, before D time units sets x to i. Then it waits at least for E time units and, making sure that x is still equal to i, enters the critical section. If x is not equal to i (meaning that some other process has requested access) then process i has to retry later. The module contains the specification of the protocol and some properties to be verified is given in Figure 3.

For every process P_i we associate a variable pc_i which represents its control state. The value of pc_i is equal to one of the integer 0,1,2 or 3. The initial predicate *Init* asserts that pc_i is equal to 0 for each process i, so the processes start with control at statement 0.

We need to verify that, with suitable conditions on D and E there will never be more than one process in its critical section. This property can be expressed as a formula:

$$\mathit{Fischer} \Rightarrow \Box\neg(pc_1 = 3 \wedge pc_2 = 3). \qquad (3)$$

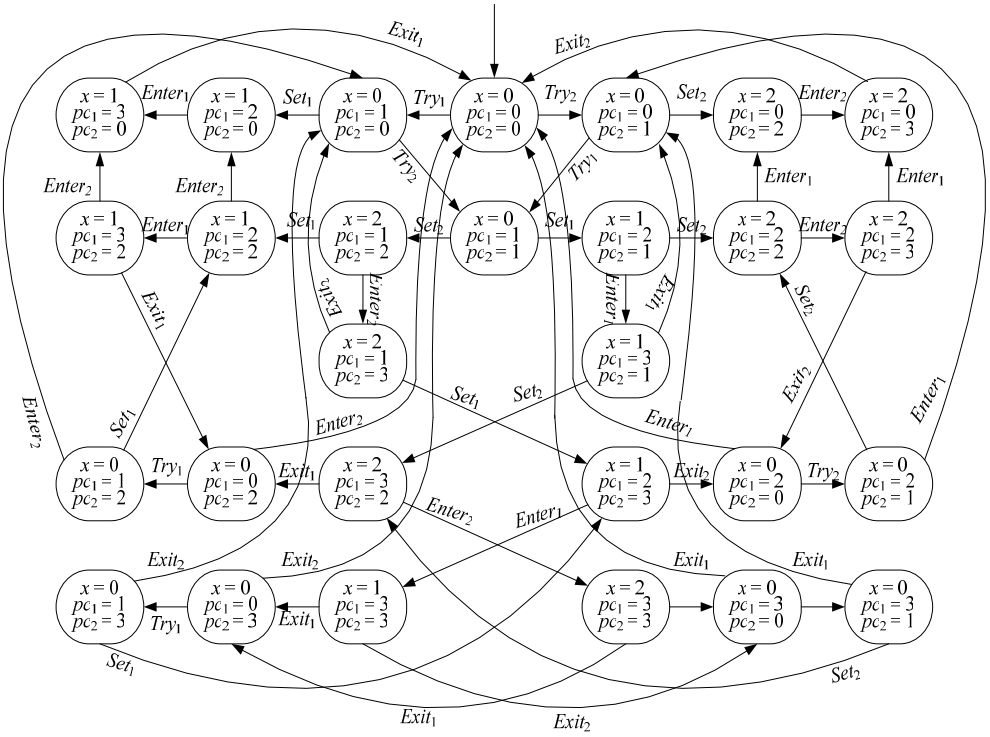

Figure 4. Predicate diagram for Fischer's protocol.

Figure 4 depicts the untimed version of TPD for Fischer's protocol. We will generate a suitable TPD of this predicate diagram. Since in the specification Set_1, $Enter_1$, Set_2 and $Enter_2$ appear as time-bounded actions, we intuitively treat those actions as bounded-actions in the TPD. For each bounded-action we set the corresponding clock, lower bound and upper bound as follows:

- $clk(Set_1) = s_1$, $L(Set_1) = 0$, $U(Set_1) = D$,
- $clk(Enter_1) = t_1$, $L(Enter_1) = E$, $U(Enter_1) = \infty$,
- $clk(Set_2) = s_2$, $L(Set_2) = 0$, $U(Set_2) = D$, and
- $clk(Enter_2) = t_2$, $L(Enter_2) = E$, $U(Enter_2) = \infty$

Starting with the initial node we traverse the diagram in order to determine whether those time-invariants still hold on each node or not. We do this by considering the bounded-actions that can be taken on every node. For example, if Set_1 is enabled then $s_1=0$ is changed to $s_1\geq 0$. The predicates $s_1=t_2$ and $s_2=t_1$ might be changed accordingly. The result is shown in Figure 6 (the numbering on some nodes will be used later). The resulted diagram, again, represents an approximation of the Fischer specification. However, we still cannot prove the mutual exclusion property.

We can strengthen the second diagram, by using assumptions on D and E. We consider two cases: $D < E$ and $D \geq E$. Based on these assumptions, we refine our diagram in Figure 6 by eliminating transitions that do not satisfy those conditions. The elimination process can be done using Simplify algorithm in Figure 7.

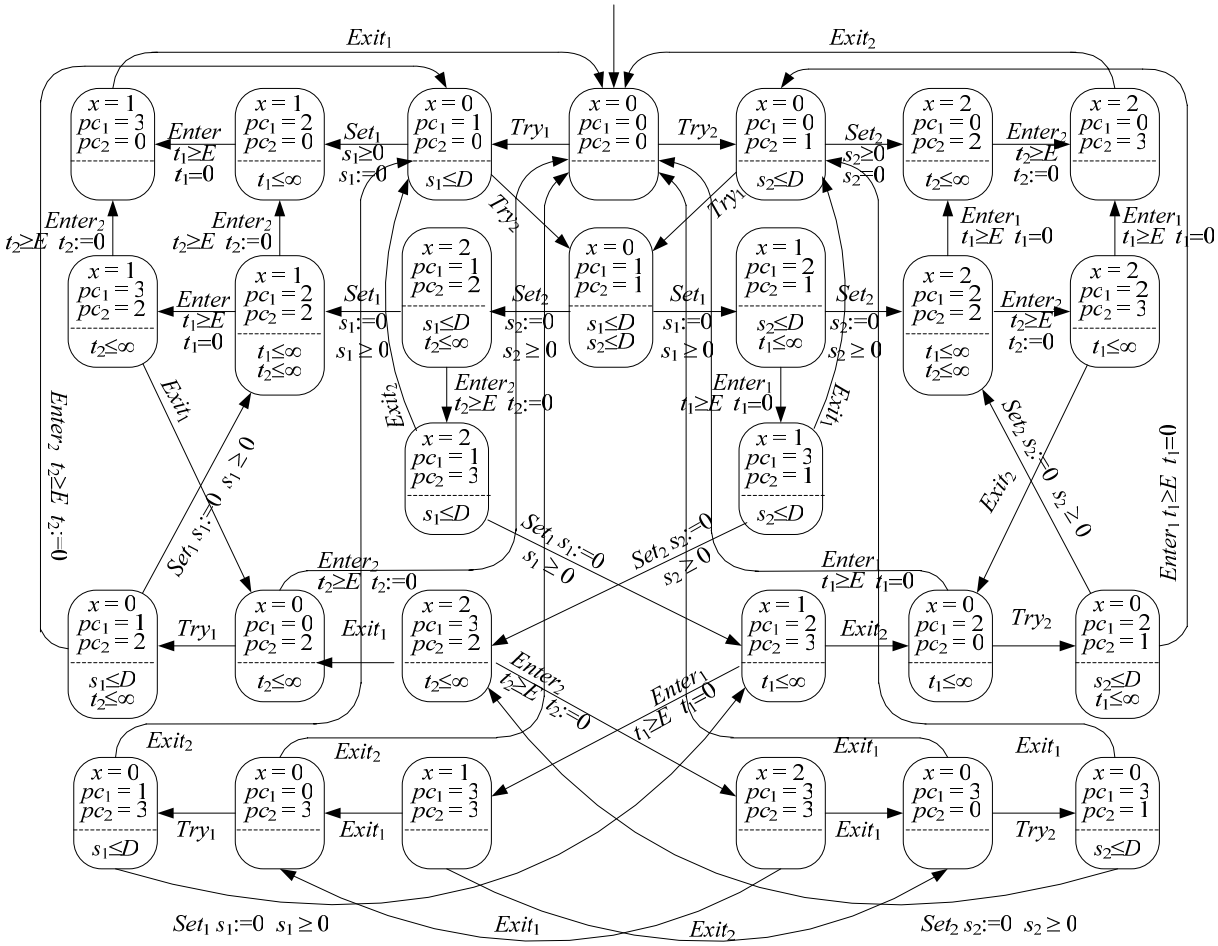

Figure 5. First TPD for Fischer's protocol.

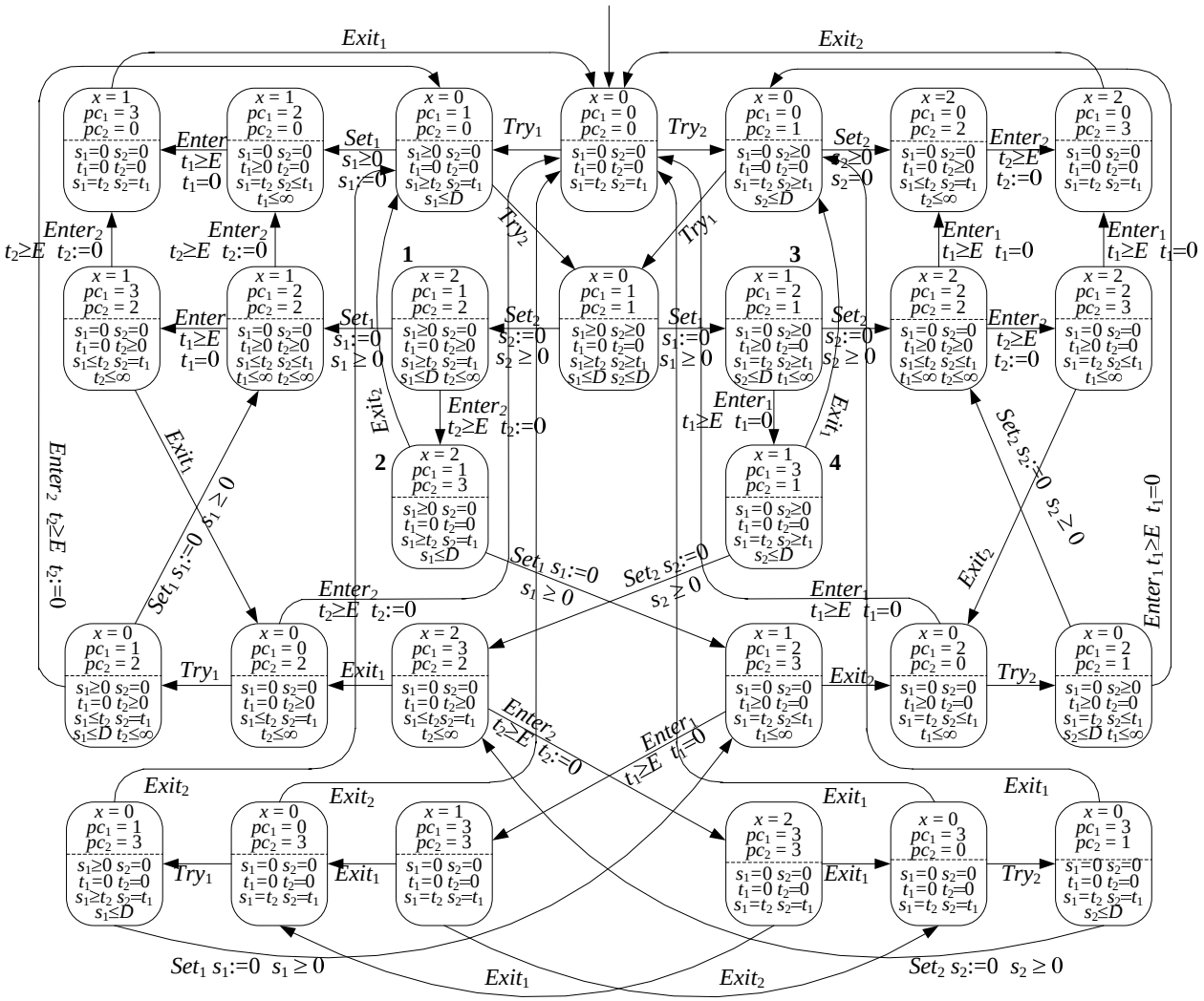

Figure 6. Second TPD for Fischer's protocol.

Algorithm Simplify
  if $n \notin visited$ **then**
    $visited = visited \cup \{n\}$
    for every $(n,m) \in \delta$
      if either condition 6a or 6b in Theorem is satisfied **then**
        $\delta_1 \equiv \delta_1 \cup \{(n,m)\}$
        Simplify (T,m, δ_1,$visited$)
      endif
    endfor
  endif

Figure 7 Simplify algorithm.

For every initial node n, we run the algorithm with TPD T, some node n and some set of edge Δ_1 as inputs. Starting with initial nodes, for every edge leading from this node, the algorithm checks whether it satisfies the required condition or not. If so, then the edge is stored and the algorithm continues to check the next edges. For example, consider node 1 and node 2 in Figure 6. The corresponding edge does not satisfy either condition 6a, since it is never the case that $s_1 \geq t_2 \wedge s_1 \leq D \wedge t_2 \geq E$ is true while $D < E$, or condition 6b, since in this case node 1 and node 2 are different nodes. The situation is the similar with node 3 and node 4. Running this algorithm over our second diagram and assumption $D<E$, we have the third diagram shown in Figure 8.

For the assumption $D \geq E$, the resulted diagram has the same structure with the one in Figure 6. As conclusion, the protocol satisfies mutual exclusion property if $D < E$.

Since we don't consider the quantitative aspect of time, we may work with the untimed version of TPD shown in Figure 8 and then model-check the resulted diagram as explained in Section 4.

6. Conclusion and related work

A class of diagrams for the verification of real-time systems is presented in this paper. The a class of diagrams, called Timed predicate diagrams (TPDs), is a variant of predicate diagrams proposed by Cansell et.al [3]. A TPD can be viewed as a predicate diagram equipped with some component in order to constraint the time. Like predicate diagrams, the verification of TPDs is a combination of deduction and algorithmic techniques.

This method is applied on the verification of mutual-exclustion property of Fischer's protocol. It has been shown that the construction of TPD for this protocol can be done by starting from the untimed version (without considering the time aspect) and then refining the diagram step by step until we get the of an appropriate one. The structure of TPDs, that allows us to work with the untimed version of TPD, made the verification easier. It is proven that the protocol satisfies the expected mutual property.

More about the TPDs can also be found in [7].

Many models for reasoning real-time systems have been proposed. The approach to real-time presented in Manna et.al. [8] and [1] is based on the computational model of *Timed Transition Systems* (TTS) in which time itself is not explicitly represented but it is reflected in a time stamp affixed to each state in a computation of a TTS. In [9], Kesten et.al. introduced a computation model for real-time systems called *Clocked Transition Systems* (CTS), which is a development of TTS. This model represents time by a set of system variables called clocks (timers) which increase uniformly whenever time progresses, but can be set to arbitrary values by system (program) transitions.

The use of diagrams in verification of real-time systems can be found, for example in [9]. In their approach, they use a special rule for proving a class of property, such as invariant and response properties. Every rule is associated with verification diagrams for real-time systems.

The generation of the TPDs for the Fischer's protocol is done manually. Boujjanni et.al [10] have successfully generated the invariants for the Fischer's protocol automatically using TReX, a tool for reachability analysis of complex systems.

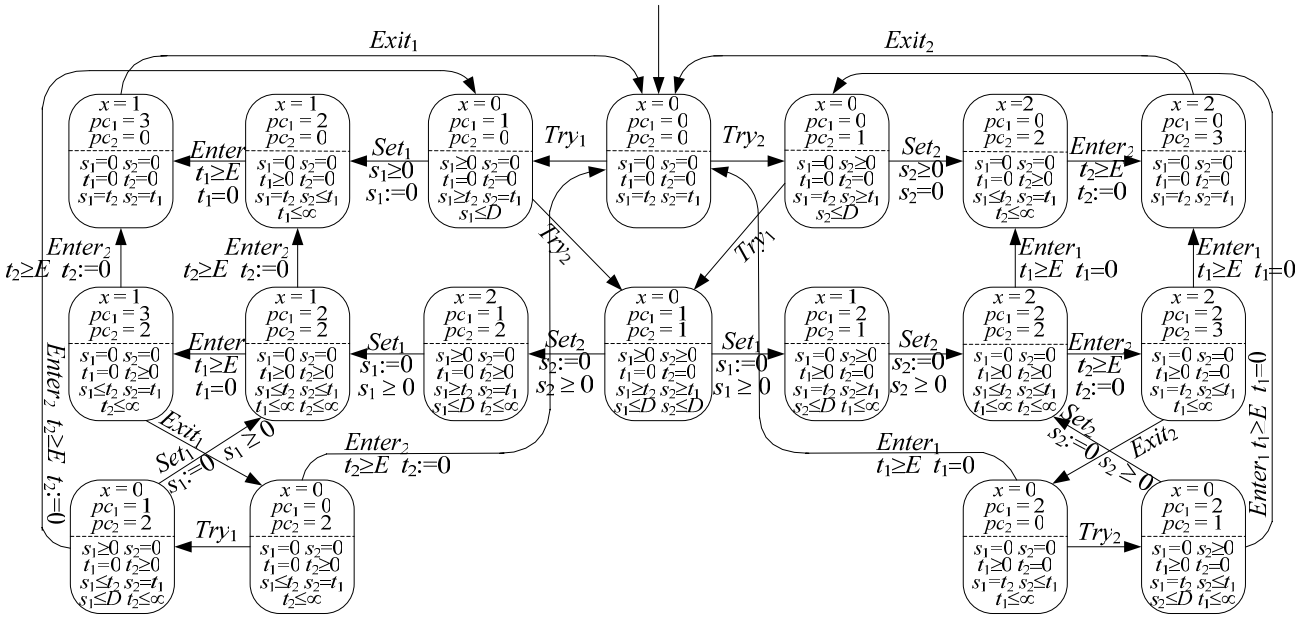

Figure 8. Third TPD for Fischer's protocol.

References

[1] Z. Manna and A. Pnueli, “Models for reactiviy”. Acta Informatica, 30:609-678, 1993.
[2] Henny B. Sipma, “Diagram-based verification of discrete, reactive and hybrid systems”, PhD Thesis, Dept. of Computer Science, Standford University, 1999.
[3] D. Cansell, D. Méry and S. Merz, , “Predicate diagrams for the verification of reactive systems”, Intl. Conf. on Integrated Formal Methods (IFM 2000), vol 1945 of Lecture Notes in Computer Science, Springer-Verlag, 2000.
[4] L. Lamport, “The Temporal Logic of Actions”, ACM Transactions on Programming Languages and Systems, 16(3):872-923, May, 1994.
[5] R. Alur, “Timed automata”, NATO ASI Summer School on Verification of Digital and Hybrid Systems, 1998.
[6] S. Yovine, “KRONOS: A verification tool for real-time”, International Journal of Software Tools for Technology Transfer, vol. 1, NBer. 1+2, p. 123-133, Springer-Verlag, 1997.
[7] C.E. Nugraheni, “Predicate diagrams as basis for the verification of reactive systems”, PhD Thesis, Institut für Informatik, University of Munich, Germany, 2004.
[8] T. Henzinger, Z. Manna, and A. Pnueli. ”Temporal proof methodologies for Timed Transition Systems”. Information and Computation, 112(2):273-337. 1994.
[9] Y. Kesten, Z. Manna, and A. Pnueli. ”Verification of Clocked and Hybrid Systems”. In G. Rozenberg and F.W. Vaandrager, editors. Lectures on Embedded Systems, vol. 1494 of Lecture Notes in Computer Science, pages 4-73. Springer-Verlag, 1998.
[10] TReX examples: Fischer’s protocol in http://ww_verimag.imag.fr/~annichin/trex/demos/fischer.html.

Cecilia E. Nugraheni received the B.S. and M.S. degrees in Informatics Engineering from Bandung Institute of Technology in 1993 and 1997, respectively. In 2004 she got her PhD degree from Institut für Informatik, University of Munich, Germany. She is now an academic staff at Computer Science Dept., Parahyangan Catholic University, Bandung, Indonesia.